LEARNING LOCAL OCEAN FLOWS USING NEURAL ODES

JASLEEN DHANOA [JKDHANOA@SEAS], MANASA SATHYAN [MSATHYAN@SEAS], URARA KONO [URARAK@SEAS]

ABSTRACT. Generating predictive models of local flow patterns in the ocean helps in designing efficient path planners for autonomous surface vehicles. Deep learning approaches that only take the past trajectory as input and predict the next position under-perform as they do not leverage local flow information. We propose using Neural Ordinary Differential Equations (NODE) to build an underlying model of local geophysical flows using the trajectory data. The model learns the vector field of the underlying flow from the trajectory data which can then be used to predict future trajectory. In this paper, we train Neural ODEs (NODEs), Augmented Neural ODEs (ANODEs), Knowledge-based Neural ODEs (KNODEs) on time-invariant Single and Double Gyre dynamical systems. We then evaluate and compare the performance of NODE, ANODE, and KNODE on qualitative and quantitative metrics and show that we can get good models of the local flow for prediction.

1. INTRODUCTION

1.1. **Problem Statement.** The study of ocean flow dynamics has many important applications, including improving our understanding of the Earth's climate and weather patterns, managing marine resources, and developing new technologies such as more efficient shipping routes and renewable energy systems. In order to design efficient path planners for autonomous surface vehicles or perform energy efficient navigation flows, it is necessary to generate predictive models of local ocean flow patterns. In this project, we approach the task of predicting flow patterns or vector fields over time by using neural ordinary differential equations (NODEs). NODEs have several advantages for studying ocean flows, including improved accuracy, efficient training, scalability, and interpretability. They are particularly effective at modeling complex and dynamic systems like ocean currents and do not require as many parameters or discretized data as traditional neural networks. NODEs are also highly scalable and can be more interpretable, making them a valuable tool for understanding ocean flow dynamics. We consider time-invariant single and double gyre ocean dynamics to evaluate and train our NODE-based models. As shown in Fig. 1, gyres are large, circular currents in the oceans that are caused by the Earth's rotation and wind patterns. Single gyres are characterized by a single, large circular current that flows in one direction. Double gyres, on the other hand, consist of two smaller gyres that flow in opposite directions and are separated by an area of weaker currents. Learning the formation and behavior of single and double gyres is helpful in predicting surface vehicle trajectories.

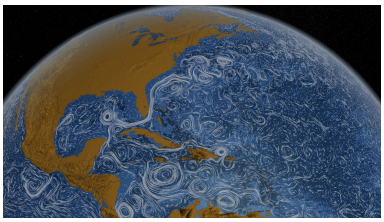


FIGURE 1. Flow patterns including gyres in the ocean as observed by NASA satellites. Image from www.nasa.gov.

1.2. **Contributions.** We train a standard NODE, Augmented NODE (ANODE), and Knowledge-based NODE (KNODE) to learn the underlying time-invariant single and double gyre dynamics for the task of trajectory prediction. We evaluate and compare the performance of these three models on both Single and Double Gyres and provide a qualitative (plots of true vs predicted trajectories and vector fields) and quantitative (error metrics such as Absolute Trajectory Error and RMSE between true and predicted trajectories) analysis.

2. BACKGROUND

Neural ODEs [1] learn the underlying ordinary differential equations (ODEs) of a system instead of just the solution to the next state given a history of past states. Neural ODEs implicitly enforce continuity which leads to a better model for our application as local flows in oceans are continuous. They can also be more efficient to train, as they require fewer parameters than traditional neural networks. **Augmented neural ordinary differential equations (ANODEs)** [2] are a variant of neural ordinary differential equations (NODEs) that are used to model complex systems with additional inputs or constraints. NODEs augment the space on which the ODE is solved, allowing the model to use additional dimensions to learn more complex functions using simpler flows. **Knowledge-embedding Neural ODEs (KNODE)** [3] are a variant of neural ordinary differential equations (NODEs) that embed first principles knowledge or prior knowledge of the underlying dynamical system into NODEs. This is shown to improve the performance of the model.

Dynamics Equations for Double Gyres -The velocity of flow in single or double gyres are represented by eq.1. When $\epsilon = 0$, the flow is time-invariant, while for $\epsilon \neq 0$, the flow undergoes a periodic expansion and contraction in the x

direction. A approximately represents the amplitude of the vector field. ϵ gives the amplitude of the motions of the gyros in the x direction. $\omega/2\pi$ provides the oscillation.

$$\dot{x} = -\pi A \sin\left(\pi \frac{f(x,t)}{s}\right) \cos\left(\pi \frac{y}{s}\right) - \mu x, \dot{y} = \pi A \cos\left(\pi \frac{f(x,t)}{s}\right) \sin\left(\pi \frac{y}{s}\right) - \mu y,$$
$$f(x,t) = \epsilon \sin(\omega t) x^2 + (1 - 2\epsilon \sin(\omega t))x \tag{1}$$

