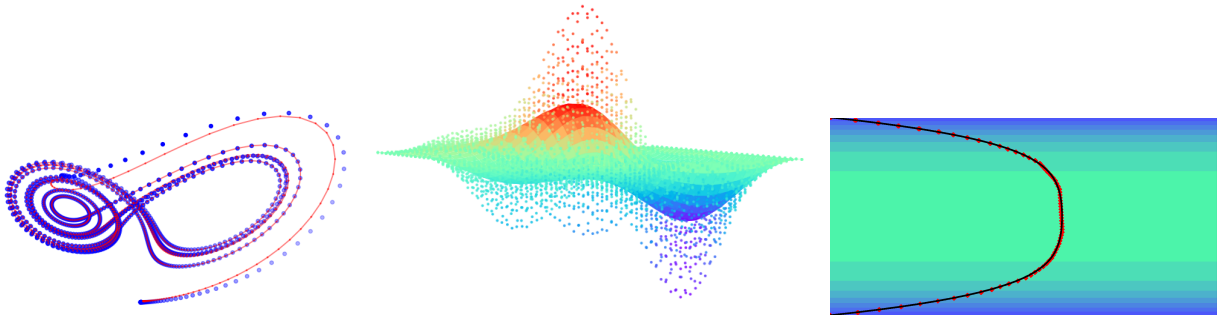
Deep Operator Network Applied to Data-Assimilated Solutions to the Lorenz System and the Two-Dimensional Heat Equation

Gary Guzzo

Dr. Tian - Math 881



Abstract

We use DeepONet [1] to approximate solutions to the data-assimilated Lorenz system, the data-assimilated two-dimensional heat equation, and simple pipe flow. Our main focus is using a DeepONet to solve the nudging algorithm (i.e. data assimilation); an iterative procedure to approximate the "true" solution to a differential equation with unknown initial condition by use of a "nudging" term and access to *some* of the "true" dynamics. We mainly treat the Lorenz system, famous for its chaotic behavior. We analyze the convergence of the nudging algorithm to the true solution.

1 Introduction

The use of machine learning to solve differential equations is at the core of this study. Often referred to in the use of neural networks (such as a DeepONet) is the universal approximation theorem [1], stating that the solution to any continuous function can be approximated to arbitrary accuracy by a neural network whose capacity is not constrained. Our studies rest on a similar approximation result, stating that "a neural network with a single hidden layer can approximate accurately any nonlinear *operator* (a mapping from a space of functions into another space of functions)" [1].

Arising from the Navier-Stokes equations is the *Lorenz system*; a system $\frac{d\mathbf{x}}{dt} = f(\mathbf{x}),\ \mathbf{x}(0) = \mathbf{x}_0$, of three coupled ordinary differential equations in the dependent variable $\mathbf{x} \in \mathbb{R}^3$ and independent variable $t \in \mathbb{R}$. We consider a modified ("*nudged*") version of the Lorenz system with dependent variables $\mathbf{u} \in \mathbb{R}^3$ and $\mathbf{x}'\vec{i} = x \in \mathbb{R}$ (where x is from the true Lorenz system), and independent variable $t \geq 0$. We make corrections to the nudged states based on the difference (at a collection of times t_i) between the x-components of each system's solution. That is, we subtract the term $\mu(u - x)$ from the first component of the nudged system in efforts to force the nudged trajectory into the orbit of the true Lorenz system. We train a DeepONet to learn this algorithm. Once the ONet is trained, it can approximate the solutions to the true and nudged

systems (with new random initial conditions) in less computation time than the methods which we use to compute the reference solutions.

1.1 Overview

For the Lorenz system, we solve by *Euler method*; simple numerical integration over time (10). For the nudged Lorenz system, we solve the *nudging algorithm* (12). For the two-dimensional heat equation, we solve using *finite-difference method* (30). For the nudged heat equation, we solve the *finite difference nudging algorithm* (36). For the Navier-Stokes equations (simple pipe flow), we solve using *finite-difference method*. In each case, we consider a new method of solving the system; training and using a *DeepONet* [1].

1.2 Layout of the Paper

In Section 2, we introduce differential equations and dynamical systems, and view a specific case; *the Lorenz system*. In Section 3 we consider the numerical setting for the Lorenz system. In Section 4, we introduce data-assimilation, define *the nudging algorithm*, and analyze its convergence to the true solution. In Section 5 we introduce DeepONet and train a DeepONet to learn the nudging algorithm (hence teaching it to approximate solutions to differential equations with missing information). In Section 6 we use DeepONet to solve the data-assimilated two-dimensional heat equation and analyze the convergence of the data-assimilated solution to the true solution. In Section 7, we use DeepONet to solve a simple case of the incompressible Navier-Stokes equations; two-dimensional pipe flow.

2 Background Information

Let $\mathbf{x}(t) = \big(x(t), y(t), z(t)\big) \in \mathbb{R}^3$ be the position of a particle in space at some time $t \geq 0$. Here, $t \in \mathbb{R}^+$ is the independent variable, and $\mathbf{x} = \mathbf{x}(t) \in \mathbb{R}^3$ is the dependent variable. We consider the ordinary differential equation (ODE):

$$\frac{d\mathbf{x}}{dt} = \mathbf{f}(\mathbf{x}), \ \mathbf{x} \in \mathbb{R}^3. \tag{1}$$

For clarity, we can write (1) as a system of ODEs:

$$\frac{dx}{dt} = f_1(x, y, z)$$
$$\frac{dy}{dt} = f_2(x, y, z)$$
$$\frac{dz}{dt} = f_3(x, y, z)$$

Given the initial position $\mathbf{x}(t_0) = \mathbf{x}_0 \in \mathbb{R}^3$ of a particle, to determine the location $\mathbf{x}(t_0 + \delta) \in \mathbb{R}^3$ of the particle at time $t = t_0 + \delta$, $\delta > 0$, we solve (or approximately solve) 1 together with the initial condition (IC) $\mathbf{x}(t_0) = \mathbf{x}_0$. By the fundamental theorem of calculus, the solution of 1 with the given IC is a continuous function of time $\mathbf{x} : [0, T] \to \mathbb{R}^3$ given by

$$\mathbf{x}(t; \mathbf{x}_0) = \mathbf{x}_0 + \int_0^t \mathbf{f}(\mathbf{x}(\tau))\mathbf{d}\tau \tag{2}$$

satisfying $\dfrac{d}{dt}\mathbf{x}(t) = \mathbf{f}(\mathbf{x}(t))$. We call $\mathbf{x}(t; \mathbf{x}_0)$ the **solution** to (1) with IC $\mathbf{x}(0) = \mathbf{x}_0$ (or equivalently, the solution to the integral equation (2) [2]). Then we can define for any $t \in \mathbb{R}$ a **solution operator** $S(t) : \mathbb{R}^3 \to \mathbb{R}^3$ by

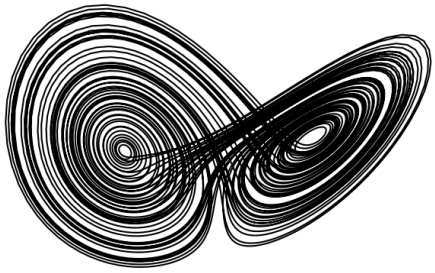
$$S(t)\mathbf{x}_0 = \mathbf{x}(t; \mathbf{x}_0) \tag{3}$$

Here, $S(t)$ takes the initial position $\mathbf{x}_0$ of a particle and returns the position of that particle at time t. Since $\mathbf{x}(t; \mathbf{x}_0)$ is continuous in t and $\mathbf{x}_0$, we also have that $S(t)\mathbf{x}_0$ is continuous in t and $\mathbf{x}_0$. The pair

$$(\mathbb{R}^3, \{S(t)\}_{t \in \mathbb{R}})$$

of the phase space $\mathbb{R}^3$ and the family of solution operators $\{S(t)\}_{t\in\mathbb{R}}$ is referred to as a **dynamical system** generated by the ODE. We will often write $(\mathbb{R}^3, S(t))$ and refer to $S(t)$ as the *trajectory* of the particle.

2.1 The Lorenz System



We now consider the Lorenz system

$$\frac{dx}{dt} = \sigma(y - x) \tag{4}$$

$$\frac{dy}{dt} = x(\rho - z) - y \tag{5}$$

$$\frac{dz}{dt} = xy - \beta z \tag{6}$$

where we take the parameters to be $\sigma = 10$, $\rho = 28$, and $\beta = 8/3$. We can write this system as the following initial value problem (IVP)

$$\frac{d}{dt}\mathbf{x} = L(\mathbf{x}) \quad ; \quad \mathbf{x}(t_0) = \mathbf{x}_0 \tag{7}$$

where $L : \mathbb{R}^3 \to \mathbb{R}^3$ is a differential operator mapping a particle's position to its velocity for any t. We seek the solution $\mathbf{x}(t; \mathbf{x}_0)$ of (7) which gives the position of the particle for any time $t \geq 0$. Solving (7) is then equivalent to solving the integral equation $\mathbf{x}(t) = \mathbf{x}_0 + \int_{t_0}^{t} L(\mathbf{x}(s))\mathbf{d}s$, allowing us to write for any $t \geq 0$ a solution operator

$$S(t)\mathbf{x}_0 := \mathbf{x}(t, \mathbf{x_0}) = \mathbf{x}_0 + \int_{t_0}^{t} \frac{d}{dt}(\mathbf{x}(s))\mathbf{d}s. \tag{8}$$

The trajectory of the particle is then the set of all solution operators $\{S(t)\}_{t\geq 0}$ (although we could take $t \in \mathbb{R}$, we choose to only consider non-negative times). Given any time $t \geq 0$, the image of $\mathbf{x}_0 \in \mathbb{R}^3$ under the function $S(t)$ is a point $\mathbf{x} \in \mathbb{R}^3$ representing the particle's position at time t. Hence, the image of $\mathbf{x}_0$ under the collection $\{S(t)\}_{t\geq 0}$ traces out the trajectory (or path) of the particle through space $\mathbb{R}^3$ over time $t \geq 0$. We will visualize the images of a few discrete solution operators in the next section.

3 Numerical Setting for the Lorenz System

Instead of trying to solve the integral in (8) exactly, we approximate via numerical integration. That is, we plug the IC $\mathbf{x}_0 = (x_0, y_0, z_0) \in \mathbb{R}^3$ into the system (4)-(6) and obtain the velocity $\frac{d}{dt}\mathbf{x}(t_0) \in \mathbb{R}^3$ of the particle at time $t = t_0$. We want to move the particle slightly in the direction of its velocity to approximate its next location. In this way, we obtain the recurrence

$$\mathbf{x}(t_{n+1}) = \mathbf{x}(t_n) + \eta \frac{d}{dt}\mathbf{x}(t_n) \ , \quad n \geq 0 \tag{9}$$

for some small damping constant $\eta > 0$. This recurrence (9) can be viewed by writing the integral equation (8) as a sequence of partial sums.

$$\mathbf{x}_{n+1} = \mathbf{x}_0 + \eta \sum_{i=0}^{n} \mathbf{g}_i \tag{10}$$

where $\eta = \delta_t$ is a time-stepping parameter and $\mathbf{g}_i = L(\mathbf{x}_i) = \frac{d}{dt}\mathbf{x}_i$ for $i \in \{0, 1, 2, \ldots, N-1\}$.

This is the *Euler method*; a simple *Runge-Kutta* method for numerical integration of an ordinary differential equation. We discuss the convergence of the Euler method in Section 3.1 (in short, it converges to a point for "steady" parameters, and does not converge to a point for "chaotic" parameters). In any case, Lorenz solutions reach a *global attractor*, introduced in Section 3.2.

We view the Euler method in a Python function below, where $N = 10$ time-steps are taken:

```
def numerical_integration(X):
    N = 10
    L = np.empty((N,3))
    L[0] = X

    def step(x,y,z,eta=.001,s=10,r=28,b=8/3):
        X = np.array((x,y,z))
        dx = s*(y-x)
        dy = r*x-y-x*z
        dz = x*y-b*z
        dX = np.array((dx,dy,dz))
        return X + eta*dX

    for i in range(1,N):
        L[i] = step(L[i-1])
    return L
```

The output of the code is an array of shape $(10, 3)$ giving the position $\mathbf{x}(t_k)$ of the particle at times $t_k \in \{t_0, \ldots, t_9\}$. We can view a partial trajectory below in a three-dimensional plot, as well as a large trajectory of $N = 5000$ time-steps.

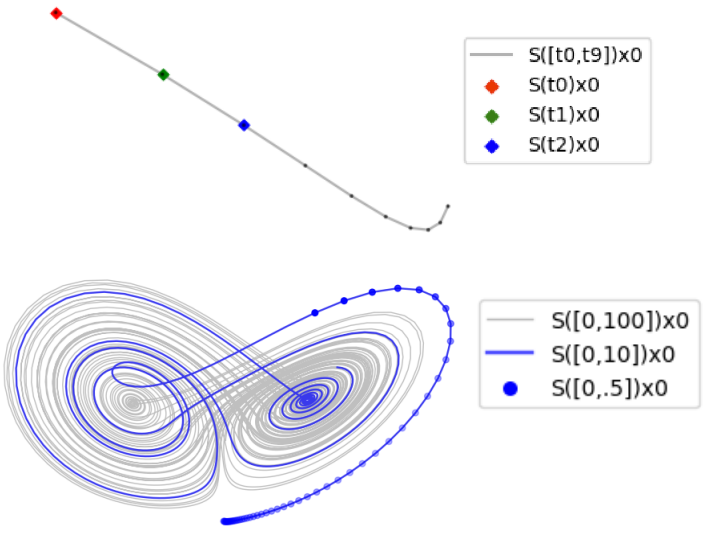


Figure 1: The top figure displays in red the point $\mathbf{x}(t_0) = \mathbf{x}_0 + \int_{t_0}^{t_0} L(\mathbf{x}(s))\mathbf{d}s = \mathbf{x}_0 \in \mathbb{R}^3$, in green the point $\mathbf{x}(t_1) = \mathbf{x}_0 + \int_{t_0}^{t_1} L(\mathbf{x}(s))\mathbf{d}s \in \mathbb{R}^3$, and in blue the point $\mathbf{x}(t_2) = \mathbf{x}_0 + \int_{t_0}^{t_2} L(\mathbf{x}(s))\mathbf{d}s \in \mathbb{R}^3$. In gray is the collection of images of $\mathbf{x}_0$ under the respective operators $\{S(t)\}$ for $t \in [t_0, t_9]$, (i.e., a curve in $\mathbb{R}^3$, or a trajectory). In the bottom figure, we repeat this procedure on a much larger time domain: $\{0 + .01k\}_{k=0}^{10000}$.

