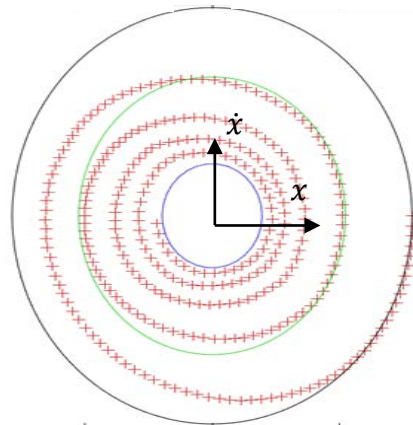


Physics-Informed Machine Learning

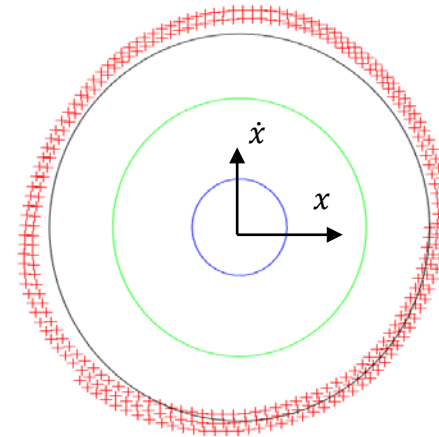
Using the laws of nature to improve generalized deep learning models

Traditional ML



vs.

Physics-Informed ML



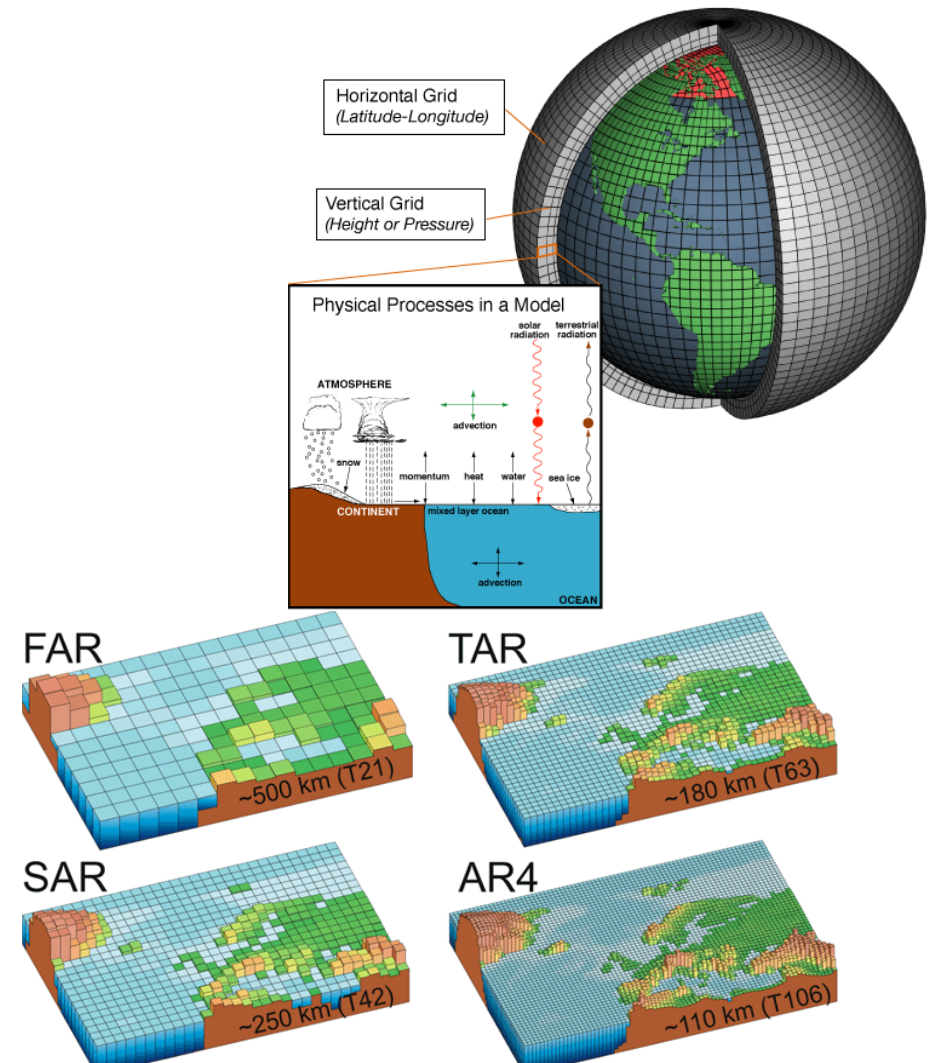
Introduction - Fusing Data and Simulation

Climate Models

- Attention is now being drawn to using ML with scientific data for better predictive power
- ML can be used as simulators to generate a prediction of the updated state of a system
- This can be useful in cases like climate models where simulations are expensive and slow
- However, these ML models are trained only to minimize the error between data and predictions, no physical information is used to inform this training
- This lack of physics-informed machine learning can limit the extrapolation efficacy
- Adding some physics into the training can aid this workflow for more robust predictions