3. Related Work

Getting predictive models of local flow patterns in the ocean helps in designing efficient path planners for autonomous surface vehicles. For example, [4, 5, 6, 7, 8] use level set method to compute the time-optimal path planning assuming that the flow vector field is known. For modeling dynamical systems, there have been data-driven approaches thanks to recent advances in machine learning and data analytics. [9, 10, 11] embed the state of the original system into delayed snapshots and represent the systems as time delay models. Recent machine learning techniques use recurrent neural networks (RNNs) such as LSTM and reservoir computers to predict two-dimensional fluid flows [12] and a one-dimensional Kuramoto-Sivashinsky model [13]. However, these method lose the continuity of the original system. Recently, some works have been focused on explicitly extracting differential equations of a dynamical system. [14] and [15] determine the best combinations of basis functions of a vector field from predetermined candidates. Neural ODEs was introduced by [1], which helps to model the derivative of the state instead of the state directly, but it is limited to learning only stable dynamics. To overcome this problem, knowledge-based Neural ODEs were introduced by [3] , enabling them to learn continuous-time dynamical system.

4. Method

We use a time-invariant double gyre system with parameters listed in Table 1 A for generating the training data for the single gyre and double gyre systems. We use the double-gyre system as it is often used to describe large-scale re-circulation in the ocean [16] and is sufficiently expressive to see if we can learn a complex model. Since we are using the time invariant system, the vector field remains constant at any given time, given by equation eq.1. Our aim is to learn the underlying dynamical system given the trajectory data and and evaluate the performance of the Neural ODE in learning the underlying flow problem.

1. Training Dataset: The training data for the single gyre consists of a single trajectory sampled from the left gyre for 50 time steps. This training data is generated by integrating the true dynamics of the double gyre with a numerical integrator (4th Order Runge-Kutta) with a step size of 0.02. The initial position for this trajectory is chosen near the edge of the gyre so that in the next 50 time steps it covers almost the entire gyre due to the nature of flow propelling it. The entire trajectory is then discretized at a resolution of 1000 data points consisting of relative time stamps and corresponding position data(x,y) which is used for training. The training data for the double gyre consists of two such trajectories sampled from each of the two gyres. We then add gaussian noise $\mathcal{N}(0, 0.1)$ to the trajectories to simulate measurement noise. It is important to choose the appropriate discretization scale and noise so as to to preserve the dynamical system after discretization and adding the noise. Fig2 shows the training data used for training the Neural ODEs(NODE, ANODE and KNODE). Fig2 A) shows the training trajectory for the single gyre consisting of noisy trajectory data (in yellow) for 1000 datapoints superimposed on the true vector field and Fig2 B) shows the training trajectories for the double gyre consisting of noisy trajectory data (in blue and green) for 1000 datapoints each superimposed on the true vector field.

Double Gyre Parameters(Train)	Values
A	10
μ	0.04
ϵ	0
s	50
ω	2π

(A) Double-Gyre Parameters for Training

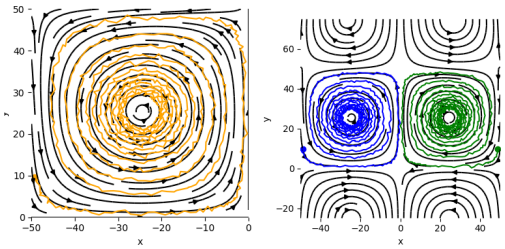
Double Gyre Parameters(Knowledge)	Values
A	5
μ	0.005
ϵ	0
s	50
ω	2π

(B) Double-Gyre Parameters fed as knowledge for KNODE

TABLE 1. Parameters of double gyre

Neural Network Layers	Dimensions
Linear 1	input dim =2(NODE, KNODE)/3(ANODE), output dim =128, followed by Tanh
Linear 2	input dim =128, output dim =64, followed by Tanh
Linear 3	input dim =64, output dim =2

TABLE 2. Neural Network architecture



(A) Training data for Single Gyre (B) Training data for Double Gyre

FIGURE 2. Training Data: A) Training data for Single Gyre (yellow) overlaid on true vector field (black) for time-invariant double gyre. B) Training data for Double Gyre (blue, green) overlaid on true vector field for time-invariant Double Gyre

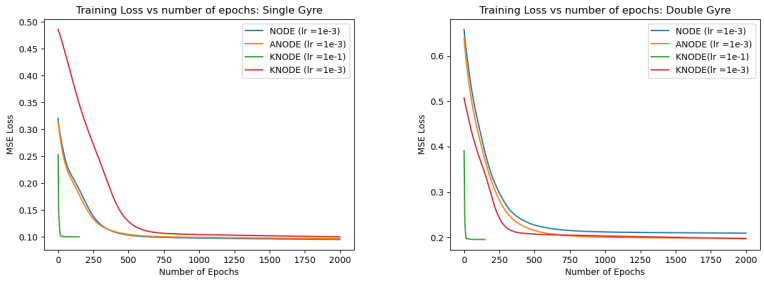
2. Neural Networks and Training: We implemented and trained Neural ODE, Augmented Neural ODE and Knowledge-based Neural ODEs for single gyre and double gyre. In order to compare the three methods, we used the same three layer neural network and Tanh non-linearity as given in Table 2 . We experimented with various non-linearities and neural network architectures and found that these parameters lead to the best performance among all the three types of neural ODEs. For Augmented Neural ODEs, we increase the dimension of input data (in addition to the two dimensions representing (x,y)) and initialize it with zeros. We experimented with different number of augmented dimensions and found that augmenting it by 1 helped learn the double gyre system the best. In order to use a larger value of augment dimension while preventing over fitting, we need a higher dimensional system and we need more data to train it as well. For Knowledge-based Neural ODEs, we used a hybrid model combining rudimentary knowledge about the system along with the neural network. The rudimentary knowledge of the system is a double gyre system with incorrect parameters as given in Table 1 B). The added knowledge has A = 5 which limits the maximum magnitude of the vector field to 16 and $\mu = 0.005$ which restricts the dissipation in the system.