3.1 Chaos in the Lorenz System

Say $\mathbf{x}_0$ and $\mathbf{y}_0$ are two distinct initial positions which are "close to" each other (for e.g. say $\mathbf{y}_0 \sim \mathcal{N}(\mathbf{x}_0, 1)$). Then it may be the case that, although $\mathbf{x}_0$ is close to $\mathbf{y}_0$, their corresponding solutions $\mathbf{x}(t; \mathbf{x}_0)$ and $\mathbf{y}(t; \mathbf{y}_0)$ are drastically different. We view an example of such a case below:

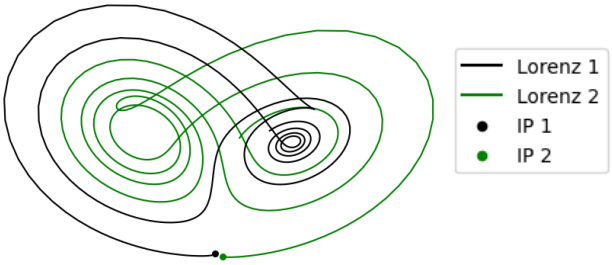


Figure 2: Two solutions $\mathbf{x}(t; \mathbf{x}_0)$ and $\mathbf{y}(t; \mathbf{y_0})$ of the Lorenz system having different (but close to each other) ICs. We used $\sigma = 10$, $\beta = 8/3$, and $\rho = 28$.

3.1.1 Lorenz Parameters: *Chaotic* versus *Steady*

With our choice of parameters $\sigma = 10$, $\beta = 8/3$, and $\rho = 28$, we see chaos remain indefinitely (i.e. the solution $\mathbf{x}$ does not converge to a point). However, by [5], if we make $\rho = 22.2 < \frac{\sigma(\sigma+\beta+3)}{\sigma-\beta-1} \approx 24.74$, then the system becomes *stable* and approaches the *point* (steady state solution):

$$\mathbf{x}_s = \Big(\pm (848/15)^{1/2}, \pm (848/15)^{1/2}, 106/5 \Big)' \approx \big(\pm 7.52, \pm 7.52, 21.2 \big)', \tag{11}$$

We see this below, where we have two Lorenz solutions; a *chaotic* solution (left), and a *stable* solution (right).

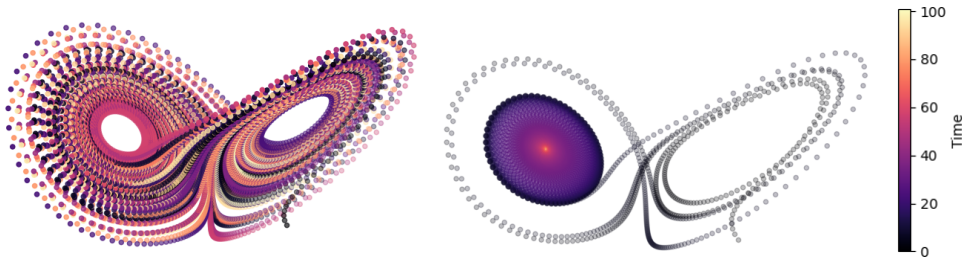


Figure 3: We use a time-step size of $\delta_t = .01$ and numerically solve for $N = 10,000$ time-steps to time $t = 100$. We set the initial condition $\mathbf{x}_0 = (10, 0, 10)'$. On the left, we use the "chaos parameters" $\sigma = 10$, $\beta = 8/3$, and $\rho = 28$, and see that chaos remains throughout the entire time interval. On the right, we uses the "stable parameters" $\sigma = 10$, $\beta = 8/3$, and $\rho = 22.2$, and see the solution approaching the steady state solution $\mathbf{x}_s \approx (-7.52, -7.52, 21.2)$ from (11).

We mainly treat the chaotic case in this study. Unless otherwise stated, *any mention of the Lorenz system will be in reference to the chaotic case* (i.e. when $\rho = 28$).

3.2 Absorbing Set of the Lorenz System

An **absorbing set** of a dynamical system $\frac{dx}{dt} = f(\mathbf{x})$ is a subset $\mathcal{B}$ of the phase space (for Lorenz, $\mathcal{B} \subset \mathbb{R}^3$) such that $t > t(\mathcal{B}) \Rightarrow x(t) \in \mathcal{B}$ for some sufficiently large time $t(\mathcal{B})$. It is shown in [2] that for any IC $\mathbf{x}_0 \in \mathbb{R}^3$, the trajectory $\mathbf{x}(t; \mathbf{x}_0)$ will eventually (for large enough t) be bounded by a sphere in $\mathbb{R}^3$

of radius $R = \frac{2\beta(\rho+\sigma)^2}{\max\{2,2\sigma,2\beta\}} \approx 28.5$. We show below a trajectory which has been absorbed by the sphere $\{\mathbf{x} \in \mathbb{R}^3 : x^2 + y^2 + (z-\rho-\sigma)^2 < 30^2\}$.

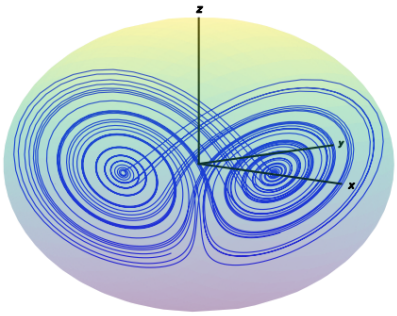


Figure 4: A solution to the (chaotic) Lorenz system bounded by a closed ball $B[30,(0,0,18)'] \subset \mathbb{R}^3$ with radius 30 centered at $\mathbf{x} = (0,0,\rho-\sigma)' = (0,0,18)'$ (left). The smallest absorbing set $\mathcal{A} := \bigcap_{t\geq 0} S(t)\mathcal{B}$ is called the **global attractor** of the dynamical system. We can imagine obtaining the global attractor of the chaotic Lorenz system by "shrink-wrapping" the ball onto the curve. In the steady case (i.e. when $\rho = 22.2$), the global attractor is the union of balls $B[\varepsilon, \mathbf{x}_s^+] \cup B[\varepsilon, \mathbf{x}_s^-]$, where $\varepsilon \to 0^+$ and $\mathbf{x}_s^+$ and $\mathbf{x}_s^-$ are the points from (11).

4 Data Assimilation (DA)

Data assimilation is a "nudging" algorithm used to recover true, continuous dynamics of a dynamical system from a sparse sample of noisy observations. In practice, we encounter differential equations whose full solutions are not available to us. Instead, we only have access to coarse measurements of the state of the system (say at a finite number of time-steps for a temporal ODE, or at a finite number of locations for a PDE). Further, the coarse measurements may be noisy, hence are only approximations to those values from the true solution. Once further, the time in which we begin recording the state of the system may be well after the true system began to evolve. A common example of such a scenario is in the prediction of weather throughout a geographical region. In this case, we would consider the compressible Navier-Stokes equations (with temperature term) over a spatial-temporal domain (say the state of Maryland over the course of a week). We may, for example, only be able to take weather measurements every ten minutes, and only every ten square miles. Nonetheless, we would like a procedure which allows us to take these coarse, noisy measurements and determine the state of the system anywhere in space and at any time.

Data assimilation introduces a new differential equation, identical to the original, except with an additional *nudging* or *correction* term and an arbitrary (or zero) initial position. The nudging term, consisting of a finite number of measurements of the difference between the DA solution and the original solution, acts to pull the DA solution toward the original solution.

4.1 Data Assimilation on the Lorenz System

We consider solutions to the IVP (7) for the Lorenz system

$$\frac{d}{dt}\mathbf{x} = L[\mathbf{x}] \quad ; \quad \mathbf{x}(t_0) = \mathbf{x}_0,$$

as the **truth** or **reference** solutions, for which we may only have sparse, noisy measurements. Furthermore, the initial position $\mathbf{x}_0$ is unknown. Now we introduce a new system of ODEs of the same form as the Lorenz system, with an additional *correction* or *nudging* term

$$\frac{d}{dt}\mathbf{u}(t) = L[\mathbf{u}](t) - \mu\big(I_M[\mathbf{u}_k](t) - I_M[\mathbf{x}_k](t)\big), \quad \forall t \in [t_k, t_{k+1}], \tag{12}$$

$$\mathbf{u}(t_0; \mathbf{u_0}) = \mathbf{u}_0.$$

Here, $\mathbf{u} = \mathbf{u}(t) = \big(u(t), v(t), w(t)\big) \in \mathbb{R}^3$ and $t \in [t_k, t_{k+1}]$ with observed measurements $\mathbf{x}_k$ of the reference solution (10). Here, I_M is an interpolation operator, where M is the number of measurements of the true solution (we will take $M = N$ so that we do not need interpolation). In this study, we consider a case in which we only have access to the x-component of the true solution, taken at observation times $t \in \{t_0, \ldots, t_N\} =: \mathcal{T}_N$. Then for any time $t \in \mathbb{R}$, and for the observation times $t_n \in \mathcal{T}_N$, we can write the system (10) as

$$\frac{du}{dt} = \sigma(v - u) - \mu(u_n - x_n) \tag{13}$$

$$\frac{dv}{dt} = u(\rho - w) - v \tag{14}$$

$$\frac{dw}{dt} = uv - \beta w \tag{15}$$

Notice that if the additional term $\mu(u_n - x_n)$ in (13) is omitted, the system is exactly the Lorenz system. The effect of the additional term is to *nudge* a trajectory toward the true solution, shown below. We point out that, in our case, the term $\mu(u - x) = \mu(u - x)\mathbf{i} + 0\mathbf{j} + 0\mathbf{k}$ is a vector in $\mathbb{R}^3$ whose second and third ($\mathbf{j}$ and $\mathbf{k}$) components are identically zero. We take the nudging constant to be $\mu = 30$.

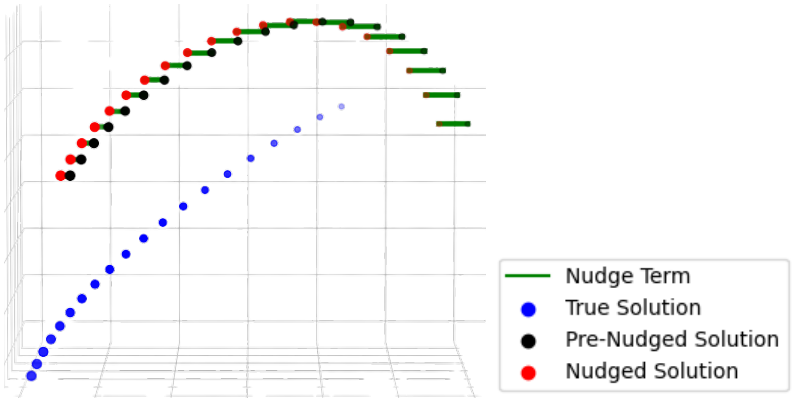


Figure 5: We view partial trajectories of the True Lorenz System (blue), the Lorenz System *Pre-Nudge* (black), and the Nudged Lorenz System (red). In green, we see the effect of the nudging term $\mu(u_n - x_n)$ pulling the $\mathbf{i}$ component of the (pre)-nudged solution toward the $\mathbf{i}$ component of the true solution.

Since the IC $\mathbf{x}(t_0) = \mathbf{x}_0$ is *unknown* to us, we take it to be $\mathbf{x}(t_0) = (0, 0, 0)' \in \mathbb{R}^3$ or "near" $(0, 0, 0)'$. We also take the IC $\mathbf{u}(t_0) = \mathbf{u_0} \in \mathbb{R}^3$ to be zero or near zero. We modify the Python function from Section 3 to compute the nudging DA solution as follows:

```
def DA_numerical_integration(U,L_): #L_ is an array(shape Nx3) = observations from true
                                    Lorenz
    N = 10
    W_ = np.empty((N,3))

    def DA_step(u,v,w,eta=.001,s=10,r=28,b=8/3):
        U = np.array((u,v,w))
        du = s*(v-u)-mu*(u-L_[0])
        dv = r*x-v-u*w
        dw = u*v-b*w
        dU = np.array((du,dv,dw))
        return U + eta*dU

    for i in range(1,N):
        W_[i] = step(W[i-1]) #W_ is an array(shape Nx3) = discrete solution to DA Lorenz
    return W
```

4.2 Convergence of the DA Solution to the True Solution

We consider a true solution $\mathbf{x}(t;\mathbf{x}_0)$ to the Lorenz system (7) and a DA solution $\mathbf{u}(t;\mathbf{u}_0)$ to the DA Lorenz system (10).