Climate modeling

- Huge domains
- Billions of variables
- Massive time scales

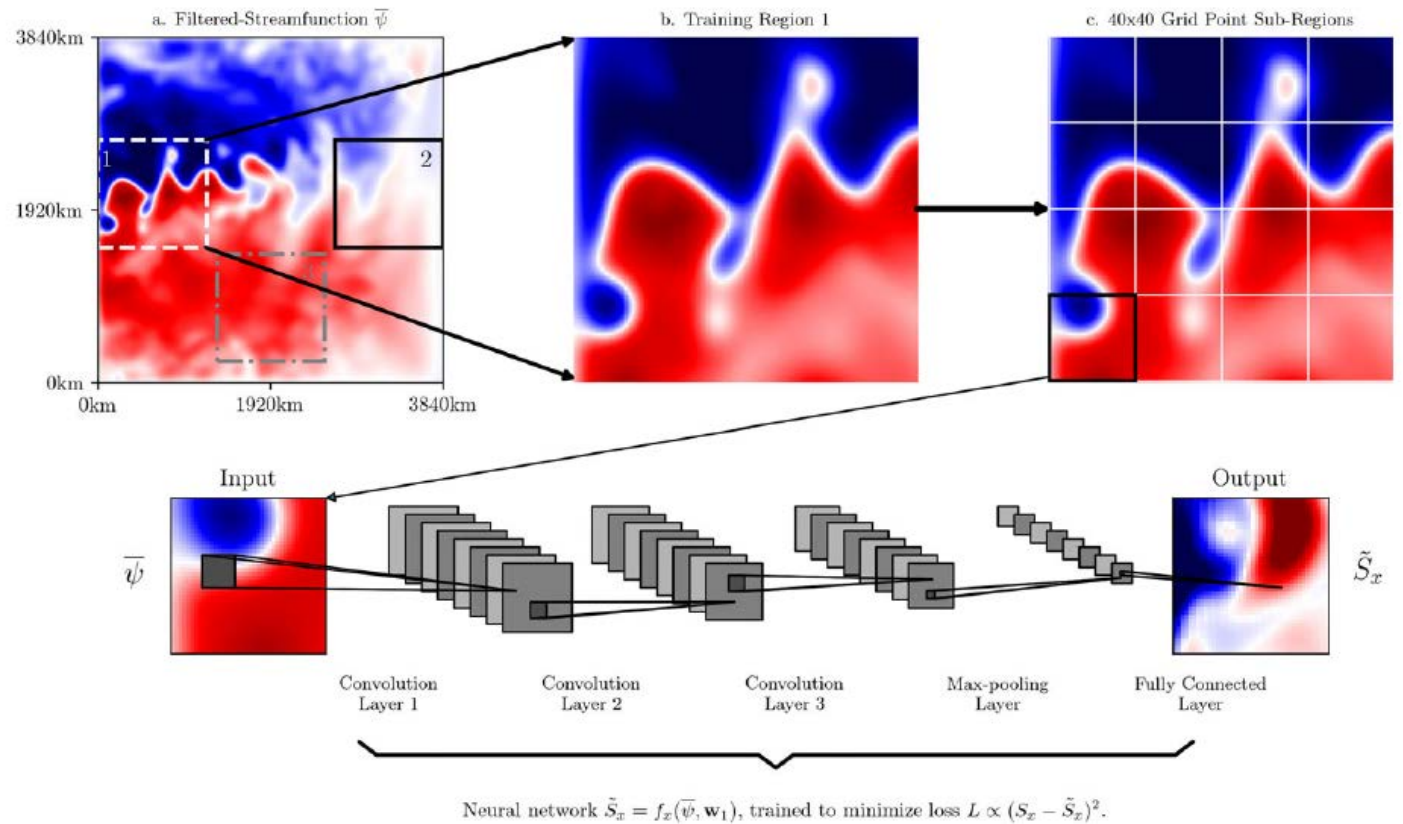


Introduction – Fusing Data and Simulation

Typically, these models involving solving complex CFD systems with interacting scales is extremely costly.

A new approach:

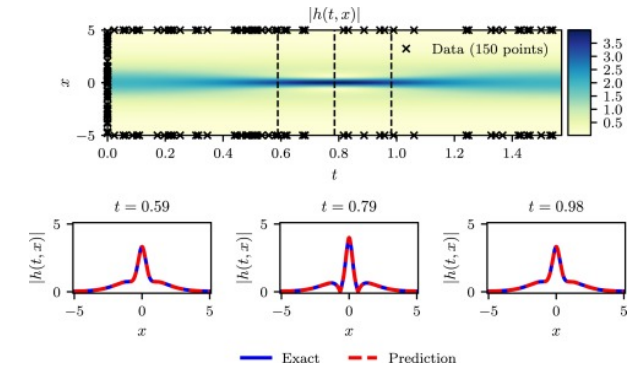
- Deep Learning-powered simulations
- Using data to power more efficient solutions
- Learn on one type of weather to help predict new weather patterns



Introduction – Physics Informed Machine Learning

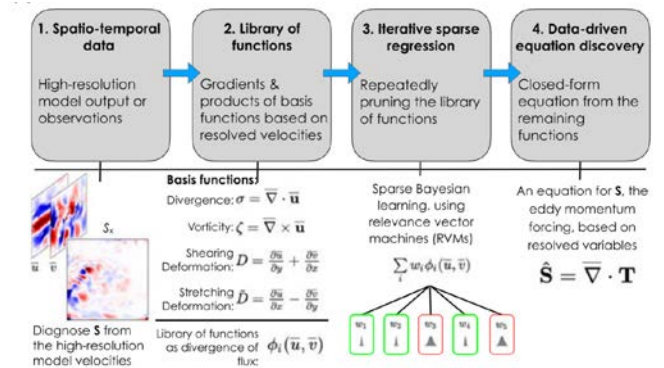
Physics-Informed Neural Networks

M. Raissi, P. Perdikaris, G.E. Karniadakis, Physics-informed neural networks: A deep learning framework for solving forward and inverse problems involving nonlinear partial differential equations, *Journal of Computational Physics*, Volume 378, 2019. <https://doi.org/10.1016/j.jcp.2018.10.045>