Lookahead : A lookahead parameter is used to divide the training data into batches (lookahead ≥ 1). Instead of generating trajectories for 50 time steps and calculating the loss over the entire trajectory which results in instability during training, we take each point in the trajectory as the initial condition and generate trajectory for just the next few time steps dictated by the value of lookahead as done in [3]. As we increase the value of lookahead parameter, we get smoother trajectory results but we also get instability in training. All the results in this paper have been generated with a lookahead of 2.

Training Method : We trained all the neural networks with Mean Square Error loss and Adam Optimizer. We used a learning rate of 1e-3 and performed Gradient Descent. The training curves for these Neural Networks are shown in Fig 3. It is interesting to note that we could train KNODE with a learning rate of 0.1 for the first 50 iterations and 1e-2 for the rest of the iterations and it had a smaller training loss than when we trained at a learning rate of 1e-3 and the qualitative results for this model looked better as well. While training the Neural ODEs, we use a 4th Order Runge Kutta solver with step-size 0.02.

5. EXPERIMENTAL RESULTS

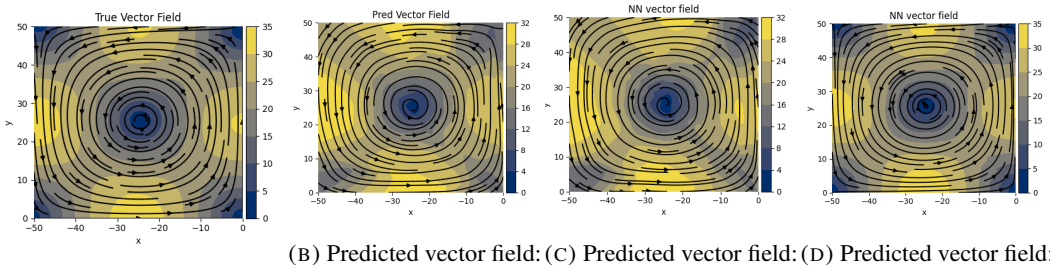
1. Training: Fig 4 show the converged learned vector fields by NODE, ANODE, and KNODE for a Single Gyre system [A-D] along with the Error Magnitude of learned vs true vector fields [E-G]. We see that KNODE has the least error in magnitude, with a maximum value of ~ 3.2 as opposed to ~ 24.0 for NODE *(please note the difference in scales in the error magnitude plots)*. Thus, KNODE performs better than NODE and especially in regions where it hasn't seen training data i.e. the edges and top left and bottom right corners. This trend is also seen for Double Gyre systems which



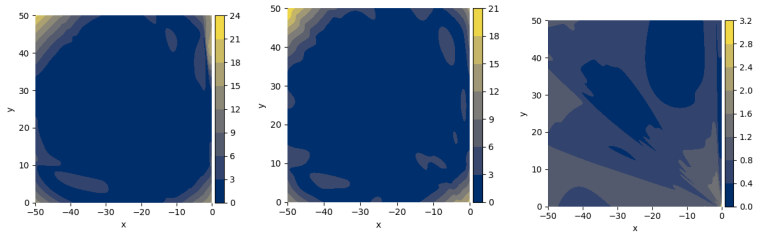
(A) Training data for single gyre (B) Training data for double gyre

FIGURE 3. Training Loss vs Number of Epochs: Single Gyre and Double Gyre

can be found in Section 7.2 of the Appendix. Plots showing the evolution of the learned vector fields at different epochs can be viewed in Section 7.3 of the Appendix.



(A) True vector field (B) Predicted vector field: NODE (C) Predicted vector field: ANODE (D) Predicted vector field: KNODE



(E) Magnitude of error in vector field: NODE (F) Magnitude of error in vector field: ANODE (G) Magnitude of error in vector field: KNODE

FIGURE 4. Learned Vector Field: True vector field (Fig A) vs predicted vector field for the Single Gyre system (Figs B-D). Figs (E-G) show the error in magnitude of vector field at each point within the domain of training data. *NOTE-different color bar scales for each plot*

2. Evaluation: We test our trained models on 1000 different trajectories. These 1000 trajectories are formed by randomly sampled 1000 initial conditions within the training domain and integrating them using 4th Order Runge kutta Solver with step size 0.02 for the duration of 1 timestep (Note than 1 time step is not just one step into the future as 50 seconds is discretized into 1000, so 1 timestep is equivalent to 20 steps for the solver.). We use Absolute Trajectory Error (ATE) and Root Mean Squared Error(RMSE) metrics to determine the performance of our learned model that is, to measure how much the predicted trajectory varies from the ground truth. Fig 5 represents a subset of the trajectories predicted by NODE, ANODE, and KNODE against the true trajectories for Single Gyre [A-D] and Double Gyre [E-H] Table 3 summarizes the evaluation metrics for all the three models on Single Gyre and Double Gyre. We observe that KNODE outperforms NODE and ANODE for both dynamical systems which is owed to the fed rudimentary knowledge helping the model learn better. This shows that by using some knowledge of a dynamical system to train a neural network, it is able to learn a much better representation of the underlying dynamics.