Assume $\mathbf{x} = \mathbf{x}(t;\mathbf{x}_0)$ solves (7) and assume $u = \mathbf{u}(t;\mathbf{u}_0)$ solves (12). That is, assume that for every time $t \geq 0$ there exist solution operators $S(t) : \mathbb{R}^3 \to \mathbb{R}^3$ and $Z(t) : \mathbb{R}^3 \to \mathbb{R}^3$ given by

$$S(t)\mathbf{x}_0 = \mathbf{x}(t;\mathbf{x}_0) = \mathbf{x}_0 + \int_0^t L(\mathbf{x})(s)\mathbf{d}s \tag{16}$$

$$Z(t)\mathbf{u}_0 = \mathbf{u}(t;\mathbf{u}_0) = S(t)\mathbf{u}_0 - \mu\int_0^t (u(s) - x(s))ds \ , \quad t \in [t_n, t_{n+1}] \tag{17}$$

Theorem 1. *Solutions (17) to the nudging algorithm (12) converge to solutions (16) of the Lorenz system (7). That is, $Z(t)\mathbf{u}_0 \to S(t)\mathbf{x}_0$ as $t \to \infty$ where $\mathbf{x}_0$ and $\mathbf{u}_0$ come from any compact set $K \subset \mathbb{R}^3$.*

To prove this, we first state results from [2]:

Proposition 2. *Given a trajectory $u(t) = S(t)u_0$, and given $\epsilon > 0$ and $T > 0$, there exists a time $t_* = t_*(\epsilon, T) > 0$ and a point $v_0 \in \mathcal{A}$ (where $\mathcal{A}$ is the global attractor for $S(t)$) such that*

$$|u(t_* + t) - S(t)v_0| \leq \epsilon \quad \forall t \in [0, T].$$

Corollary 3. *Given a solution $u(t)$, There exists 1.) A sequence of errors $\{E_n\}_{n=1}^{\infty}$ with*

$$E_n \to 0, \ n \to \infty,$$

2.) An increasing sequence of times $\{t_n\}_{n=1}^{\infty}$ with

$$[t_n, t_{n+1}] \to \mathbb{R}, \ n \to \infty,$$

and 3.) A sequence of points $\{v_n\}_{n=1}^{\infty}$ with $v_n \in \mathcal{A}$, such that

$$|u(t) - S(t - t_n)v_n| \leq E_n, \ \forall t \in [t_n, t_{n+1}].$$

Furthermore, the jumps $|v_{n+1} - S(t_{n+1} - t_n)v_n| \to 0$ as $n \to \infty$.

Proof of Theorem 1:
Let $\{S(t)\}_{t\geq 0}$ be the semi-group of solution operators which solve (4). We wonder about the distance $|\mathbf{u} - \mathbf{x}|$ between the states $\mathbf{u}(t)$ and $\mathbf{x}(t)$ as $t \to \infty$. The semi-group $\{S(t)\}$ is *dissipative* since in [2] we found a compact absorbing set $\mathcal{B} := B(R, (\sigma + \rho)\mathbf{j}) \subset \mathbb{R}^3$, where $R \approx 28.5$ from Section 3.2. Thus there exists a global attractor

$$\mathcal{A}_S := \bigcap_{t\geq 0} S(t)\mathcal{B}$$

for the semi-group $S(t)$. Thus, by Proposition 2, $\forall \epsilon > 0$ and $\forall T > 0$, $\exists$ a time $t_* \in [0, T]$ and a point $\mathbf{z}_0 \in \mathcal{A}_S$ such that

$$||\mathbf{x}(t_* + t) - S(t)\mathbf{z}_0|| \leq \epsilon \ \ \forall t \in [0, T].$$

In other words, there is some point $\mathbf{z}_0$ in the global attractor such that if we wait until time t_*, the trajectory $S(t + t_*)\mathbf{x}_0$ will be near $\mathbf{z}_0$ in the time interval $[0, T]$.

Claim: $\{Z(t)\}_{t\geq 0}$ is dissipative.
Proof: Let $K \subset \mathbb{R}$ be compact and let $\mathbf{u}_0 \in K$. Then $Z(t)\mathbf{u}_0 = S(t)\mathbf{u}_0 - \mu\int_{t_n}^t (u(s) - x(s))ds$. Now we have $S(t)$, which is dissipative, operating on $\mathbf{u}_0$. Then $S(t)\mathbf{u}_0$ is such that after a long enough time τ, its image

$S(\tau)S(t)\mathbf{u}_0 \in \mathcal{A}_S$ is absorbed into the global attractor. Then, since $\mathbf{x}(t) = S(t)\mathbf{x}_0$ also lives in the global attractor, the states $\mathbf{x}(t+\tau)$ and $S(t+\tau)\mathbf{u}_0$ tend to one another, causing the integrand $\mu(S(t+\tau)\mathbf{u}'\mathbf{i} - \mathbf{x}(t+\tau)'\mathbf{i})$ to tend to zero, and hence $Z(t)\mathbf{u}_0$ is absorbed. This completes the proof of the claim.

So the semi-group $\{Z(t)\}$ is dissipative and hence possesses a global attractor $\mathcal{A}_Z$. Now consider the trajectories

$$\mathbf{x}(t; \mathbf{x}_0) = S(t)\mathbf{x}_0$$

and

$$\mathbf{u}(t; \mathbf{u}_0) = S(t)\mathbf{u}_0 - \mu \int_0^t (u(s) - x(s))ds \ , \quad t \in [t_n, t_{n+1}]$$

as in (16) and (17). Assume that the IC $\mathbf{u}_0 = (0, a, b)' \in \mathbb{R}^3$ where $a, b \in \mathbb{R}$ are chosen arbitrary and small. Assume we have a random IC $\mathbf{x}_0$ near $\mathbf{u}_0$ (possibly with large variance). We choose an initial time interval of $t \in [t_0, t_1]$ and compute

$$||\mathbf{u}(t) - \mathbf{x}(t)|| = ||S(t)\mathbf{u}_0 - S(t)\mathbf{x}_0 + \mu \int_0^t (u(s) - x(s))ds||$$

$$\leq ||S(t)\mathbf{u}_0 - S(t)\mathbf{x}_0|| + ||\mu \int_0^t (u(s) - x(s))ds|| \ , \quad t \in [0, t_1]$$

Say we wait until $t = t_*$ when $S(t)\mathbf{u}_0$ and $S(t)\mathbf{x}_0$ reach the global attractor and say we are considering the time interval $t \in [t_n, t_{n+1}]$. We then know by Corollary 3 that

$$||S(t - t_n)\mathbf{u}_n - \mathbf{x}(t)|| \to 0$$

as $n \to \infty$ (i.e. as we keep considering later and later time intervals). Then, as $\mathbf{x}(t)$ and $S(t-t_n)\mathbf{u}_n$ converge, the x-components $\mathbf{x}(t)'\mathbf{i}$ and $S(t - t_n)\mathbf{u}_n'\mathbf{i}$ converge as well. Thus the integral converges to zero and we get that

$$||\mathbf{u}(t) - \mathbf{x}(t)|| \to 0 \quad as \quad t \to \infty,$$

which completes the proof of Theorem 1.

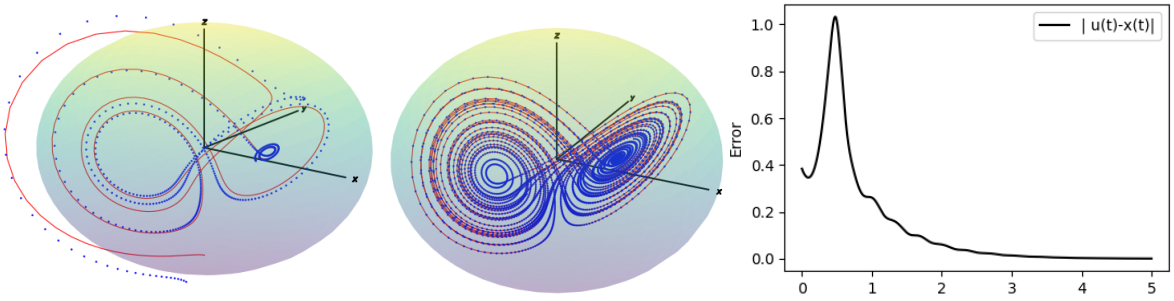


Figure 6: We have a true Lorenz solution (in blue) and a nudged Lorenz solution (in red). On the left, we view the first 1000 timestamps, and observe that the solutions 1) have not yet converged to one another, and 2) are not bounded by the sphere. In the middle, we view timestamps 1000 to 5000, and observe the solutions 1) beginning to converge to one another and 2) residing entirely within the sphere. The sphere (an *absorbing set*) is given by $B = \{(x, y, z) \in \mathbb{R}^3 : x^2 + y^2 + (z - \rho - \sigma)^2 \leq 31^2\}$. For our time-step size we used $\delta_t = \eta = .001$. On the right, we view for each time $t_i \in \{t_1, \ldots, t_{5000}\}$ the distance between the trajectories $\mathbf{u}$ and $\mathbf{x}$ at time t_i using the norm given in (25) from Section 5.4.1.

5 Deep Operator Network (DeepONet, DNN)

A Deep Operator Network [1] is a concatenation of two neural networks, a *branch net* and a *trunk net*. When using DNNs to approximate solutions to differential equations

$$\frac{du}{dx} = g(u(x)) \ ; \ u(0) = u_0 \tag{18}$$

with solution operator given by

$$G_x u(x) = u_0 + \int_0^x g(u(z))dz, \tag{19}$$

the branch net typically inputs a collection of states $u(x_i)$ of the system, and the trunk net typically inputs locations y_i at which to evaluate $g(u)$. We view this using the graphics below [1]:

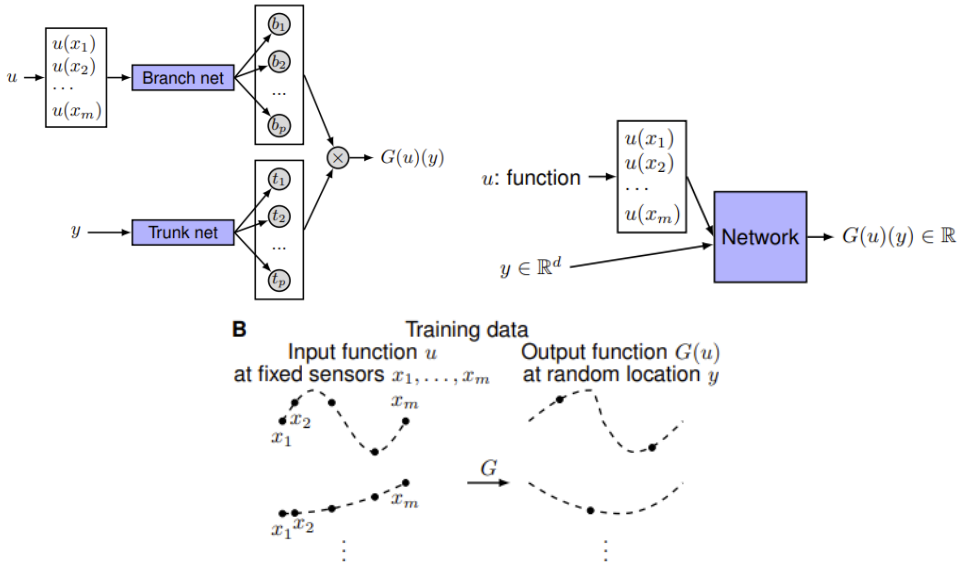


Figure 7: In top figures [1], we view an *unstacked Deep ONet* taking inputs of states u_k and inputs of locations y_j at which to evaluate the next state $G(u_k)(y_j)$. In the bottom figure [1], we view states u_1, u_2, ... at different locations x_i. Then, the solution operator G takes states $u_1, \ldots$ to new states $G(u_1)(y_j), \ldots$, which will act as the *label* of the DeepONet. Hence, a trained DeepONet will take as input the states u and the locations y and output a prediction of the state $G(u)(y)$.

For the Lorenz system, the ODE (7) and its solution operator (8) are used in place of (18) and (19). Thus if we wished to learn the true Lorenz solution (8) (without nudging), we would input the states $\mathbf{x} \in \mathbb{R}^3$ into the branch net, and we would input the times $t_i \geq 0$ into the trunk net, using a label $S(t_i)\mathbf{x}$. However, we wish to *learn the nudging algorithm* (12), so we modify this setup.

5.1 Using a DeepONet to Learn Data Assimilation

We will train a DeepONet to learn the nudging algorithm (12). Following the data structure as in [3], we train our ONet to learn *the next state of the nudging system* (12) for each $t_n \in \{t_1, \ldots, t_N\}$. That is, we take our nth training datum to be

$$X_n = \{\mathbf{u}(t_n), \mathbf{x}(t_n)\}$$
$$Y_n = \mathbf{u}(t_{n+1})$$

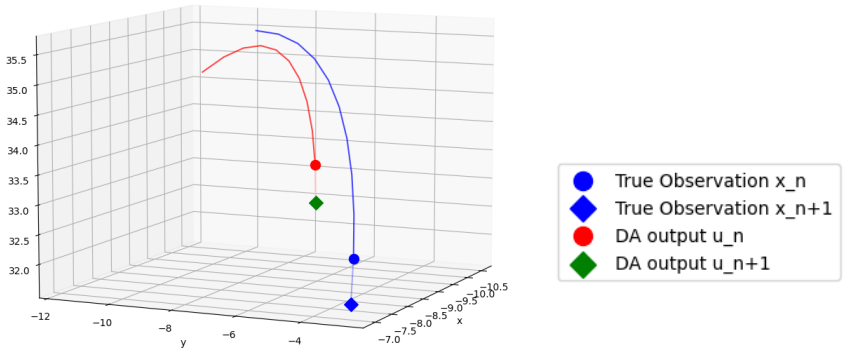


Figure 8: True Lorenz (blue) versus DA Lorenz (red). We use $\{\mathbf{u}_n, \mathbf{x}_n\}$ as out nth training input, and $\mathbf{u}_{n+1}$ (green) as our nth training label.