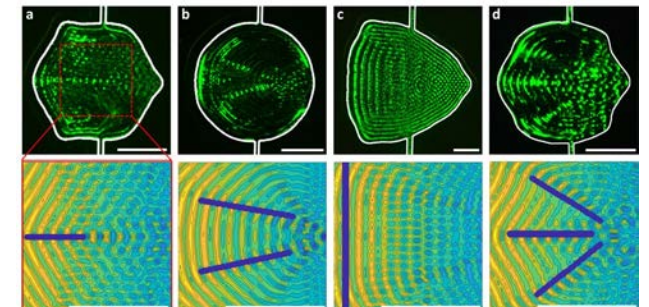
Learning equations

Zanna, L., Bolton, T. (2020). Data-driven equation discovery of ocean mesoscale closures. *Geophysical Research Letters*, 47, e2020GL088376. <https://doi.org/10.1029/2020GL088376>



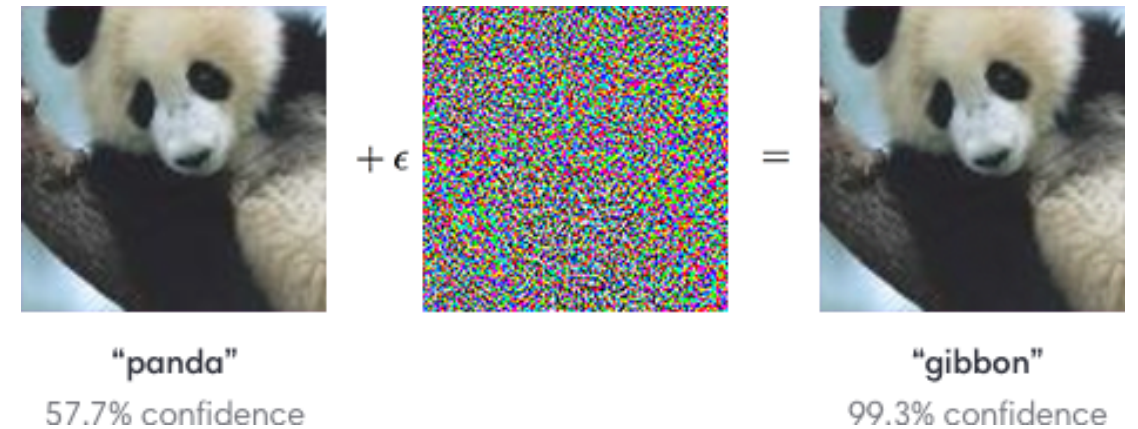
Synthetic data for inverse-design

Raymond, S.J., Collins, D.J., O'Rourke, R. *et al.* A deep learning approach for designed diffraction-based acoustic patterning in microchannels. *Sci Rep* 10, 8745 (2020). <https://doi.org/10.1038/s41598-020-65453-8>



How well can we trust ML-infused physics?

- A system cannot be well predicted by simply learning the data for a physical system.
- This system obeys laws that simply fitting data may not learn
- When using this model to extrapolate, this can lead to divergences
- This may be mediated by adding some existing knowledge to the training process



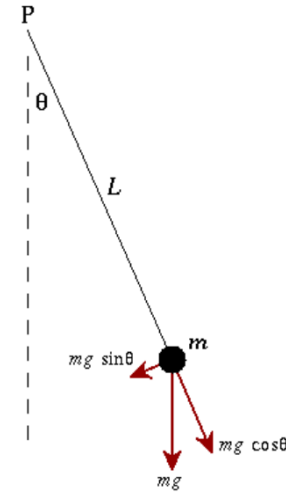
Conventional neural networks can be brittle and vulnerable to *adversarial* attacks.

A simpler example – A pendulum

Complex systems like climate models can be represented by much simpler models.

These models are typically mechanical systems where kinetic and potential energy oscillate. Pendulums and other oscillators are used often in physics to understand more complex processes.

Here we will focus on the prediction of the motion of a pendulum released from different heights.