6. DISCUSSION

We implement Neural ODE (NODE), Augmented Neural ODE (ANODE) and Knowledge based Neural ODE (KNODE) for time-invariant Single and Double Gyre systems and compare their performance quantitatively and qualitatively.

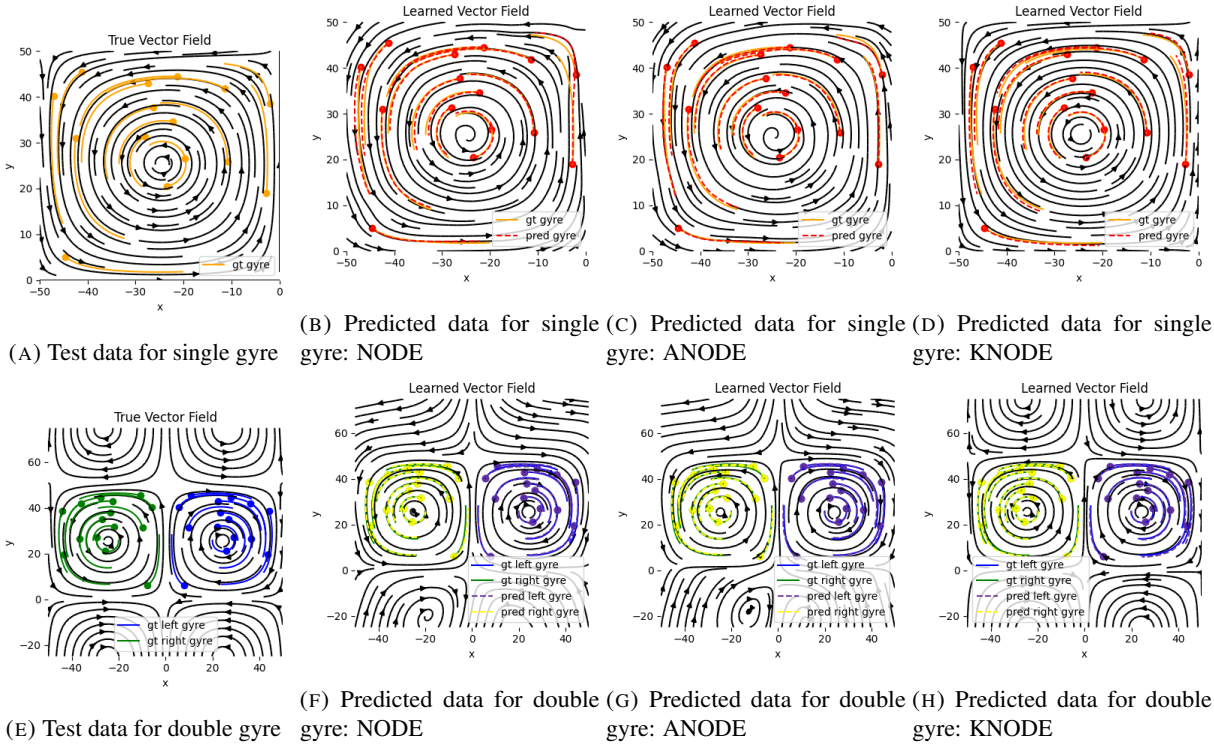
Model	ATE	RMSE
NODE	0.381	0.756
ANODE	0.373	0.723
KNODE	0.197	0.283

(A) Single Gyre

Model	ATE	RMSE
NODE	0.485	0.704
ANODE	0.461	0.706
KNODE	0.351	0.375

(B) Double Gyre

TABLE 3. Evaluation Metrics



(A) Test data for single gyre

(B) Predicted data for single gyre: NODE

(C) Predicted data for single gyre: ANODE

(D) Predicted data for single gyre: KNODE

(E) Test data for double gyre

(F) Predicted data for double gyre: NODE

(G) Predicted data for double gyre: ANODE

(H) Predicted data for double gyre: KNODE

FIGURE 5. Test Data: Testing data for Single Gyre (yellow)(A), Double Gyre (blue, green)(E). The test data consists of the initial positions (shown by the dots) and the expected trajectories for 1 time step. These trajectories are overlaid on the true vector field (black). (B,C,D,F,G,H) Predicted trajectories from NODE, ANODE, KNODE are are overlaid on the predicted vector field (black) for time-invariant Single Gyre and Double Gyre systems respectively.