In this way, we can generate $N \in \mathbb{N}$ training data points so that the input into the DNN for a given trajectory is the collection

$$X = \begin{pmatrix} \{\mathbf{u}(t_0), \mathbf{x}(t_0)\} \\ \{\mathbf{u}(t_1), \mathbf{x}(t_1)\} \\ \vdots \\ \{\mathbf{u}(t_{N-1}), \mathbf{x}(t_{N-1})\} \end{pmatrix}, \quad Y = \begin{pmatrix} \mathbf{u}(t_1) \\ \mathbf{u}(t_2) \\ \vdots \\ \mathbf{u}(t_N) \end{pmatrix} \tag{20}$$

We can view one such input data point and its label below

Figure 9: Viewing the input data X and label Y as trajectories. Nudged Lorenz (red), True Lorenz (blue), Nudged Lorenz *one step further* (green).

5.2 Structuring the Data

To handle the initial condition of the true Lorenz problem being unknown to us, we generate many random ICs in $\mathbb{R}^3$, (say $K = 1000$ of them). Our training data

$$\mathcal{X} := \left\{ \left\{ \mathbf{u}(t_n), \mathbf{x}(t_n) \right\}_{n=0}^{N-1} \right\}_{k=1}^{K}, \quad \mathcal{Y} := \left\{ \left\{ \mathbf{u}(t_n) \right\}_{n=1}^{N} \right\}_{k=1}^{K} \tag{21}$$

is the 2-tuple *input* $\mathcal{X} = \{\mathcal{X}_W, \mathcal{X}_L\}$ and the *label* $\mathcal{Y}$. The set $\mathcal{X}_W$ is the (concatenated) collection of all nudged trajectories $W(t_n)\mathbf{u}_0^{(k)}$ and the set $\mathcal{X}_L$ is the (concatenated) collection of all reference trajectories $S(t_n)\mathbf{x}_0^{(k)}$ for $k \in \{1, \ldots, K\}$ and for $t_n \in \{t_0, \ldots, t_{N-1}\}$. The set of labels $\mathcal{Y}$ is the (concatenated) collection of all nudged trajectories one step further in time than those in $\mathcal{X}_W$. That is,

$$\mathcal{X}_W = \left\{ \mathbf{u}^{(1)}(t_0), \ldots, \mathbf{u}^{(1)}(t_{N-1}), \mathbf{u}^{(2)}(t_0), \ldots, \mathbf{u}^{(2)}(t_{N-1}), \ldots, \mathbf{u}^{(K)}(t_0), \ldots, \mathbf{u}^{(K)}(t_{N-1}) \right\},$$

$$\mathcal{X}_L = \left\{ \mathbf{x}^{(1)}(t_0), \ldots, \mathbf{x}^{(1)}(t_{N-1}), \mathbf{x}^{(2)}(t_0), \ldots, \mathbf{x}^{(2)}(t_{N-1}), \ldots, \mathbf{x}^{(K)}(t_0), \ldots, \mathbf{x}^{(K)}(t_{N-1}) \right\},$$

$$\mathcal{Y} = \left\{ \mathbf{u}^{(1)}(t_1), \ldots, \mathbf{u}^{(1)}(t_N), \mathbf{u}^{(2)}(t_1), \ldots, \mathbf{u}^{(2)}(t_N), \ldots, \mathbf{u}^{(K)}(t_1), \ldots, \mathbf{u}^{(K)}(t_N) \right\}.$$

Thus the entry $\mathcal{X}_{W,n}^{(k)} = \mathbf{u}^{(k)}(t_n)$ gives the state

$$W(t_n)\mathbf{u}_0^{(k)} = \mathbf{u}_0^{(k)} + \eta \sum_{i=0}^{n-1} \dot{\mathbf{g}}_i^{(k)} \tag{22}$$

of the kth trajectory at time $t = t_n$ of the nudged DA system (i.e. a point $\left(u^{(k)}(t_n), v^{(k)}(t_n), w^{(k)}(t_n)\right)' \in \mathbb{R}^3$). This equation is analogous to (10) from Section 2, where we take $\dot{\mathbf{g}}_\mathbf{i} = L(\mathbf{u})(t_i) - \mu\big(\mathbf{u}(t_i) - \mathbf{x}(t_i)\big)$ from the nudging system (12). Similarly, $\mathcal{X}_{L,n+1}^{(k)} = \mathbf{x}^{(k)}(t_{n+1})$ gives the state

$$L(t_{n+1})\mathbf{x}_0^{(k)} = \mathbf{x}_0^{(k)} + \eta \sum_{i=0}^{n} \mathbf{g}_i^{(k)} \tag{23}$$

of the kth trajectory at time $t = t_{n+1}$ in the true Lorenz solution. Since we train our DeepONet to learn the next state of the nudging algorithm (12), the label of $\left(\mathcal{X}_{W,n}^{(k)}, \mathcal{X}_{L,n}^{(k)}\right) = \left(\mathbf{u}^{(k)}(t_n), \mathbf{x}^{(k)}(t_n)\right)$ is the entry $\mathcal{Y}_{n+1}^{(k)} = \mathbf{u}^{(k)}(t_{n+1})$, which gives the state

$$W(t_{n+1})\mathbf{u}_0^{(k)} = \mathbf{u}_0^{(k)} + \eta \sum_{i=0}^{n} \dot{\mathbf{g}}_i^{(k)}. \tag{24}$$

We choose a fixed initial condition $\mathbf{u}_0^{(k)} = (0, .1, .2)'$ for each true trajectory. We use random initial conditions $\mathbf{x}_0^{(k)} \sim \mathcal{N}((0, .1, .2)', 10)$ for each nudged trajectory.

5.2.1 Visualizing the Data

We can view a full training data-set below, as a collection of (concatenated) trajectories of the *nudging DA system* (red), the *true reference system* (blue), and the *future nudging DA system* (green). In essence, we will input states of the true and nudging systems and train the DeepONet to learn the *next* states of the nudging system.

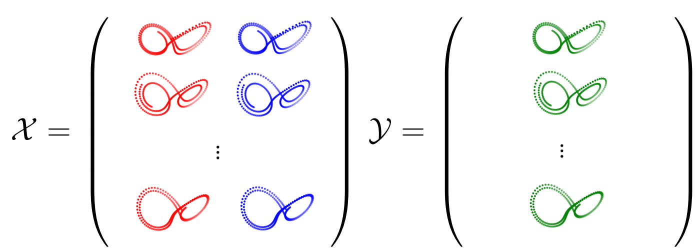


Figure 10: Input $\mathcal{X}$ is a 2-tuple, where each member is a tensor of shape $KN \times 3$ (we "flatten" or reshape from shape $K \times N \times 3$), giving the state of the true and nudging systems on $t_n \in \{t_0 \ldots, t_{N-1}\}$ for all K trajectories. Label $\mathcal{Y}$ is a tensor of shape $KN \times 3$, giving the state of the nudging system on $t_n \in \{t_1, \ldots, t_N\}$ for all K trajectories.

5.3 Structuring the DeepONet

We will train a Deep Operator Network (DeepONet, ONet, DNN) [1] to learn the nudging algorithm (12) given in Section (3.1). Once trained, the ONet is able to predict future states of a dynamical system in less computation time compared to solving the true reference system or the DA nudging system.

A DeepONet consists of two neural networks (in our case, feed-forward neural networks (FNN)) which are concatenated after some number of hidden layers. One FNN (the *branch net*) will take as input the nudging DA Lorenz solution $\mathcal{X}_W$ and the other FNN (the *trunk net*) will take as input the true reference Lorenz solution $\mathcal{X}_L$. The label for the DNN is then the future state of the nudging DA Lorenz solution $\mathcal{Y}$.

5.3.1 Functional Setting

We define a function $f_0 : \mathbb{R}^3 \to \mathbb{R}^j$ for $j \in \mathbb{Z}^+$ by $f_0(\mathbf{u}) = A_0\mathbf{u} + \mathbf{b}_0$ to be the first *hidden layer* of the ONet (we are ignoring the activation function here, but in the study we use *ReLu*). Subsequent hidden layers f_ℓ are given by composition of functions (layers) $f_{\ell-1} \circ \ldots \circ f_1 \circ f_0$. We do this for both the branch net and trunk net, obtaining (from their respective final layers) the two vectors $\vec{v}_B, \vec{v}_T \in \mathbb{R}^j$ for $j \in \mathbb{Z}^+$ and computing their dot product $\vec{v}_B \cdot \vec{v}_T \in \mathbb{R}$. So we can define the *trunk net* as $F_T : \mathbb{R}^3 \to \mathbb{R}^j$ by

$$F^{(T)}(\mathbf{x}) = f^{(T)}_{\ell_T} \circ \ldots \circ f^{(T)}_1 \circ f^{(T)}_0(\mathbf{x}) \in \mathbb{R}^j$$

and the *branch net* $F^{(B)} : \mathbb{R}^3 \to \mathbb{R}^j$ in a similar way. Thus the **Deep Operator Network** (DeepONet, ONet, DNN) is given by the function $\mathcal{DNN} : \mathbb{R}^3 \times \mathbb{R}^3 \to \mathbb{R}$ given by

$$\mathcal{DNN}(\mathbf{u}, \mathbf{x}) = F^{(T)}(\mathbf{u})' F^{(B)}(\mathbf{x}) \in \mathbb{R},$$

and is the vector dot product of the final layers of the branch and trunk nets, each of which is an FNN; a composition of functions which are (up to the activation functions) linear. We use three separate DeepONets, each one learning the respective coordinates u, v, and w of $\mathbf{u} \in \mathbb{R}^3$; the solution to the nudging algorithm (12).

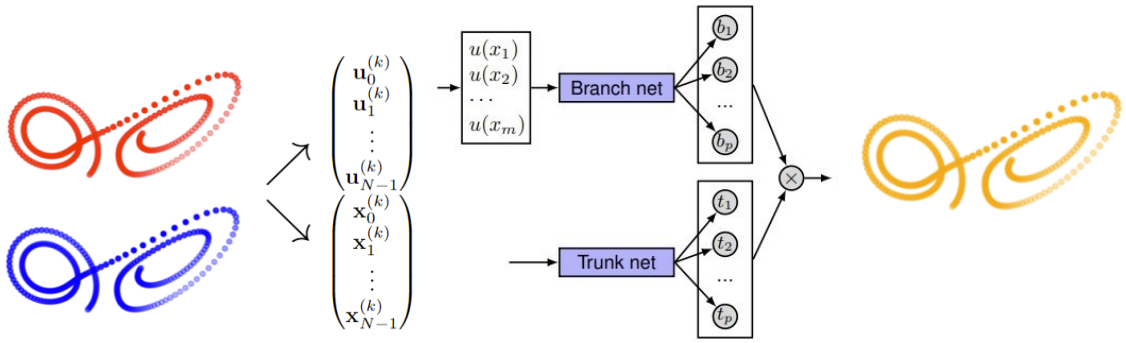


Figure 11: We view a flowchart of an input datum (nudged solution in red, true solution in blue) being fed into a *trained* DeepONet, which outputs the predicted nudged solution (in gold). When we say that the ONet has been trained, we mean that the ONet has *trained on* a data set $(\mathcal{X}_W, \mathcal{X}_L), \mathcal{Y}$, where the label was $\mathcal{Y}^{(k)} =$  as in Section 4.2.1.

5.4 Network Architecture and Results

For the branch net, we use 4 hidden layers, each hidden layer having 100 nodes. For the trunk net, we use the same architecture. We use a learning rate of .01 and a Glorot normal kernel initializer. We train on $K = 2000$ trajectories for 1000 epochs. We take $\eta = .005$ and $N_t = 3000$ so that our time domain is $\mathcal{T} = \{0, .005, .01, .015, \ldots, 14.95, 15\} = \{.005n\}_{n=0}^{3000}$. We restrict our time domain to $\mathcal{T}_R = \{.005n\}_{n=200}^{3000} = \{2, 2.005, \ldots, 15\}$ to synthesize the mystery of the initial position of the true Lorenz system.

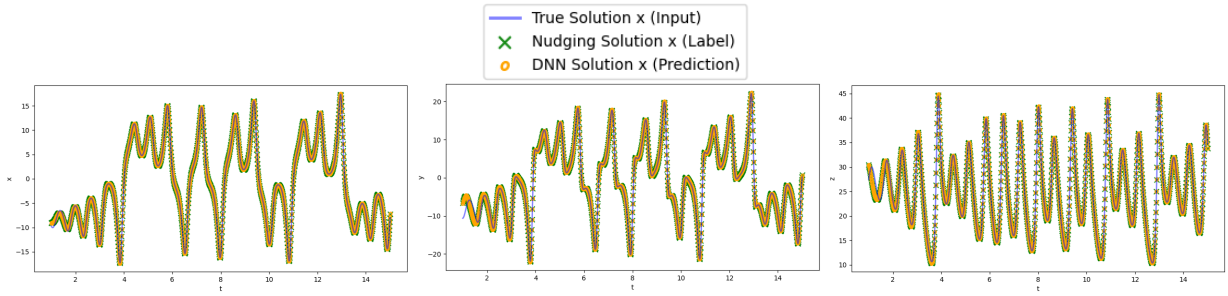


Figure 12: The x, y, and z components, respectively from left to right. We see in blue the true solution, in green the nudged solution, and in gold the DNN prediction.

5.4.1 Obtaining Error Estimates

We consider two types of loss. First, we consider for each trajectory index k and each time index n the loss

$$||\mathbf{u}_n^{(k)} - \mathbf{x}_n^{(k)}|| = \sqrt{(x_n^{(k)} - u_n^{(k)})^2 + (y_n^{(k)} - v_n^{(k)})^2 + (z_n^{(k)} - w_n^{(k)})^2} \tag{25}$$

between the true nudged solutions $\mathbf{u}$ and the true solutions $\mathbf{x}$. We do the same between the DNN prediction $\mathbf{u}^{DNN}$ and the nudged solution $\mathbf{u}$, as well as between the DNN prediction $\mathbf{u}^{DNN}$ and the true solution $\mathbf{x}$.