Equation of motion

$$\ddot{\theta}(t) + \frac{g}{L} \sin \theta(t) = 0$$

Energy of the system

$$E = \frac{1}{2} m \dot{\theta}^2 + mg(1 - \cos(\theta))$$

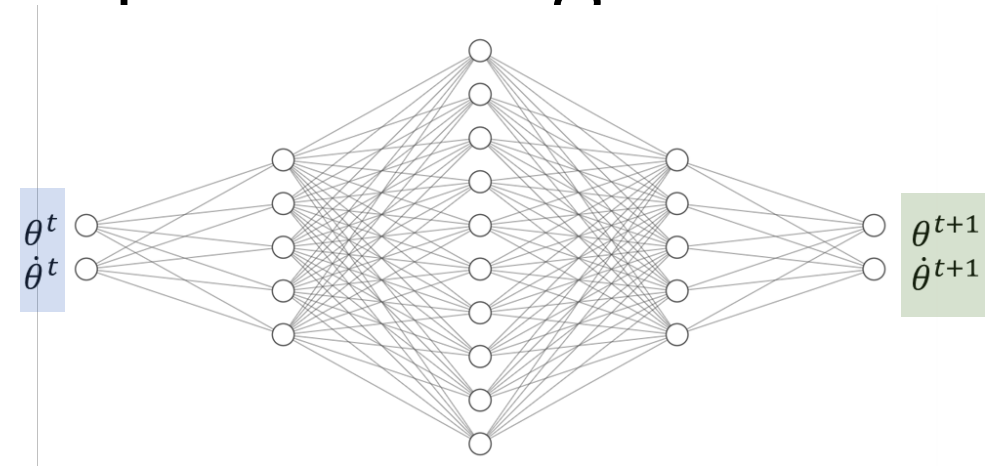
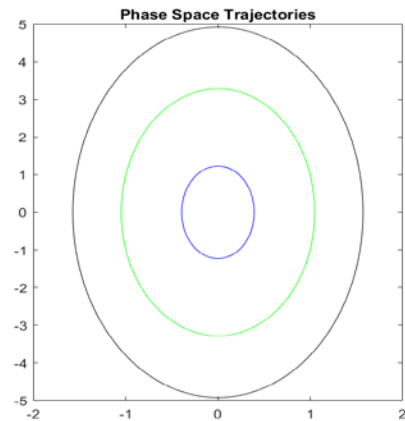
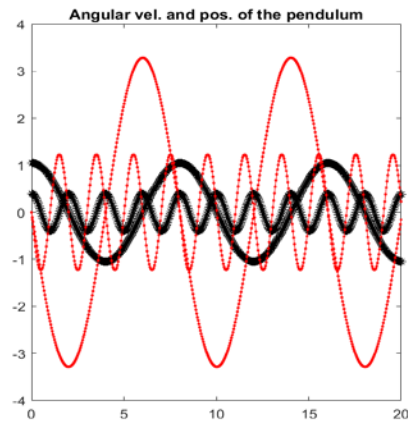
Results – Conventional Deep Learning

Input is angular position and velocity

Output is angular position and velocity

Network is acting like a time integrator

Dataset consists of pendulum trajectories released from different heights.



Network Architecture

Input layer: 2 input neurons for angle and angular velocity

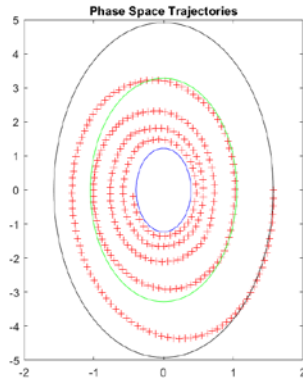
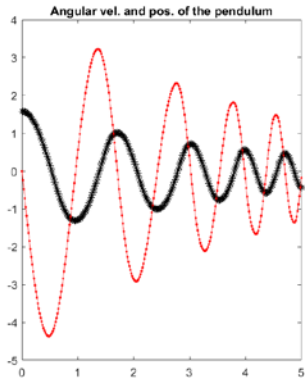
Internal layers: 3 fully connected hidden layers with ReLU activation of the form : 50,100,50

Output Layer: 2 output neurons for angle and angular velocity

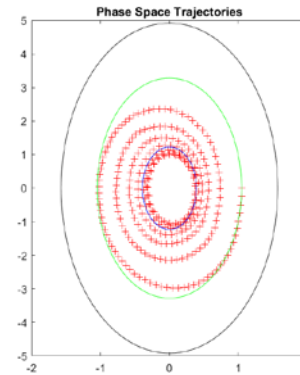
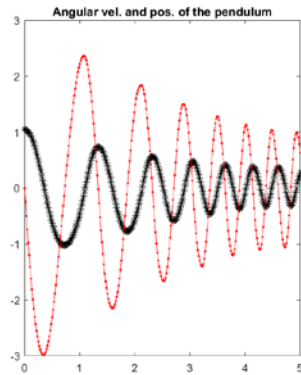
Training with an ADAM optimizer and the commonly-used mean squared error loss function:

$$Loss_{mse} = |Y - T|^2$$

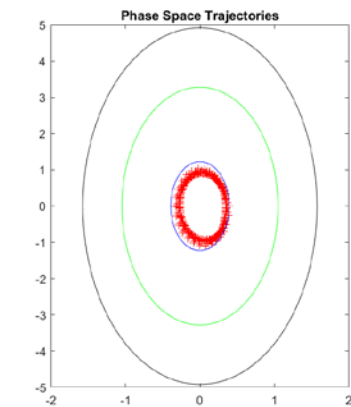
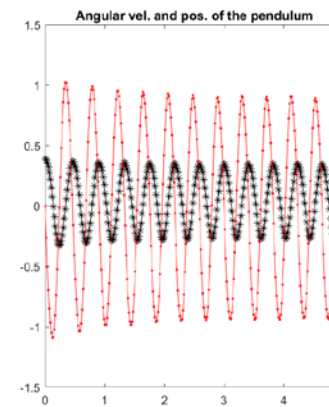
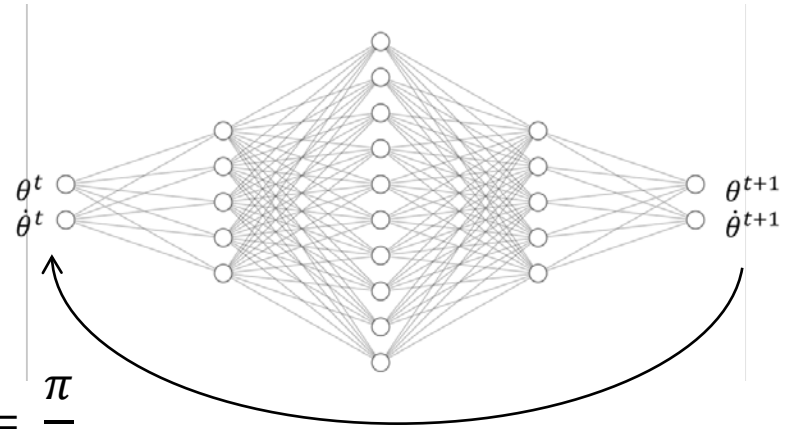
Results – Conventional Deep Learning