KNODE performs the best at modeling the underlying vector field. The basic knowledge of the dynamical system introduced during training gives KNODE a head start and helps it to overcome the error in the dynamics. It is also able to perform well when trained on noisy data and generalize in regions where it hasn't encountered training data. So, if we have prior knowledge about the system we can utilise it to learn a good local model of the vector fields.

Future work: Although we tried to learn *time-varying* Single and Double Gyre system by incorporating time into NODE, KNODE, ANODE training, we were not able to learn good dynamical representations. Appendix D presents the results of learning double gyros with NODEs. Given more time, we would train our time-varying models with more data for each time instant via multiple trajectories. We would also like to derive analytical solutions for the learnt model using SINDy [17] and compare them with the true model.

ACKNOWLEDGMENT

The authors would like to acknowledge Pratik Chaudhari, Tom Z. Jiahao, KongYao Chee and M. Ani Hsieh for their insightful advice on our work.

REFERENCES

[1] Ricky T. Q. Chen et al. "Neural Ordinary Differential Equations". In: (June 2018). URL: http://arxiv.org/abs/1806.07366.

[2] Emilien Dupont, Arnaud Doucet, and Yee Whye Teh. "Augmented Neural ODEs". In: *Advances in Neural Information Processing Systems*. Ed. by H. Wallach et al. Vol. 32. Curran Associates, Inc., 2019. URL: https://proceedings.neurips.cc/paper/2019/file/21be9a4bd4f81549a9d1d241981cec3c-Paper.pdf.

[3] Tom Z. Jiahao, M. Ani Hsieh, and Eric Forgoston. "Knowledge-based learning of nonlinear dynamics and chaos". In: *Chaos* 31 (11 Nov. 2021). ISSN: 10897682. DOI: 10.1063/5.0065617.

[4] T. Lolla et al. "Path planning in time dependent flow fields using level set methods". In: IEEE, May 2012, pp. 166–173. ISBN: 978-1-4673-1405-3. DOI: 10.1109/ICRA.2012.6225364. URL: http://ieeexplore.ieee.org/document/6225364/.

[5] Tapovan Lolla, Patrick J. Haley, and Pierre F.J. Lermusiaux. "Time-optimal path planning in dynamic flows using level set equations: realistic applications". In: *Ocean Dynamics* 64 (10 Oct. 2014), pp. 1399–1417. ISSN: 16167228. DOI: 10.1007/s10236-014-0760-3.

[6] Venkata Ramana Makkapati, Jack Ridderhof, and Panagiotis Tsiotras. "Reachability-based Covariance Control for Pursuit-Evasion in Stochastic Flow Fields". In: American Institute of Aeronautics and Astronautics Inc, AIAA, 2022. ISBN: 9781624106316. DOI: 10.2514/6.2022-1382.

[7] Wei Sun et al. "Multiple-pursuer/one-evader pursuit-evasion game in dynamic flowfields". In: *Journal of Guidance, Control, and Dynamics* 40 (7 2017), pp. 1627–1637. ISSN: 15333884. DOI: 10.2514/1.G002125.

[8] Sri Venkata and Tapovan Lolla. "Path Planning in Time Dependent Flows using Level Set Methods". 2010.

[9] Floris Takens. *Detecting strange attractors in turbulence*. 1981. DOI: 10.1007/BFb0091924.

[10] Johan Paduart et al. "Identification of nonlinear systems using Polynomial Nonlinear State Space models". In: *Automatica* 46 (4 Apr. 2010), pp. 647–656. ISSN: 00051098. DOI: 10.1016/j.automatica.2010.01.001.

[11] Eric A Wan. *Time series prediction by using a connectionist network with internal delay lines*. Time Series Prediction, 1994. URL: https://www.researchgate.net/publication/2454592.

[12] Maan Qraitem et al. "Bridging the gap: Machine learning to resolve improperly modeled dynamics". In: *Physica D: Nonlinear Phenomena* 414 (Dec. 2020), p. 132736. ISSN: 01672789. DOI: 10.1016/j.physd.2020.132736.

[13] Alexander Wikner et al. "Combining machine learning with knowledge-based modeling for scalable forecasting and subgrid-scale closure of large, complex, spatiotemporal systems". In: *Chaos: An Interdisciplinary Journal of Nonlinear Science* 30 (5 May 2020), p. 053111. ISSN: 1054-1500. DOI: 10.1063/5.0005541.

[14] Steven L. Brunton, Joshua L. Proctor, and J. Nathan Kutz. "Discovering governing equations from data by sparse identification of nonlinear dynamical systems". In: *Proceedings of the National Academy of Sciences* 113 (15 Apr. 2016), pp. 3932–3937. ISSN: 0027-8424. DOI: 10.1073/pnas.1517384113.