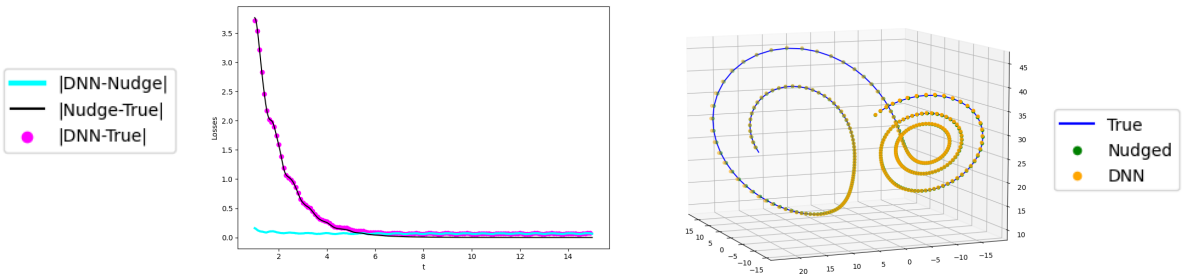


Figure 13: We view the loss among the three trajectories; True, Nudged, and DNN. On the left, we view the last 300 states of these trajectories.

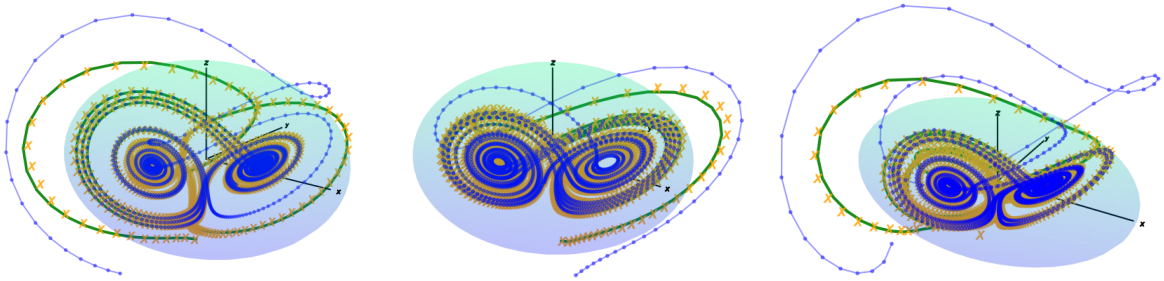


Figure 14: We view a few trajectories showing the True (blue), Nudged (green), and ONet (orange) trajectories converging.

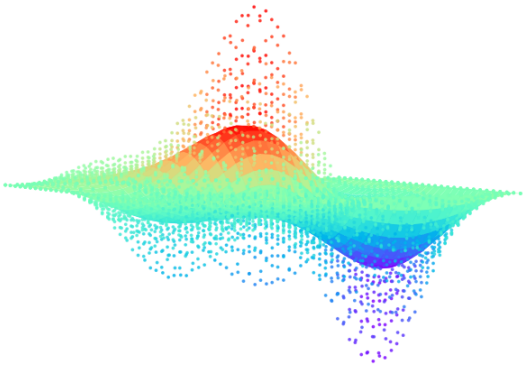
We also compute the RMSE loss among only the *first coordinate* of the three systems. The loss is given

by

$$RMSE = \sqrt{\frac{1}{400(\Gamma-1)} \sum_{n=1}^{\lfloor \Gamma \rfloor} \sum_{k=1}^{400} (u_n^{(k)} - x_n^{(k)})^2}$$

where $k \in \{1, \ldots, 400\}$ are the number of trajectories, $n \in \{1, \ldots, \lfloor \Gamma \rfloor\}$ are the number of time-steps, and $\Gamma \approx 1540$ is a bound arising in [3] on the convergence of the solutions. Across our 400 test trajectories ($\mathcal{X}_{TEST}$, not different form the original test set) and using nudging to compute $\mathbf{u}$, we obtain $RMSE < 0.17$. Using the trained DeepONet to compute $\mathbf{u}$, we obtain $RMSE < 0.5$.

6 Learning the Heat Equation



We now consider a partial differential equation; the *two-dimensional heat equation* (or *diffusion equation*). Let $\bar{\Omega} = [0, a] \times [0, b] \subset \mathbb{R}^2$ be a rectangular spatial domain and let $T = \mathbb{R}^+ \cup \{0\}$ be a time domain. Then we call $\bar{\Omega} \times T$ a *spatial-temporal domain*. An element $(x, y, t) \in \bar{\Omega} \times T$ represents a location $(x, y) \in \bar{\Omega}$ at a time $t \in T$. For any spatial-temporal location $(x, y, t) \in \bar{\Omega} \times T$, we take $u = u(x, y, t) \in \mathbb{R}$ to be the *temperature* of, say a sheet of metal, at the location $(x, y) \in \bar{\Omega}$ at the time $t \in T$. The **heat equation** (with initial and boundary conditions) is then given by

$$\frac{\partial u}{\partial t} = \alpha \left(\frac{\partial^2 u}{\partial x^2} + \frac{\partial^2 u}{\partial y^2} \right) \text{ in } \Omega \tag{26}$$

$$u = 0 \text{ on } \partial\Omega \tag{27}$$

$$u(x, y, 0) = \phi(x, y) \tag{28}$$

$$\frac{\partial u}{\partial t}(x, y, 0) = \psi(x, y), \tag{29}$$

where α is the constant diffusion coefficient. Equation (26) says that the change in temperature over time is proportional to the "concavity" of the temperature distribution. Equation (27) says that the temperature is fixed at zero on the boundary $\partial\Omega = \{(x, y) : x = 0 \vee x = a \vee y = 0 \vee y = b\}$ of the rectangle. Equation (28) says that, at time $t = 0$, the heat of the "plate" (the rectangle $\bar{\Omega}$) is given by the function $\phi(x, y)$. Equation (29) says that, at time $t = 0$, the heat is instantaneously changing according to the function $\psi(x, y)$.

6.1 Discrete Setting

Consider the discrete spatial domain $\mathcal{D} = \mathcal{D}_x \times \mathcal{D}_y = \{0, \delta_x, 2\delta_x, \ldots, N_x\delta_x\} \times \{0, \delta_y, 2\delta_y, \ldots, N_y\delta_y\} \subset \bar{\Omega} \subset \mathbb{R}^2$, where $N_x, N_y \in \mathbb{Z}^+$ and $\delta_x, \delta_y \in \mathbb{R}^+$. In this way, $a := \delta_x N_x$ is the right edge of $\mathcal{D}$ and $b := \delta_y N_y$ is the bottom edge of $\mathcal{D}$. Consider a function $\Phi : \mathcal{D} \to \mathcal{R} \subset \mathbb{R}$ given by $\Phi\big((x_i, y_j)\big) = \Phi_{ij}$.

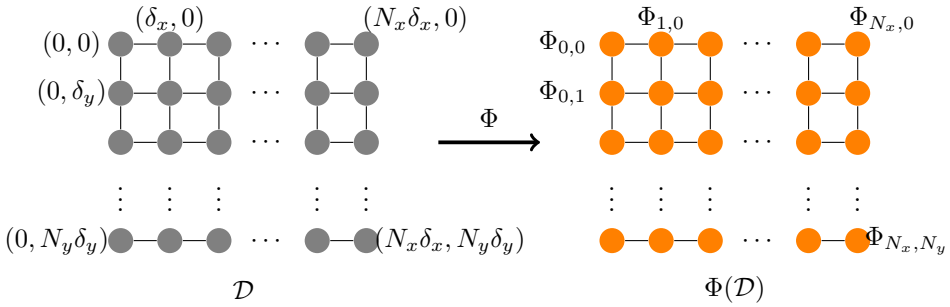


Figure 15: A discrete domain $\mathcal{D} \subset \mathbb{R}^2$ of shape $2 \times N_x \times N_y$ being mapped by Φ into a discrete co-domain $\mathcal{R} \subset \mathbb{R}$ of shape $1 \times N_x \times N_y$. That is, every element of $\mathcal{D}$ is an ordered pair (x_i, y_j) which Φ maps to a real number $\Phi(x_i, y_j) \in \mathcal{R} \subset \mathbb{R}$.

Functions such as Φ which take $\mathcal{D}$ into $\mathbb{R}$ belong to the space $\ell^2(\mathcal{D})$ of functions defined on $\mathcal{D}$ measurable with the ℓ^2 norm. We view an example below of a function $\Phi : D \to R \subset \mathbb{R}$ by $\Phi(x_i, y_j) = \sin(\pi x_i)\sin(\pi y_j)$.

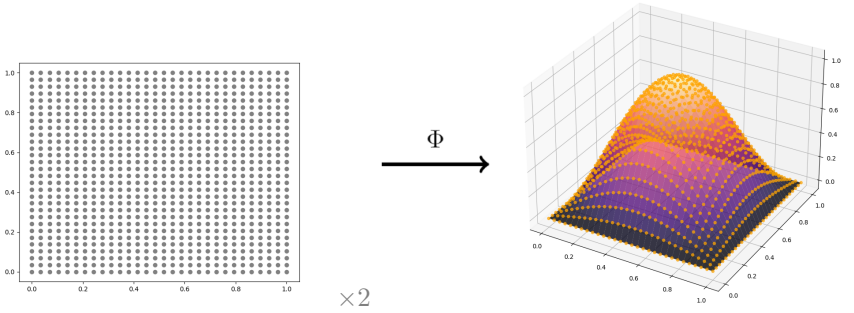


Figure 16: A discrete domain D of size $2 \times 30 \times 30$ being mapped onto its image $\Phi(D)$ under the function $\Phi(x_i, y_j) = \sin(\pi x_i)\sin(\pi y_j)$.

Let $\mathcal{T} = \{t_0, \ldots, t_N\} \subset T \subset \mathbb{R}$ be a time domain. Then we call $\mathcal{D} \times \mathcal{T} \subset \bar{\Omega} \times T \subset \mathbb{R}^2 \times \mathbb{R}$ a spatial-temporal domain. That is, an element $(x_i, y_j, t_n) \in \mathcal{D} \times \mathcal{T}$ represents a location (x_i, y_j) at time t_n. We will use functions whose input come from $\mathcal{D} \times \mathcal{T}$ to approximate solutions to the problem (26)-(29). In fact, $\mathcal{D} \times \mathcal{T}$ is used as input to the *trunk net* when learning the heat equation.

6.2 Obtaining a Discrete Approximation using FDM

We use finite difference method (FDM) to approximate the solution to (26)-(29). As done for the wave equation, we consider the discrete spatial-temporal domain $\mathcal{T} \times \mathcal{D}$ which has shape (N_t, N_x, N_y) and mesh size $(\delta_t, \delta_x, \delta_y)$. We replace the partial differential equation (26) with a *partial difference equation*. For each point $(t_n, x_i, y_j) \in \mathcal{T} \times \mathcal{D}$ we define

$$\frac{u_{i,j}^{n+1} - u_{i,j}^n}{\delta_t} = \alpha\Big(\frac{u_{i+1,j}^n - 2u_{i,j}^n + u_{i-1,j}^n}{\delta_x^2} + \frac{u_{i,j+1}^n - 2u_{i,j}^n + u_{i,j-1}^n}{\delta_y^2}\Big) \tag{30}$$

as the difference equation corresponding to (26), where we replaced derivatives with approximate derivatives. Impose the boundary and initial condition (27) and (28) by setting

$$u_{i,j}^n = 0 \;\forall\; (i,j) \in \{(0,j), (N_x,j), (i,0), (i,N_y)\} \tag{31}$$

$$u^0_{i,j} = \phi_{i,j} \tag{32}$$

$$\frac{u^0_{i,j} - u^{-1}_{i,j}}{\delta_t} = 0 \tag{33}$$

where $\phi_{i,j} = \phi(x_i, y_j)$ is a discrete random surface (such as one generated in Section 6) and the initial velocity is taken to be identically 0 (this IC is automatically enforced in practice). By [4], the system is stable so long as $\delta_t \leq \frac{1}{2}\delta_x\delta_y$.

6.2.1 Solving the Difference Equation

We can solve (30) for the first term, $u^{n+1}_{i,j}$ to give us the next state of the system. We obtain

$$u^{n+1}_{i,j} = \alpha\delta_t\Big(\frac{u^n_{i+1,j} - 2u^n_{i,j} + u^n_{i-1,j}}{\delta_x^2} + \frac{u^n_{i,j+1} - 2u^n_{i,j} + u^n_{i,j-1}}{\delta_y^2}\Big) + u^n_{i,j} \tag{34}$$

Thus if we generate a random surface $u^0_{i,j} = \phi(x_i, y_j)$ (in practice, a NumPy array of shape (N_x, N_y)), then we can compute $u^1_{i,j}$ using (34) with $u^n = u^0$. In the next iteration, we compute $u^2_{i,j}$, again using (34), with $u^n = u^1$.

We continue the iterative procedure for N_t time steps to obtain the full (discrete) trajectory of the heat equation; a NumPy array U of shape (N_t, N_x, N_y). Say $\nu(x, y, y)$ solves (26)-(29). Then choosing the spatial-temporal location $(t_n, x_i, y_j) \in \mathcal{T} \times \mathcal{D}$ should give a small ℓ^2 loss $||\nu(x_i, y_j, t_n) - u^n_{i,j}||$ between the analytical solution $\nu(x_i, y_j, t_n)$ and the FDM approximation $u^n_{i,j}$ at the location $(x_i, y_j, t_n) \in \mathcal{D} \times \mathcal{T}$.