$$\theta^{t=0} = \frac{\pi}{2}$$
$$\dot{\theta}^{t=0} = 0$$



$$\theta^{t=0} = \frac{\pi}{3}$$
$$\dot{\theta}^{t=0} = 0$$



$$\theta^{t=0} = \frac{\pi}{6}$$
$$\dot{\theta}^{t=0} = 0$$

Predicted results decay from desired trajectory to a dominant (minimal) energy trajectory

Solution – Modified Loss Function

The traditional loss function seeks to minimize the predicted value (Y) from the target/true value (T):

$$Loss = f(Y - T)$$

However, we want to ensure that there is a notion that other factors are important, such as the conservation of energy. In a mechanical system like the pendulum the energy (E) can be expressed as a sum of the kinetic and potential terms:

$$E = \frac{1}{2}m\dot{\theta}^2 + mg(1 - \cos(\theta))$$

The new loss, therefore, is a function of the values Y,T and the energy, E:

$$Loss = f(Y - T, E_Y - E_T)$$

$$Loss_{mse} = |Y - T|^2$$

$$Loss_{PIML} = (1 - \lambda)Loss_{mse} + \lambda|E_Y - E_T|^2$$

```
layers = [...  
    sequenceInputLayer(2)  
    fullyConnectedLayer(50);  
    reluLayer;  
    fullyConnectedLayer(100);  
    reluLayer;  
    fullyConnectedLayer(50);  
    reluLayer;  
    fullyConnectedLayer(2);  
    mse_energyConsvLoss('dE');  
];
```

```
classdef energyConsvLoss < nnet.layer.RegistrationLayer  
    properties  
        % (Optional) Layer properties.  
        % Layer properties go here.  
    end  
    methods  
        function layer = energyConsvLoss(name)  
            % Creates a physics-based loss function  
            % that includes the conservation of Energy  
            % Layer constructor function goes here.  
            layer.Name = name;  
            layer.Description = 'MAE + Energy Conservation loss';  
        end  
        function loss = forwardLoss(layer, Y, T)  
            R = size(Y,3);  
            meanAbsoluteError = sum(abs(Y-T),3)/R;  
            % Take mean over mini-batch.  
            N = size(Y,4);  
            MAEloss = sum(meanAbsoluteError)/N;  
            % Layer forward loss function goes here.  
            energy_system_pred = -9.81.*cos(Y(1,:)) + (1/2).*(Y(2,:).^2);  
            pred_energy = sum(energy_system_pred)/N;  
            energy_system_true = -9.81.*cos(T(1,:)) + (1/2).*(T(2,:).^2);  
            true_energy = sum(energy_system_true)/N;  
            e_factor = 1.0e-5;%e-8;%2*N;  
            L = 1;  
            energy_loss = e_factor*(abs(true_energy - pred_energy));  
            loss_mag = sqrt(energy_loss^2 + MAEloss^2);  
            loss = L*MAEloss*MAEloss/loss_mag + (1-L)*energy_loss*energy_loss/loss_mag;  
        end  
    end  
end
```

Tools Used

FILE NAVIGATE TEXT CODE SECTION RUN

Current Folder: C:\Users\Sam\Dropbox\MATLAB_EXPO_2021\matlab_code

Live Editor: C:\Users\Sam\Dropbox\MATLAB_EXPO_2021\matlab_code\pgml_physics_eng_output_newsol_test2021feb.mlx

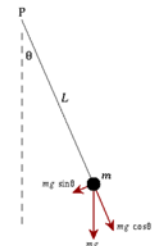
Workspace: Name -

Improving Generalizability of Neural Networks with Physics-Based Loss Functions

Lessons from Feb 12: with lots of data the normal net is better for long and short extrapolations. But when the dataset is small the pml net is better

Step 0: The trajectory of a simple pendulum

The governing equation for a simple pendulum as shown in the figure below can be expressed as a function of the angular position, $\theta(t)$, and the angular acceleration, $\ddot{\theta}(t)$

$$\ddot{\theta}(t) + \frac{g}{L} \sin(\theta(t)) = 0$$


The solution to this equation can be found as the analytical harmonic equation $\theta(t) = \theta_0 \cos(\omega t)$ with ω being the natural frequency of the pendulum: $\omega = \sqrt{\frac{g}{L}}$.

Therefore, for different values of ω , the pendulum has a defined, fixed, trajectory in phase space, with the natural frequency acting as the energy of the particular phase trajectory.

```

1 retrain_regenerate=true;
2
3 if retrain_regenerate
4 close all; clear; clc;
5 sample_size=700;
6 PREDICTION_LENGTH=5*sample_size;
7 t_max=50;
8 retrain_regenerate=true;
9 % Initial Conditions
10 theta_0=pi/4; % Initial angle in radians, measured from the vertical.
11 thetadot_0=0.0; %Initial velocity of the pendulum.
12
13 % Pendulum Parameters
14 length=1.0;

```

Step 1: Learning the trajectory of a simple pendulum

Now that we have a simulator that can produce the data, we want to build a neural network that can predict the next position and velocity of the pendulum when given the current position and velocity of the pendulum.