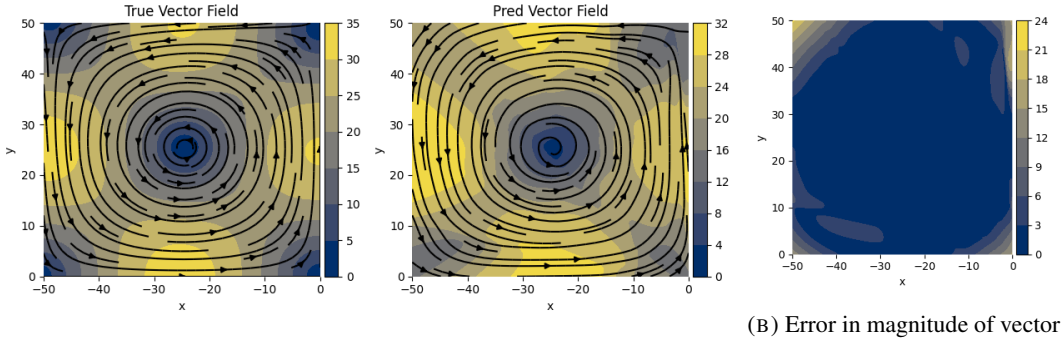
[15] Abd AlRahman R. AlMomani, Jie Sun, and Erik Bollt. "How entropic regression beats the outliers problem in nonlinear system identification". In: *Chaos: An Interdisciplinary Journal of Nonlinear Science* 30 (1 Jan. 2020), p. 013107. ISSN: 1054-1500. DOI: 10.1063/1.5133386.

[16] Dhanushka Kularatne, Eric Forgoston, and M. Ani Hsieh. "Exploiting Stochasticity for the Control of Transitions in Gyre Flows". In: Robotics: Science and Systems Foundation, June 2018. ISBN: 978-0-9923747-4-7. DOI: 10.15607/RSS.2018.XIV.060. URL: http://www.roboticsproceedings.org/rss14/p60.pdf.

[17] Dynamicslab. URL: https://github.com/dynamicslab/pysindy.

APPENDIX A. LEARNED VECTOR FIELDS OF SINGLE GYRES

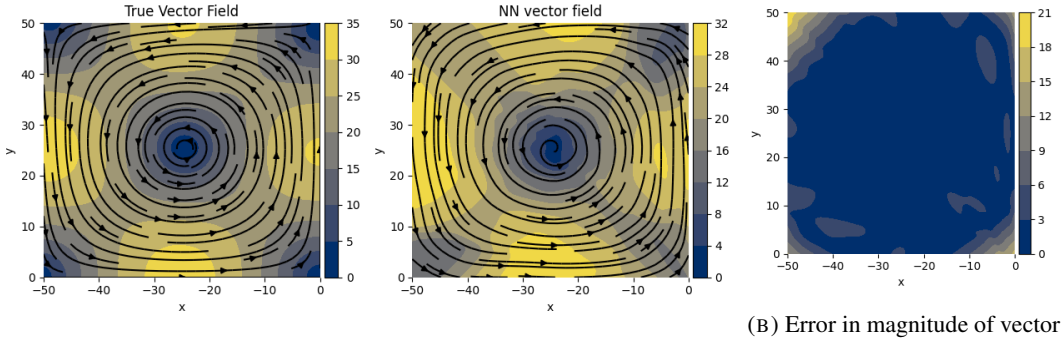
Figs (5-7) shows the learned vector fields for NODEs, ANODEs, and KNODEs respectively.



(A) True vector field vs predicted vector field

(B) Error in magnitude of vector field

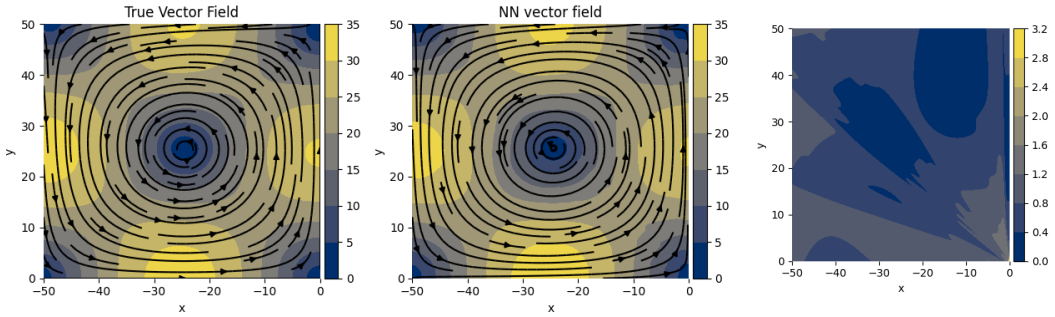
FIGURE 6. NODE: Learned Vector Field: A) True vector field vs predicted vector field . B) Error in magnitude of vector field at each point within domain of training data



(A) True vector field vs predicted vector field

(B) Error in magnitude of vector field

FIGURE 7. ANODE: Learned Vector Field: A) True vector field vs predicted vector field . B) Error in magnitude of vector field at each point within domain of training data