We define the following Python function to compute the finite difference approximation:

```
def FDM(u = random_surface, nt=nt, nx=nx, ny=ny, a=a):
    U = np.zeros((nt,ny,nx))

    for n in range(nt):
        un = u.copy()
        uxx = 1/dx**2 *(un[0:-2, 1:-1]-2*un[1:-1, 1:-1] + un[2:, 1:-1])
        uyy = 1/dy**2 *(un[1:-1, 0:-2]-2*un[1:-1, 1:-1] + un[1:-1, 2:])
        laplac = uxx+uyy

        u[1:-1,1:-1] = a*dt* laplac + un[1:-1,1:-1]

        #Boundary condition
        u[:,0],u[:,-1],u[0,:],u[-1,:]=0,0,0,0

        U[n] =
    return U #U is an array of shape (nt,ny,nx)
```

6.2.2 Solution Operator

We know that an initial condition $u_0 = \phi(x, y)$ is a function $\phi \in \ell^2(\mathcal{D})$ whose graph is a (discretized) surface representing the distribution of heat over $\mathcal{D}$. Then to discuss future states u_n (also functions in $\ell^2(\mathcal{D})$) of the system, we introduce a *solution operator* as follows. Say $u = u(x_i, y_j, t_n)$ is the solution (34) to (26)-(29), where the IC was $u(x, y, 0) = u_0(x, y)$. Then for any $t_n \in \mathcal{T}$ we can define the solution operator $S(t_n) : \ell^2(\mathcal{D}) \to \ell^2(\mathcal{D})$ by

$$S(t_n)u_0 = u(x_i, y_j, t_n) = u^n_{i,j} \tag{35}$$

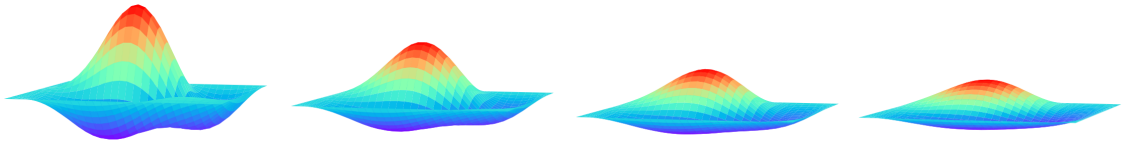


Figure 17: States $S(0)u_0$, $S(0.05)u_0$, $S(0.1)u_0$, and $S(0.15)u_0$ of the system (26)-(29).

6.3 Data Assimilation (Nudging Algorithm)

Like (1) and (7), the heat equation has a *first time derivative*, which can be solved for. Assume $u_{i,j}^n$ is the FDM solution (34) to the difference equation (30) with BC and ICs (31)-(33). We define a new difference equation

$$\frac{w_{i,j}^{n+1} - w_{i,j}^n}{\delta_t} = \alpha\Big(\frac{w_{i+1,j}^n - 2w_{i,j}^n + w_{i-1,j}^n}{\delta_x^2} + \frac{w_{i,j+1}^n - 2w_{i,j}^n + w_{i,j-1}^n}{\delta_y^2}\Big) - \mu\big(w_{i,j}^n - u_{i,j}^n\big) \tag{36}$$

involving a correction term $\mu(w_{i,j}^n - u_{i,j}^n)$, where $u_{i,j}^n$ is the solution (34). We set the boundary and initial conditions

$$w_{i,j}^n = 0 \;\forall\; (i,j) \in \{(0,j),(N_x,j),(i,0),(i,N_y)\} \tag{37}$$

$$w_{i,j}^0 = \varphi_{i,j} \tag{38}$$

$$\frac{w_{i,j}^0 - w_{i,j}^{-1}}{\delta_t} = 0 \tag{39}$$

where the initial condition $w_{i,j}^0 = \varphi_{i,j}$ is different from the initial condition $u_{i,j}^0 = \phi_{i,j} = \phi(x_i, y_j)$ from (32). We solve the nudged difference equation (36) in the same way that we solved the original difference equation (30). That is, we solve for $w_{i,j}^{n+1}$ and obtain

$$w_{i,j}^{n+1} = \delta_t\bigg(\alpha\Big(\frac{w_{i+1,j}^n - 2w_{i,j}^n + w_{i-1,j}^n}{\delta_x^2} + \frac{w_{i,j+1}^n - 2w_{i,j}^n + w_{i,j-1}^n}{\delta_y^2}\Big) - \mu\big(w_{i,j}^n - u_{i,j}^n\big)\bigg) + w_{i,j}^n \tag{40}$$

Define the corresponding solution operator $Z(t_n) : \ell^2(\mathcal{D}) \to \ell^2(\mathcal{D})$ by

$$Z(t_n)w_0 = w^n = S(t_n)w_0 - \delta_t\mu(w^n - u^n) \tag{41}$$

We can solve this nudged difference equation using the following simple Python code:

```
def nudged_FDM(U = FDM(), w = random_surface, mu=mu, nt=nt, nx=nx, ny=ny, a=a):
    W = np.zeros_like(U)

    for n in range(nt):
        wn = w.copy()
        wxx = 1/dx**2 *(wn[0:-2, 1:-1]-2*wn[1:-1, 1:-1] + wn[2:, 1:-1])
        wyy = 1/dy**2 *(wn[1:-1, 0:-2]-2*wn[1:-1, 1:-1] + wn[1:-1, 2:])
        laplac = wxx + wyy

        w[1:-1,1:-1] = dt*(a*laplac- mu*(wn[1:-1,1:-1] - U[n][1:-1,1:-1])) + wn[1:-1,1:-1]

        #Boundary condition
        w[:,0],w[:,-1],w[0,:],w[-1,:]=0,0,0,0

        W[n]=w
    return W #W is an array of shape (nt,ny,nx)
```

We obtain the solution $u(x_i, y_j, t_n) = u_{i,j}^n$ to the true difference equation (30) and the solution $w(x_i, y_j, t_n) = w_{i,j}$ to the nudged difference equation (36) and plot the result for timestamps $t_i \in \{0, .02, .04, .06, .08\}$ below.

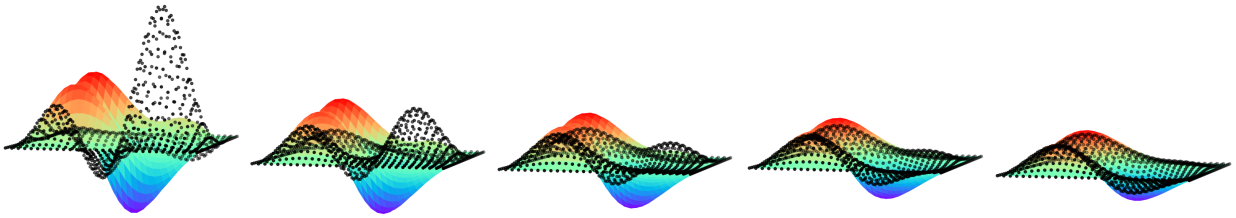


Figure 18: The true numerical solution $u_{i,j}^n$ in rainbow, and the nudged numerical solution $w_{i,j}^n$ in black. Notice it only took until time $t = .08$ for the nudged solution to approach the true solution.

6.4 Analysis of the Nudging Algorithm

The discrete initial value problem (30)-(33) has an approximate solution (34) and is described by the solution operator (35). Thus, we obtain a **semi-dynamical system** consisting of the *semi-group of solution operators* $\{S(t_n)\}_{t_n \geq 0}$ together with the *phase space* $\ell^2(\mathcal{D})$, written

$$\Big(S(t_n), \ell^2(\mathcal{D})\Big). \tag{42}$$

Note that the phase space is a space of functions in which each state $S(t_n)u_0 \in \ell^2(\mathcal{D})$ of the semi-dynamical system resides. Recall that such states (functions in $\ell^2(\mathcal{D})$) can be visualized as a (collection of points in $\mathbb{R}^3$ lying on a) surface in $\mathbb{R}^3$.

In a similar way, we can define a semi-dynamical system for the nudged heat equation (36)-(39) as

$$\Big(Z(t_n), \ell^2(\mathcal{D})\Big). \tag{43}$$

6.4.1 Dissipation of the Heat Equation

In order for the semigroup $\{S(t_n)\}$ to be dissipative, it must possess a compact absorbing set $\mathcal{B} \subset \ell^2(\mathcal{D})$. Intuitively, we can again imagine a rectangular sheet of metal on which we have temperature measurements, where a positive (hot) temperature would have positive "height", and a negative (cold) temperature would have negative "height". We can expect that the temperature at every point on the sheet of metal will eventually reach "room temperature", corresponding to a "height" of zero.

To bound a function $u \in \ell^2(\mathcal{D})$, we need to find a finite collection of finite functions $v_\alpha \in \ell^2(\mathcal{D})$ such that its graph $\big(\mathcal{D}, u(\mathcal{D})\big) \subset \mathbb{R}^2 \times \mathbb{R}$ lies entirely within a (finite union of) finite graphs $\bigcup_{\alpha \in I} \big(\mathcal{D}, v_\alpha(\mathcal{D})\big) \subset \mathbb{R}^3$.

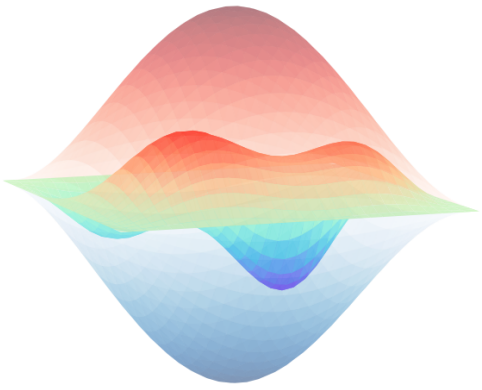


Figure 19: Two functions $v_\alpha(x, y) = \sin \pi x \sin \pi y$ and $v_\beta(x, y) = -\sin \pi x \sin \pi y$ bounding a temperature state $u(x, y; t) = S(t)u_0$ for some arbitrary fixed time t.

Of course, since $\mathcal{D}$ is discrete, we require each point $u(x_i, y_j) \in \mathbb{R}$ lying within an interval $[v_\alpha(x_i, y_j), v_\beta(x_i, y_j)] \subset \mathbb{R}$ for some appropriate finite functions $v_\beta, v_\alpha \in \ell^2(\mathcal{D})$, shown below:

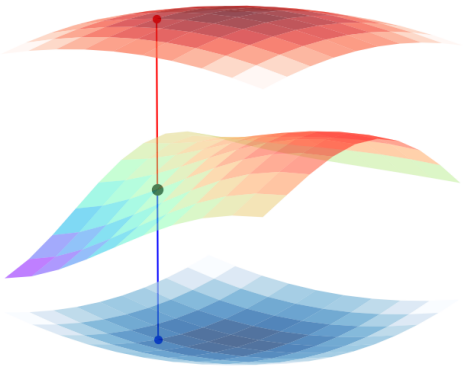


Figure 20: For $D \subset \mathcal{D}$ we view the graphs $(D, v_\alpha(D))$ in red, $(D, u(D))$ in rainbow, and $(D, v_\beta(D))$ in blue. We see the point $u(x_i, y_j)$ lying in the interval $[v_\beta(x_i, y_j), v_\alpha(x_i, y_j)] \subset \mathbb{R}$.

Instead of viewing the *graph* $(\mathcal{D}, u(\mathcal{D})) \subset \mathbb{R}^3$ of a function $u \in \ell^2(\mathcal{D})$ residing entirely within a region bounded by the graphs of a finite number of finite functions $v_\alpha \in \ell^2(\mathcal{D})$, we say that the *function* $u \in \ell^2(\mathcal{D})$ itself lies in a subset $\mathcal{B}$ of the *space of functions* $\ell^2(\mathcal{D})$ which is bounded by the *functions* $v_\alpha, v_\beta \in \ell^2(\mathcal{D})$. That is, elements in the set $\mathcal{B} \subset \ell^2(\mathcal{D})$ are functions whose graphs are bounded between a finite number of graphs of, say, the two functions $v_\alpha, v_\beta \in \ell^2(\mathcal{D})$.

We know by [2] that the semi-dynamical system (42) is dissipative (i.e. possesses a compact absorbing set $\mathcal{B} \subset \ell^2(\mathcal{D})$). Thus there exists a *global attractor* $\mathcal{A}_S \subset \ell^2(\mathcal{D})$ for the semi-group $S(t)$. The global attractor $\mathcal{A}_S \subset \ell^2(\mathcal{D})$ is a finite collection of functions $g_\alpha \in \ell^2(\mathcal{D})$ whose graphs give the plane $\mathcal{P}_\mathcal{A} = \{(x, y, z) \in \mathbb{R}^3 | z = 0\}$. In other words, the heat distribution over the domain $\mathcal{D} \subset \mathbb{R}^2$ will *dissipate* to a temperature of zero at every point $(x_i, y_j) \in \mathcal{D}$.

In this sense, the only function we require to be in $\mathcal{A}_S$ is the constant function $g : \mathcal{D} \to \mathbb{R}$ given by

$$g(x_i, y_j) = 0 \quad \forall\ (x_i, y_j) \in \mathcal{D}.$$

6.4.2 Dissipation of the Nudged Heat Equation

We have confirmed empirically that the nudged heat solution (40) converges in time to the true heat solution (34) (e.g. in Figure 18). Now that we know that the solution operator $S(t_n)$ for the true heat equation is dissipative, we can recreate the proof for Theorem 1 given in Section 4.2.