If we take the first solution that we made as our dataset, we have two signals, angular position and velocity, that show a pattern that emerges over 10 periods. We will cut this data in the form 70:15:15 for training:validation:testing.

```

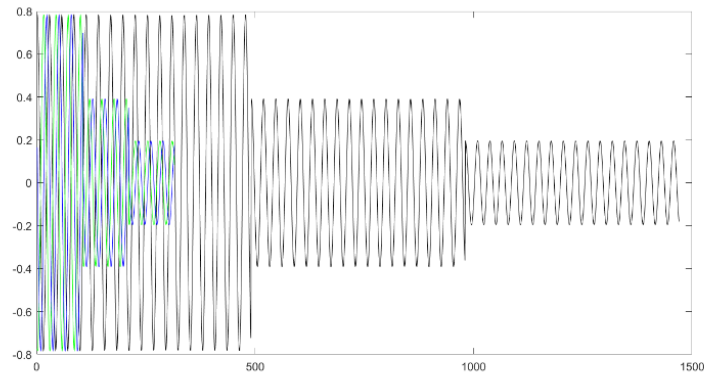
if retrain_regenerate
pos_train = [theta_sol_0(1:round(0.7*sample_size)),theta_sol_1(1:round(0.7*sample_size)),theta_sol_2(1:round(0.7*sample_size))];
vel_train = [thetadot_sol_0(1:round(0.7*sample_size)),thetadot_sol_1(1:round(0.7*sample_size)),thetadot_sol_2(1:round(0.7*sample_size))];
pos_val = [theta_sol_0(round(0.7*sample_size)+1:round(0.85*sample_size)),theta_sol_1(round(0.7*sample_size)+1:round(0.85*sample_size))];
vel_val = [thetadot_sol_0(round(0.7*sample_size)+1:round(0.85*sample_size)),thetadot_sol_1(round(0.7*sample_size)+1:round(0.85*sample_size))];
pos_test = [theta_sol_0(round(0.85*sample_size)+1:end),theta_sol_1(round(0.85*sample_size)+1:end),theta_sol_2(round(0.85*sample_size)+1:end)];
vel_test = [thetadot_sol_0(round(0.85*sample_size)+1:end),thetadot_sol_1(round(0.85*sample_size)+1:end),thetadot_sol_2(round(0.85*sample_size)+1:end)];
t_predict = linspace(0,t_max,PREDICTION_LENGTH);
end

```

```

figure;
plot(pos_train,'k');
hold on;
plot(pos_val,'g');
hold on;
plot(pos_test,'b');
set(gcf,'position',[0,0,1000,500]);

```



Now that we have split the original dataset, we still need to split this again to differentiate between the input and the target to the neural network. For this approach we will use each timestep as the input and the next timestep as the output. To do this we just need to offset the input and target arrays by one so that the two arrays are aligned and are of the same size. We will need to do this for the training, validation, and testing sets.

Live scripts

Create Live Scripts in the Live Editor

Live scripts are program files that contain your code, output, and formatted text together in a single interactive environment called the Live Editor. In live scripts, you can write your code and view the generated output and graphics along with the code that produced it. Add formatted text, images, hyperlinks, and equations to create an interactive narrative that you can share with others.

Benefits of Live Scripts

- **Interactivity:** You can interact with your code and output in real time, allowing you to explore different scenarios and visualize results as you work.
- **Collaboration:** Live scripts are easy to share with others, making it simple to collaborate on projects and share your findings.
- **Documentation:** You can include formatted text, images, and hyperlinks to create a rich, interactive narrative that documents your work.
- **Flexibility:** Live scripts can be used for a wide range of applications, from data analysis and visualization to scientific computing and engineering.

Create Live Script

To create a live script in the Live Editor, go to the Home tab and click **New Live Script**. You can also use the `edit` function in the Command Window. For example, type `edit('penney.mlx')` to open or create the file `penney.mlx`. To ensure that a live script is created, specify an `.mlx` extension. If an extension is not specified, MATLAB® defaults to the `.m` extension, which only supports plain code.

Open Existing Script as Live Script

If you have an existing script, you can open it as a live script in the Live Editor. Opening a script as a live script creates a copy of the file and leaves the original file unaltered. MATLAB converts publishing markup from the original script to formatted content in the new live script.

To open an existing script (such as `penney.mlx`) from the Editor, right-click the document tab and select **Open script here as Live Script** from the context menu. Alternatively, go to the Editor tab, click **Save**, and select **Save as Live Script**. Then, set the Save as type to MATLAB Live Code (*.mlx) and click **Save**.