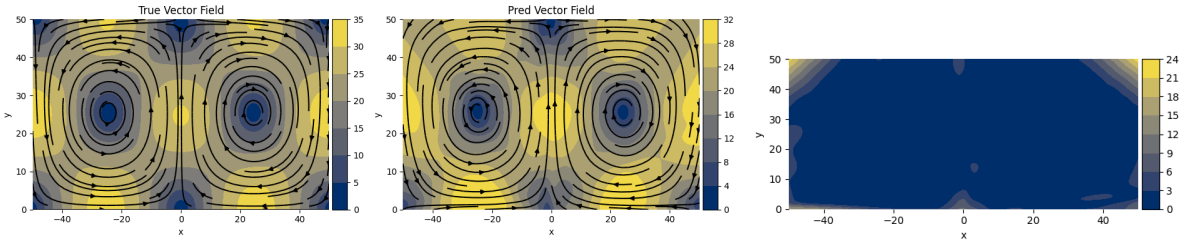
(A) True vector field vs predicted vector field

(B) Error in magnitude of vector field

FIGURE 8. KNODE: Learned Vector Field: A) True vector field vs predicted vector field . B) Error in magnitude of vector field at each point within domain of training data

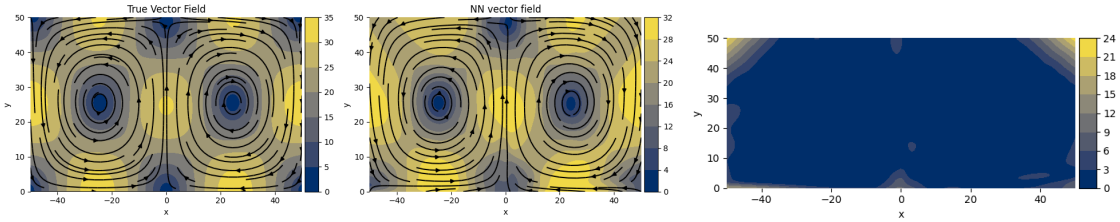
APPENDIX B. LEARNED VECTOR FIELDS OF DOUBLE GYRES

Figs (8-10) shows the learned vector fields for NODEs, ANODEs, and KNODEs respectively.



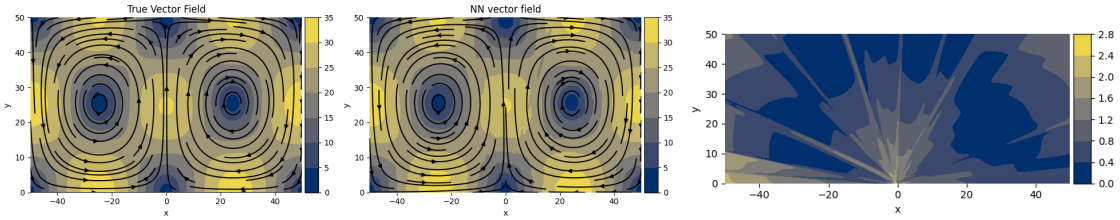
(A) True vector field vs predicted vector field (B) Error in magnitude of vector field

FIGURE 9. NODE: Learned Vector Field: A) True vector field vs predicted vector field . B) Error in magnitude of vector field at each point within domain of training data



(A) True vector field vs predicted vector field (B) Error in magnitude of vector field

FIGURE 10. ANODE: Learned Vector Field: A) True vector field vs predicted vector field . B) Error in magnitude of vector field at each point within domain of training data

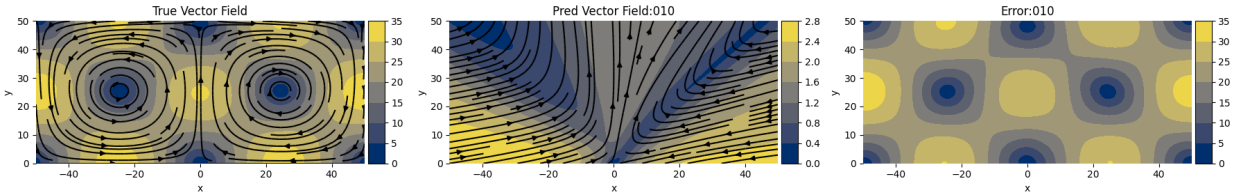


(A) True vector field vs predicted vector field (B) Error in magnitude of vector field

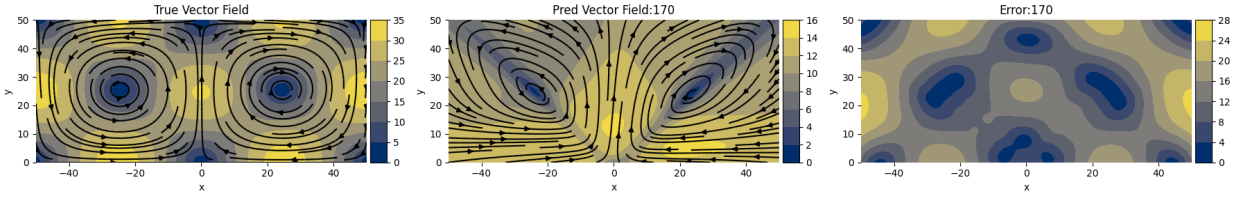
FIGURE 11. KNODE: Learned Vector Field: A) True vector field vs predicted vector field . B) Error in magnitude of vector field at each point within domain of training data