Assume $u = u(x_i, y_j, t_n)$ solves (30) and assume $w = w(x_i, y_j, t_n; u)$ solves (36). That is, assume for every time $t_n \in \{\delta_t n\}_{n=0}^\infty$ there exist solution operators $S(t_n) : \ell^2(\mathcal{D}) \to \ell^2(\mathcal{D})$ and $Z(t_n) : \ell^2(\mathcal{D}) \to \ell^2(\mathcal{D})$ given by

$$S(t_n)u_0(x_i, y_j) = \alpha\delta_t\Big(\frac{u^n_{i+1,j} - 2u^n_{i,j} + u^n_{i-1,j}}{\delta_x^2} + \frac{u^n_{i,j+1} - 2u^n_{i,j} + u^n_{i,j-1}}{\delta_y^2}\Big) + u^n_{i,j} \tag{44}$$

$$Z(t_n)w_0(x_i, y_j) = \delta_t\Bigg(\alpha\Big(\frac{w^n_{i+1,j} - 2w^n_{i,j} + w^n_{i-1,j}}{\delta_x^2} + \frac{w^n_{i,j+1} - 2w^n_{i,j} + w^n_{i,j-1}}{\delta_y^2}\Big) - \mu\big(w^n_{i,j} - u^n_{i,j}\big)\Bigg) + w^n_{i,j} \tag{45}$$

Theorem 4. *Solutions (45) to the heat nudging algorithm (36)-(39) converge to solutions (44) of the true heat equation (30)-(33). That is, $Z(t_n)w_0 \to S(t_n)u_0$ as $t_n \to \infty$ where u_0 and w_0 come from any compact set $K \subset \ell^2(\mathcal{D})$.*

Proof of Theorem 4:
Let $u_0, w_0 \in K \subset \ell^2(\mathcal{D})$ where K is compact in $\ell^2(\mathcal{D})$ (i.e. every open cover of K has a finite sub-cover, where an example of (the graphs of) such a finite-subcover is shown in Section 6.4.1). The semi-group of solution operators $\{S(t_n)\}_{t\geq 0}$ defined in (44) is dissipative, ensuring that the solution $u(x_i, y_j, t_n)$ traced out by $S(t_n)$ will approach the zero function. That is, $S(t_n)u_0 \to \mathbf{0}$ as $t_n \to \infty$.

In other words (and by Proposition 2 and Corollary 3), the semi-group $S(t_n)$ reaches its global attractor $\mathcal{A}_S$ after an appropriate amount of time t_*, giving us

$$|u(t_* + t_m) - S(t_m)g_0| \to 0, \quad t_m \to \infty,$$

where $t_m \geq 0$ and $g_0 \in \mathcal{A}_S \subset \ell^2(\mathcal{D})$ is a function in the global attractor. Next, recall from (41) that the solution operator $Z(t_n)$ can be expressed in terms of $S(t_n)$:

$$Z(t_n)w_0 = S(t_n)w_0 - \delta_t\mu\Big(w(t_n) - u(t_n)\Big).$$

We can apply Proposition 2 and Corollary 3 to the contribution $S(t_n)w_0$ in $Z(t_n)w_0$ since we know its semi-group $\{S(t_n)\}$ to be dissipative. So we know that there exists a time $\tau_* \geq 0$ such that

$$|S(\tau_* + \tau_m)w_0 - S(\tau_m)h_0| \to 0, \quad \tau_m \to \infty,$$

where $\tau_m \geq 0$ and $h_0 \in \mathcal{A}_S$. Hence we obtain

$$\begin{aligned}
|Z(t_n)w_0 - S(t_n)u_0| &= \big|S(t_n)w_0 - \delta_t\mu\big(w(t_n) - u(t_n)\big) - S(t_n)u_0\big| \\
&= \big|S(t_n)w_0 - S(t_n)u_0 + \delta_t\mu\big(u(t_n) - w(t_n)\big)\big| \\
&\leq \big|S(t_n)w_0 - S(t_n)u_0\big| + \delta_t\mu\big|w(t_n) - u(t_n)\big|.
\end{aligned}$$

Since $\{S(t_n)\}$ is dissipative and acting on each of w_0 and u_0, sending $t_n \to \infty$ then gives

$$\big|S(t_n)w_0 - S(t_n)u_0\big| \to 0.$$

Now we are left to analyze the term $\delta_t\mu|w(t_n) - u(t_n)| = \delta_t\mu|Z(t_n)w_0 - S(t_n)u_0|$. Applying the same steps to this term as we did above results in the term $(\delta_t\mu)^2|Z(t_n)w_0 - S(t_n)u_0|$.

Continuing the procedure to write $(\delta_t\mu)^k|Z(t_n)w_0 - S(t_n)u_0|$, we choose $\mu < 1/\delta_t$, which gives $(\delta_t\mu)^k \to 0$ as $k \to \infty$. Note that k is independent from n and arises naturally as we continue to expand $|Z(t_n)w_0 - S(t_n)u_0|$.

We obtain

$$|Z(t_n)w_0 - S(t_n)u_0| \to 0, \quad n \to \infty,$$

which was to be proved. This completes the proof of Theorem 4.

6.5 Defining the ONet

We define a branch net $F_B(w) : \mathcal{R} \to \mathbb{R}^j,\ j \in \mathbb{Z}^+$, as in Section 5.3.1, where F_B takes values w representing the (nudged) heat in $\mathcal{R}$ of the sheet of metal at different *sensor values*; points $\mathbf{x}_1, \ldots, \mathbf{x}_m$ from $\mathcal{D}$ at which we have measurements of the temperature w. We take $\mathbf{x}_1, \ldots \mathbf{x}_m$ to be the entire grid $\mathcal{D}$. The output of F_B is a vector $F_B(w) \in \mathbb{R}^j$. These temperatures w represent the nudged solutions.

We define a trunk net $F_T(u) : \mathcal{R} \to \mathbb{R}^j,\ j \in \mathbb{Z}^+$, as in Section 5.3.1, where F_T takes values u representing the heat in $\mathcal{R}$ of the sheet of metal at the same sensor locations used in F_B. The output is also a vector $F_T(u) \in \mathbb{R}^j$. These temperatures u represent the true solutions.

6.5.1 Generating Input Data for the ONet

We generate $K = 100$ random initial conditions (discretized surfaces) $u_0^{(k)} \in \ell^2(\mathcal{D})$ by plotting random points in $\mathbb{R}^3$, fitting a surface to the points using support vector machine, and multiplying the resulting surface by a surface which we know to be zero on the boundary. We then solve each initial value problem using FDM to generate a collection of solutions, shown in blue below. These solutions are used as input into the trunk net. For the branch net, we use $K = 100$ arbitrary (fixed or random) initial conditions $w_0^{(k)} \in \ell^2(\mathcal{D})$, and solve each corresponding FDM nudging algorithm, obtaining a collection of nudged solutions, shown in red below.

We can write the kth true solution as

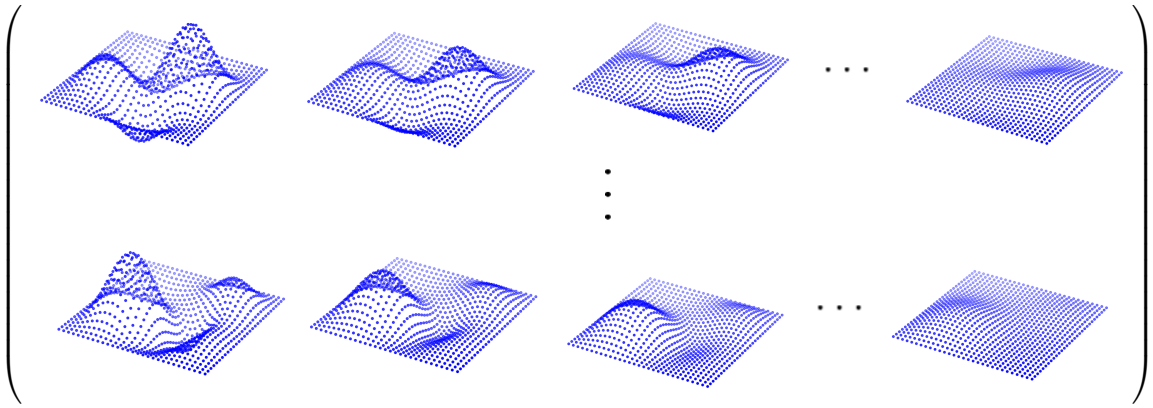
$$\mathcal{U}^{(k)} = \{S(t_n)u_0^{(k)}\}_{t_0 \le t_n \le t_{N_t}} = \{u_0^{(k)}, u_1^{(k)}, u_2^{(k)}, \cdots, u_{N_t}^{(k)}\}.$$

Then we can write the kth nudged solution as

$$\mathcal{W}^{(k)} = \{Z(t_n)w_0^{(k)}\}_{t_0 \le t_n \le t_{N_t}} = \{w_0^{(k)}, w_1^{(k)}, w_2^{(k)}, \cdots, w_{N_t}^{(k)}\}.$$

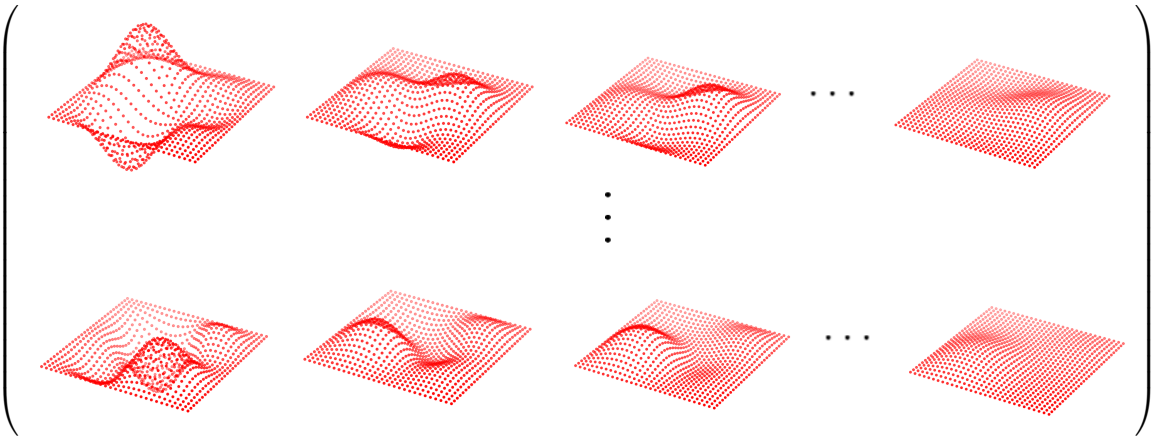
We next write the collection of all $K = 100$ true solutions as

$$\underline{\mathcal{U}} = \{\mathcal{U}_{k=1,\dots,K}^{(k)}\} = \left\{\{S(t_n)u_0^{(k)}\}_{t_0 \le t_n \le t_{N_t}}\right\}_{k=1,\dots,K},$$



and the collection of all $K = 100$ nudged solutions as

$$\underline{\mathcal{W}} = \{\mathcal{W}_{k=1,\dots,K}^{(k)}\} = \left\{\{Z(t_n)w_0^{(k)}\}_{t_0 \le t_n \le t_{N_t}}\right\}_{k=1,\dots,K}.$$



6.6 Training the ONet

We can define the *training data* as the tuple

$$\mathcal{X}^{Train} = \Big(\underline{\mathcal{W}}\ ,\ \underline{\mathcal{U}}\Big),$$

with the corresponding *training label*

$$\mathcal{Y}^{Train} = \underline{\mathcal{W}}^*,$$

where we take $\underline{\mathcal{W}}^*$ to be states of the nudged system which are one step further than those in $\underline{\mathcal{W}}$. That is, the entry $\underline{\mathcal{W}}^{(k)}_{i,j,n}$ is the temperature value of the kth trajectory of the nudging system at location (x_i, y_j) at time t_n, and its *label* is the temperature value of the kth trajectory at the same location *one step further in time*, giving us

$$(\mathcal{X}_0^{Train})^{(k)}_{i,j,n+1} = w^{(k)}_{i,j,n+1} = (\mathcal{Y}^{Train})^{(k)}_{i,j,n}.$$

So when we input a datum (and its label) into the DeepONet, it takes the form

$$\Big[\Big(w^{(k)}_{i,j,n}, u^{(k)}_{i,j,n}\Big), w^{(k)}_{i,j,n+1}\Big].$$

Thus, an input datum is a nudged temperature $w_{i,j,n}$ and a true temperature $u_{i,j,n}$, and its corresponding label is the nudged temperature $w_{i,j,n+1}$ one time-step further. We see this below:

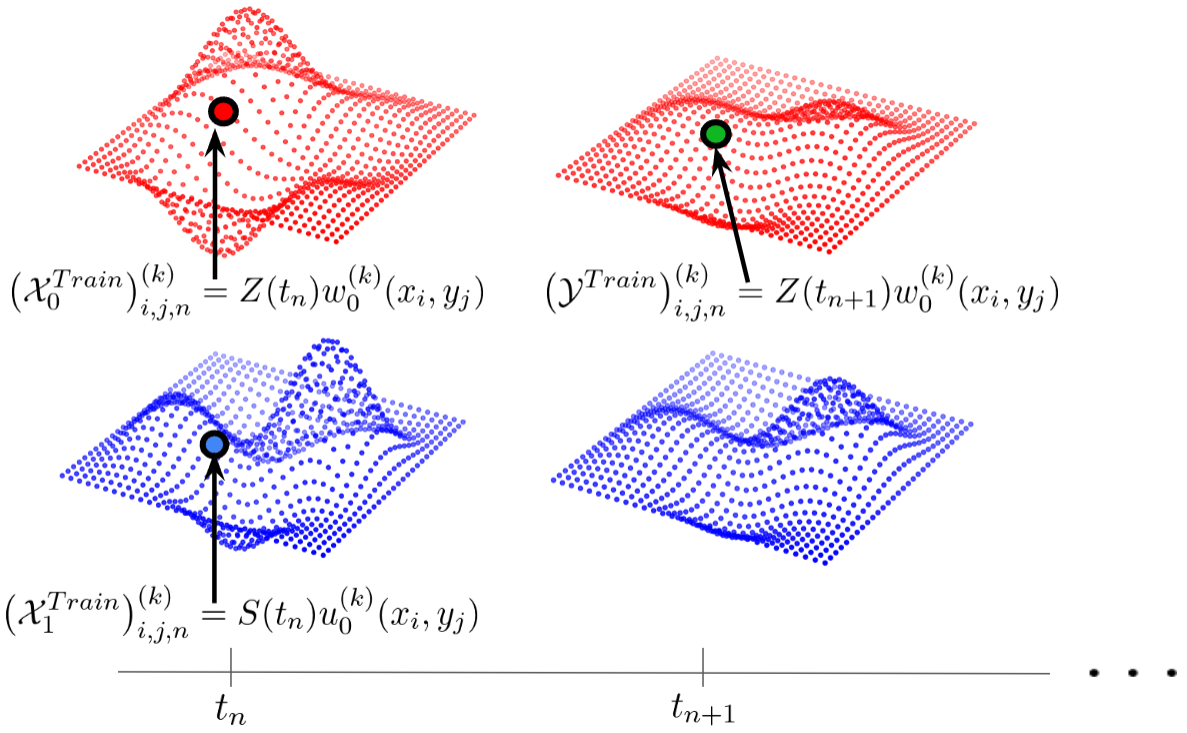


Figure 21: Two states of the kth trajectories (nudged in red, true in blue). The large red dot represents the nudged temperature $w^{(k)}_{i,j,n}$, the large blue dot represents the true temperature $u^{(k)}_{i,j,n}$, and the large green dot represents the nudged temperature $w^{(k)}_{i,j,n+1}$. The tuple $\Big[(\mathcal{X}_0^{Train})^{(k)}_{i,j,n}\ ,\ (\mathcal{X}_1^{Train})^{(k)}_{i,j,n}\Big] = \Big[w^{(k)}_{i,j,n}\ ,\ u^{(k)}_{i,j,n}\Big]$ is the input datum whose label is $(\mathcal{Y}^{Train})^{(k)}_{i,j,n} = w^{(k)}_{i,j,n+1}$.

6.6.1 Parameters, Network Architecture, and Results

We compute $K = 100$ true trajectories and $K = 100$ nudged trajectories. We use a spatial grid of shape $N_x \times N_y = 30 \times 30$ where $\delta_x = \delta_t = 1/30$, a time-step size of $\delta_t = .002$ for $N_t = 100$ time-steps, giving us a final time value of $t_{N_t} = N_t \delta_t = 0.2$. We set the diffusion (heat) proportionality constant $\alpha = 0.1$.

For both the branch and trunk nets, we use hidden layers with numbers of nodes: $20, 400, 200, 100$. We use a *ReLu* activation function, a Glorot-normal kernel initializer, a learning rate of .007, and a batch size of 20 (too large of a batch size will exhaust the CPU's memory). We train for 5000 epochs.

We obtain a training loss (between the ONet prediction and the nudged solution) [1] of 7.41×10^{-7} and a test loss [1] of 1.38×10^{-6}. We also compute the following average (among all $K = 100$ trajectories) ℓ^2 losses at the final time-step:

$$||ONet - Nudged|| = .0209,$$
$$||Nudged - True|| = .0088,$$
$$||True - ONet|| = .014.$$

We can view the average ℓ^2 loss over time:

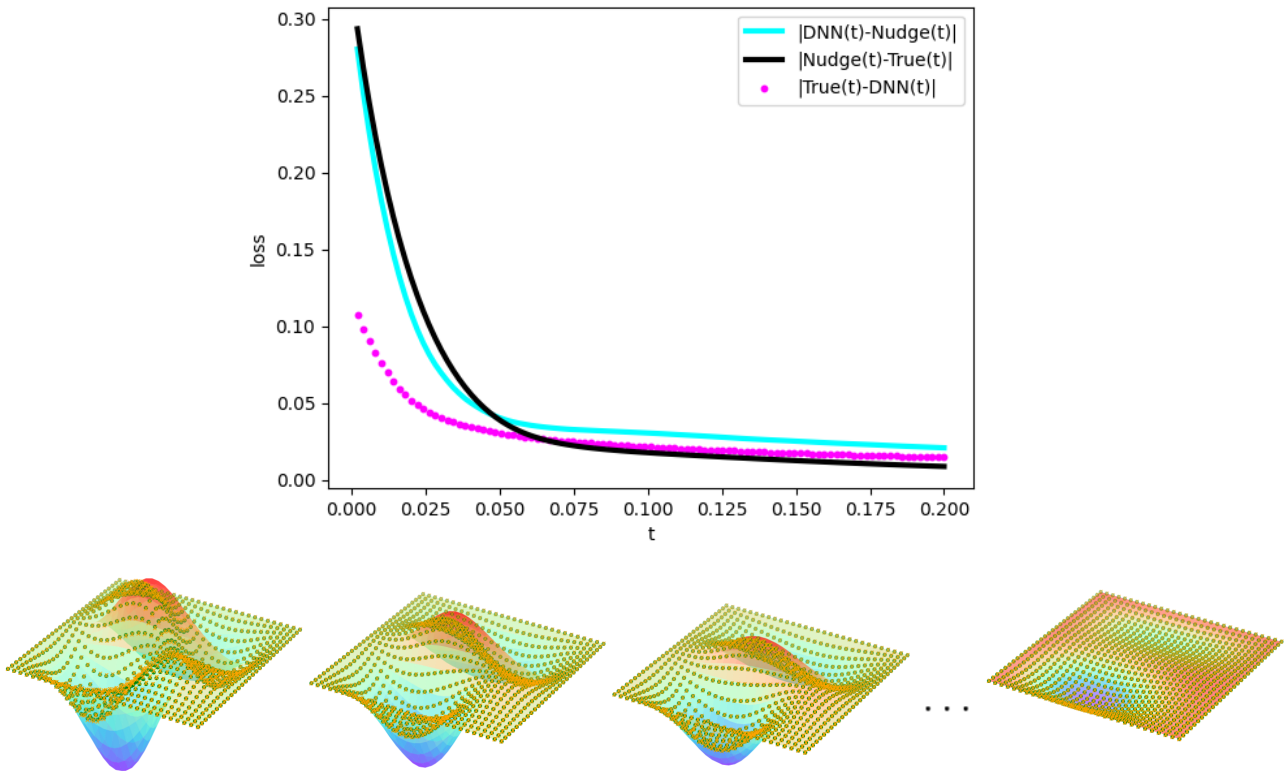
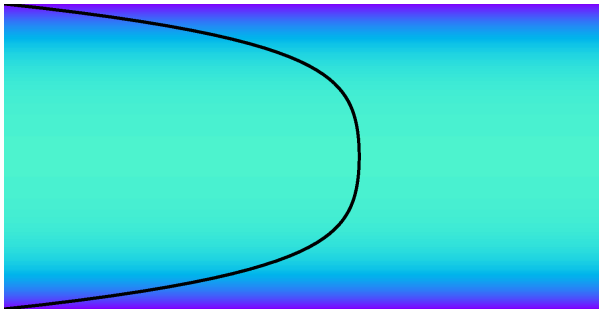


Figure 22: We view the ℓ^2 losses converging in time to zero. We view some states of the trajectories of the true system (rainbow), the nudged system (green), and the ONet prediction (orange). It is hard to visually distinguish between the nudged and ONet states because of how quickly and accurately the ONet predicts its label (nudged solution).

7 Learning Simple Pipe Flow with a DeepONet



We consider a partial differential equation whose solution and phase space define a (semi)dynamical system. The velocity field of a fluid flow is given by

$$\vec{u} = \vec{u}(\vec{x}, t) = \big(u(\vec{x}, t),\ v(\vec{x}, t)\big) \tag{46}$$

$$\vec{x} = \big(x,\ y\big) \in O$$

We use the incompressible Navier-Stokes Equations (NSE)

$$\frac{\partial \vec{u}}{\partial t} + \big(\vec{u} \cdot \nabla\big)\vec{u} = -\frac{1}{\rho}\nabla p + \eta \Delta \vec{u} + \vec{F} \tag{47}$$

$$\nabla \cdot \vec{u} = 0 \tag{48}$$

where $\rho > 0$ is the constant density of the fluid, $\eta > 0$ is its kinematic viscosity, $p = p(\vec{x}, t) \in \mathbb{R}$ is the pressure, and $\vec{F} = (F_1, F_2) \in \mathbb{R}^2$ is a body force, as the PDE which governs our fluid flow in the region O. The top equation represents the momentum of the fluid, and is derived from Newton's Second Law ($F = ma$). The bottom equation represents the conservation of mass, and acts as a constraint satisfying the incompressibility of the fluid. It is also referred to as the divergence-free equation, pointing out that there will be no "compression" or "expansion" (i.e. divergence) of the fluid.

We use finite-difference method (FDM) to approximate the solution to (47),(48) with the no-slip and period boundary conditions:

$$u(x, 0, t) = u(x, h, t) = 0$$

$$v(x, 0, t) = v(x, h, t) = 0$$

$$u(x, y, t) = u(x + L, y, t)$$

$$v(x, y, t) = v(x + L, y, t)$$

For this simple problem, we could have also made physical assumptions to reduce the PDE to one solvable in closed-form. However, the finite-difference scheme is adaptable to data-assimilation, so we use FDM to leave the topic open for future study.

7.1 Training and Using the ONet

We now have three scalar fields to learn; U, V, and P, where $\vec{U} = (U, V)'$ is the velocity vector field and P is the pressure scalar field. Hence, we train three DeepONets, one to learn each of U, V, and P. Since there is no vertical component of the velocity in simple pipe flow, loss for the V ONet is zero.

We construct our input and label in the following way: The spatial-temporal domain $\mathcal{D} \times \mathcal{T} = \{x_1, \ldots, x_{N_x}\} \times \{y_1, \ldots, y_{N_y}\} \times \{t_0, \ldots, t_{N_t}\}$ is of shape (N_x, N_y, N_t). We take $N_x N_y$ copies of $\mathcal{T}$, $N_x N_t$ copies of $\mathcal{D}_y$, and

$N_y N_t$ copies of $\mathcal{D}_x$. Then we define a matrix $[D]_{i=0,j=1}^{N_x N_y N_t, 3}$ where, say the ith row denotes the location (x, y, t) with index

$$\begin{pmatrix} N_y N_t + i \pmod{N_x} \\ N_x N_t + N_x i \pmod{N_x N_y} \\ N_x N_y i \pmod{N_x N_y N_t} \end{pmatrix}.$$

In essence, we stretch out our spatial-temporal domain and align it with the (similarly flattened) trajectory $U(x, y, t)$ (or $V(x, y, t)$ or $P(x, y, t)$ for the vertical velocity or pressure, respectively). In this way, we ensure that for every spatial-temporal "location" $(\vec{x}, t)$, we associate the correct "fluid velocity" $\vec{U}(\vec{x}, t)$ (or pressure $P(\vec{x}, t)$). A discrete solution U (or V or P) will be of shape (N_t, N_x, N_y), and its domain will be of shape $(3, N_t, N_x, N_y)$ since each position (x, y, t) is an ordered triplet. Thus one trajectory requires $4N_t N_x N_y$ bits of memory.

7.1.1 Parameters and Results

We use four hidden layers of shapes $(100, 200, 200, 100)$ for both the branch net and the trunk net. We use a *ReLu* activation and an AdaM optimizer. We train for 14000 epochs for the U component, 200 epochs for the V component (zero), and 1000 epochs for the P component. We use a batch size of 100 and a learning rate of .01. We obtain a summed ℓ^2 loss across all time stamps of less than 3. We view the last time-step of the trajectory below:

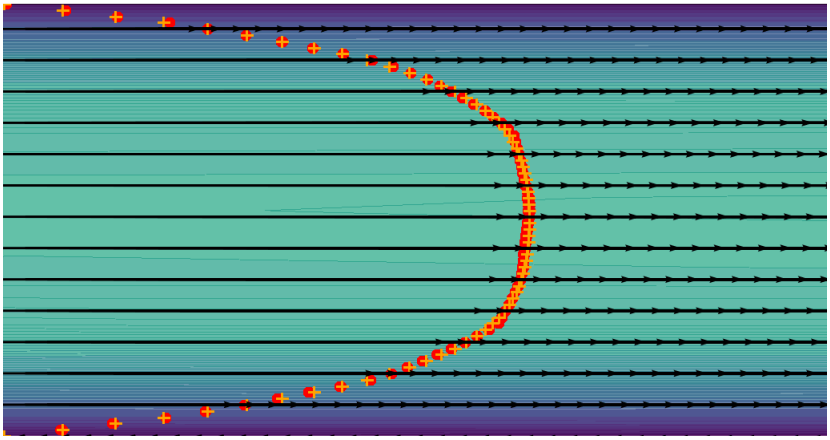


Figure 23: The contour plot and the red scatter plot represent the true U velocity, and the orange scatter plot represents the ONet's prediction of the U velocity at the last time-step (i.e. approaching a steady state).

References

[1] Lu L, Jin P, Em Karniadakis G. *DeepONet: Learning nonlinear operators for identifying differential equations based on the universal approximation theorem of operators.* CoRR. 2019.

[2] Robinson J. *Infinite-Dimensional Dynamical Systems: An Introduction to Dissipative Parabolic PDEs and the Theory of Global Attractors.* Cambrudge University. 2001.

[3] Antil H, Lohner R, Price R. *Data Assimilation with Deep Neural Nets Informed by Nudging.* arXiv. 2021.

[4] Pierce, Anthony. *Lecture 8: Solving the Heat equations using finite difference methods.* University of British Columbia. 26 January 2018.

[5] Clack, Christopher. *Dynamics of the Lorenz Equations.* University of Manchester. 2006.