Deep Learning Toolbox

MathWorks | Products | Solutions | Academic | Support | Community | Events

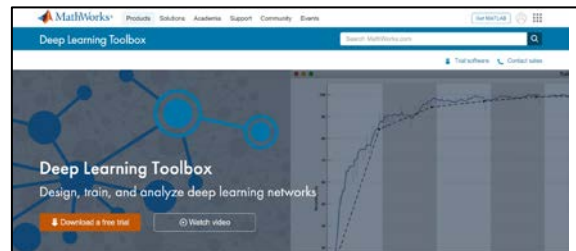
Deep Learning Toolbox

Search MathWorks.com

Deep Learning Toolbox

Design, train, and analyze deep learning networks

Download a free trial | Watch video

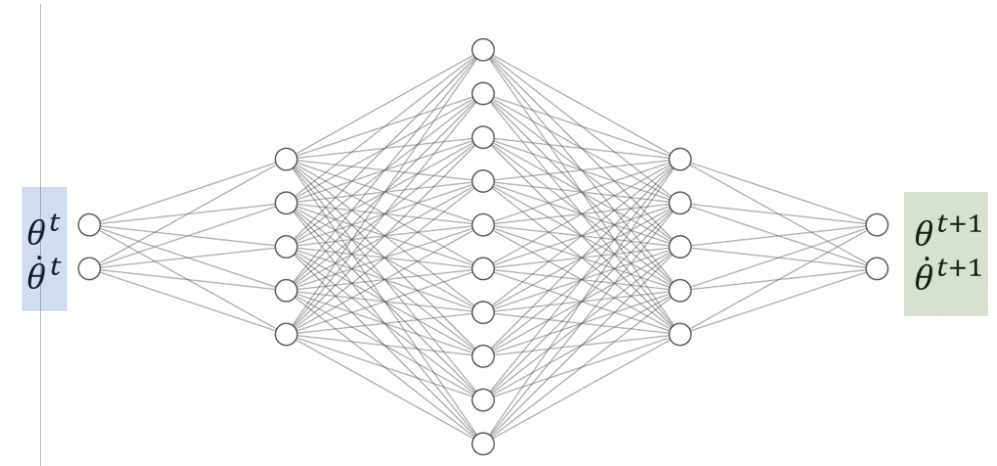


Results – PIML Deep Learning

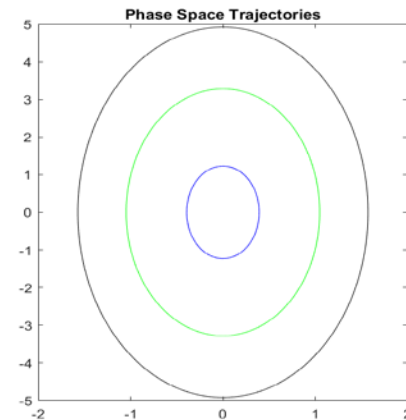
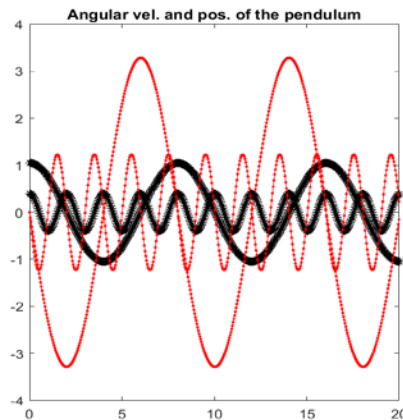
Input is position and velocity

Output is position and velocity

Network is acting like a time integrator



Dataset same as before.

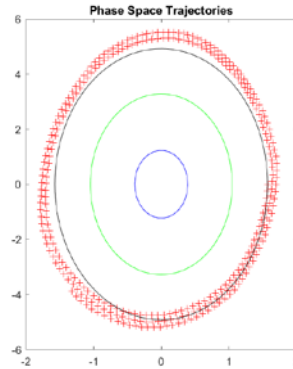
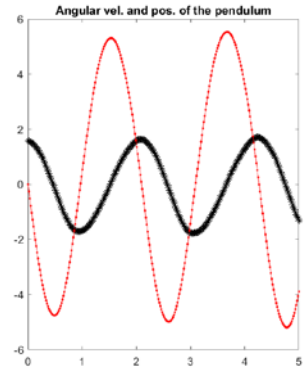


Training with the new PIML loss function:

$$Loss_{mse} = |Y - T|^2$$

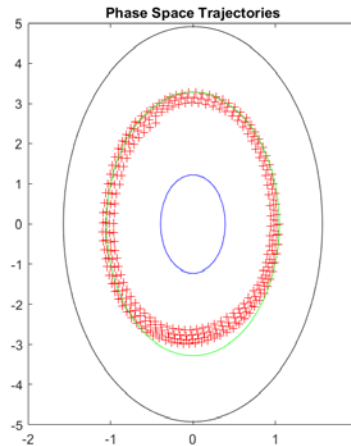
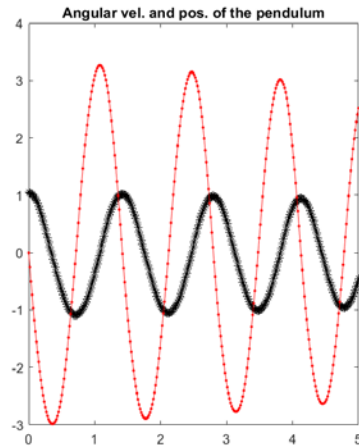
$$Loss_{PIML} = (1 - \lambda)Loss_{mse} + \lambda|E_Y - E_T|^2$$