APPENDIX C. EVOLUTION OF TRAINING

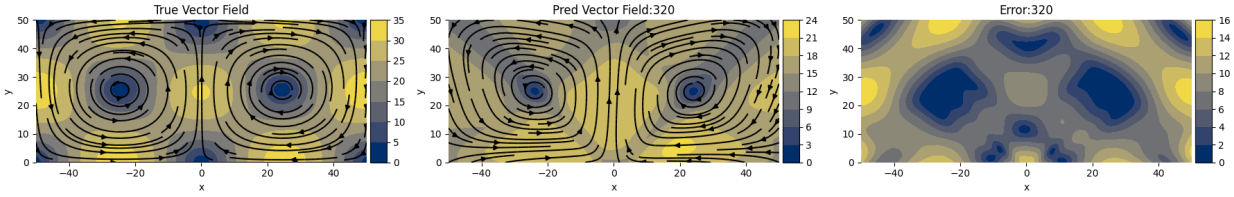
In this section, we show the how our neural network learns the underlying vector at different epochs in a NODE Double-Gyre system.



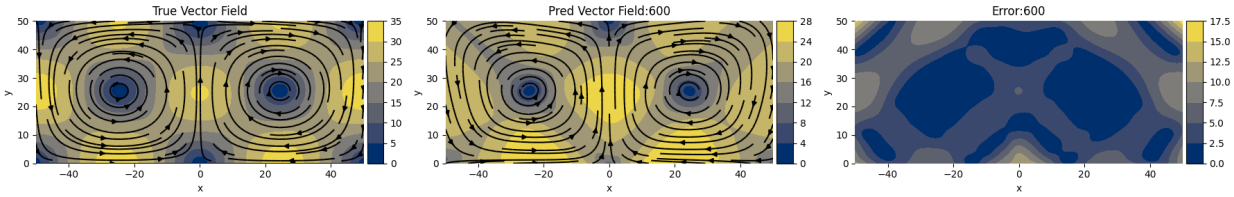
(A) True vector field vs predicted vector field and error in magnitude of vector field: Epoch 10



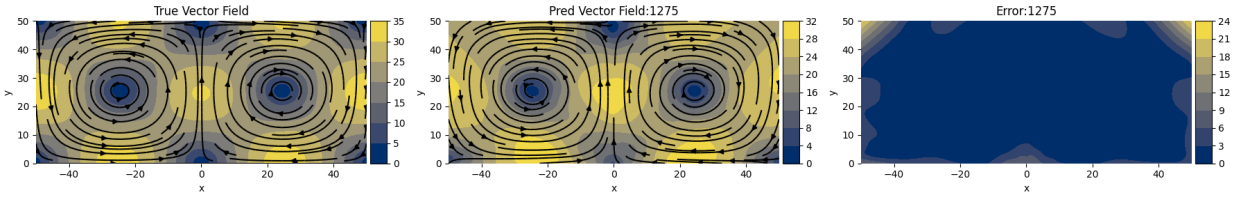
(B) True vector field vs predicted vector field and error in magnitude of vector field: Epoch 170



(C) True vector field vs predicted vector field and error in magnitude of vector field: Epoch 320



(D) True vector field vs predicted vector field and error in magnitude of vector field: Epoch 600

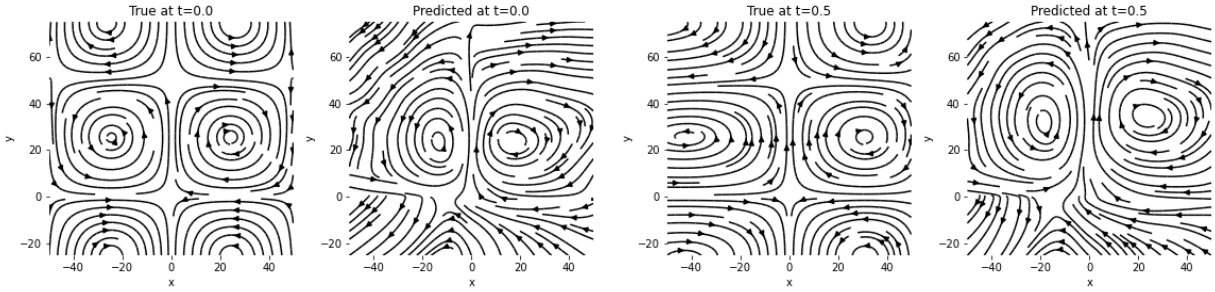


(E) True vector field vs predicted vector field and error in magnitude of vector field: Epoch 1275

FIGURE 12. NODE: Training Evolution for Double Gyre system

APPENDIX D. RESULTS OF NODE IN TIME-VARYING DOUBLE GYROS

In this section, we show the results of NODEs in time-varying double gyros. The trajectories of ten particles in each gyro were fed into training. Unlike learning time-invariant gyros, time was also incorporated into the training process. We used the same parameters as Tab. 1a except $\epsilon = 0.5$, which causes a periodic, horizontal expansion and contraction. Fig. 13 shows the true and learned time-varying vector field trained for 5000 iterations at $t = 0$ and 0.1. The predicted vector field diverged in y direction as time went by. For future work, we can try training with more data both in terms of the number of trajectories and the period of time.



(A) t=0 (B) t=0.5

FIGURE 13. True and predicted vector field of time-varying double gyros