Results – PIML Deep Learning



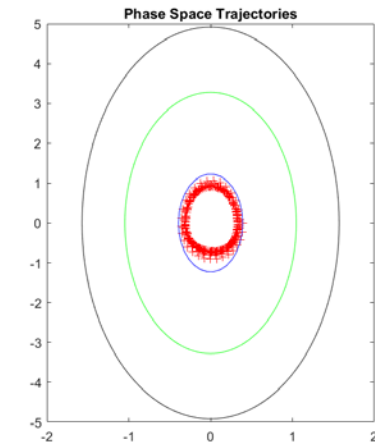
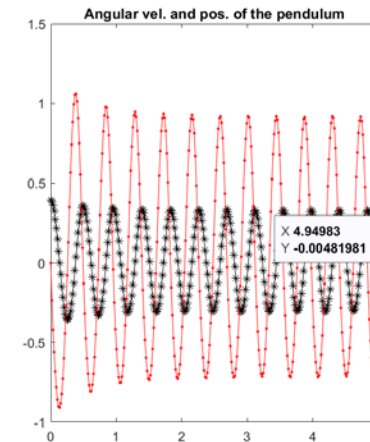
$$\theta^{t=0} = \frac{\pi}{2}$$

$$\dot{\theta}^{t=0} = 0$$



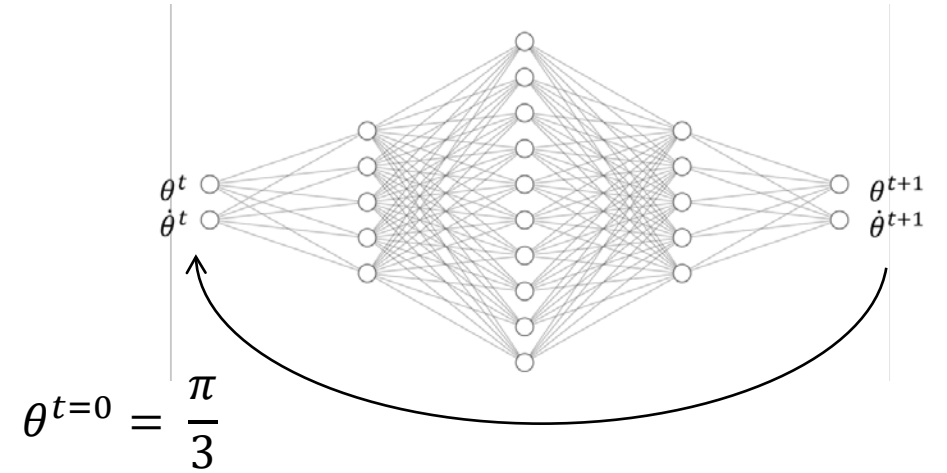
$$\theta^{t=0} = \frac{\pi}{3}$$

$$\dot{\theta}^{t=0} = 0$$



$$\theta^{t=0} = \frac{\pi}{6}$$

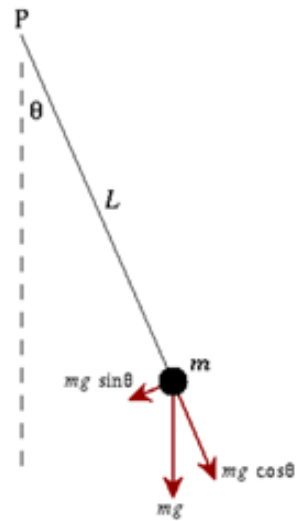
$$\dot{\theta}^{t=0} = 0$$



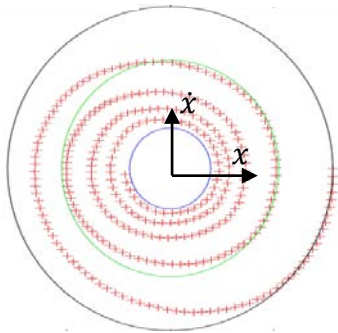
Predicted results no longer decay to the smallest value. Energy is maintained for the prediction cycle

Takeaways - The Future of PIML

Physics-Informed Machine Learning is still very young but full of potential...

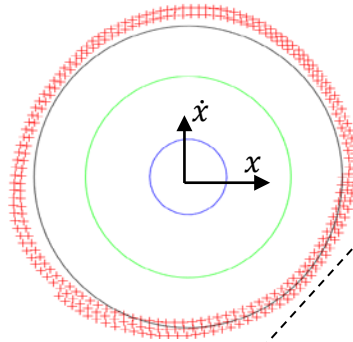


Traditional ML



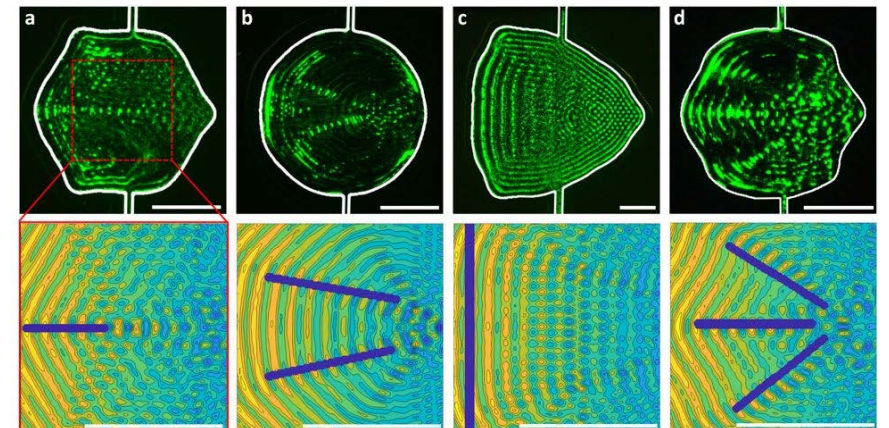
VS.

Physics-Informed ML



PIML-based loss functions for more robust predictions in hybrid (simulation/data) systems

MATLAB-powered PIML-based design tools in biomedical engineering



[Read More](#)

Thank you!