Physics-Informed Neural Networks (PINNs)

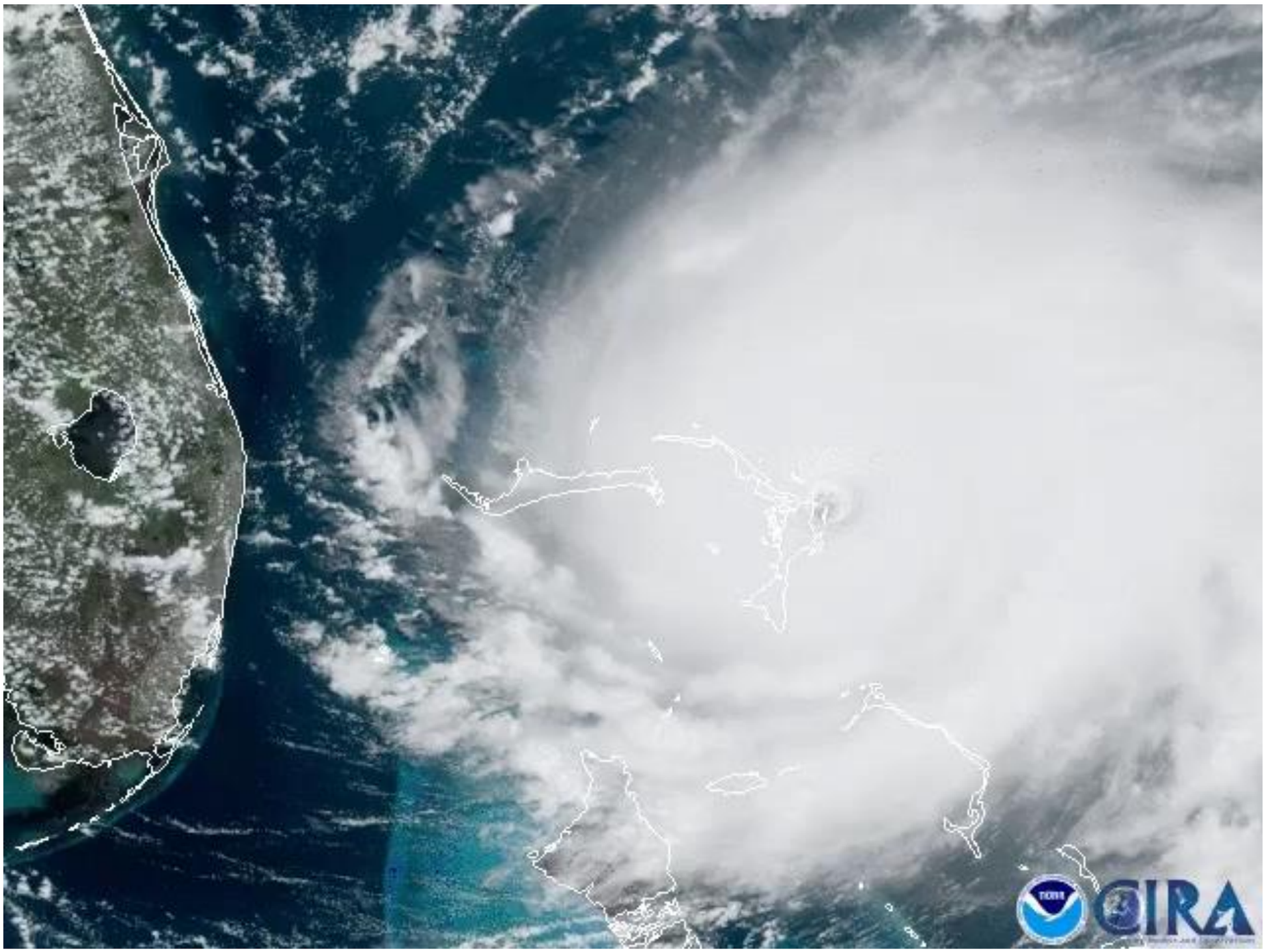
George Em Karniadakis

Division of Applied Mathematics, Brown University (also @MIT & PNNL)

The **CRUNCH** group: **Home of "Math + Machine Learning + X"**

https://www.brown.edu/research/projects/crunch/home

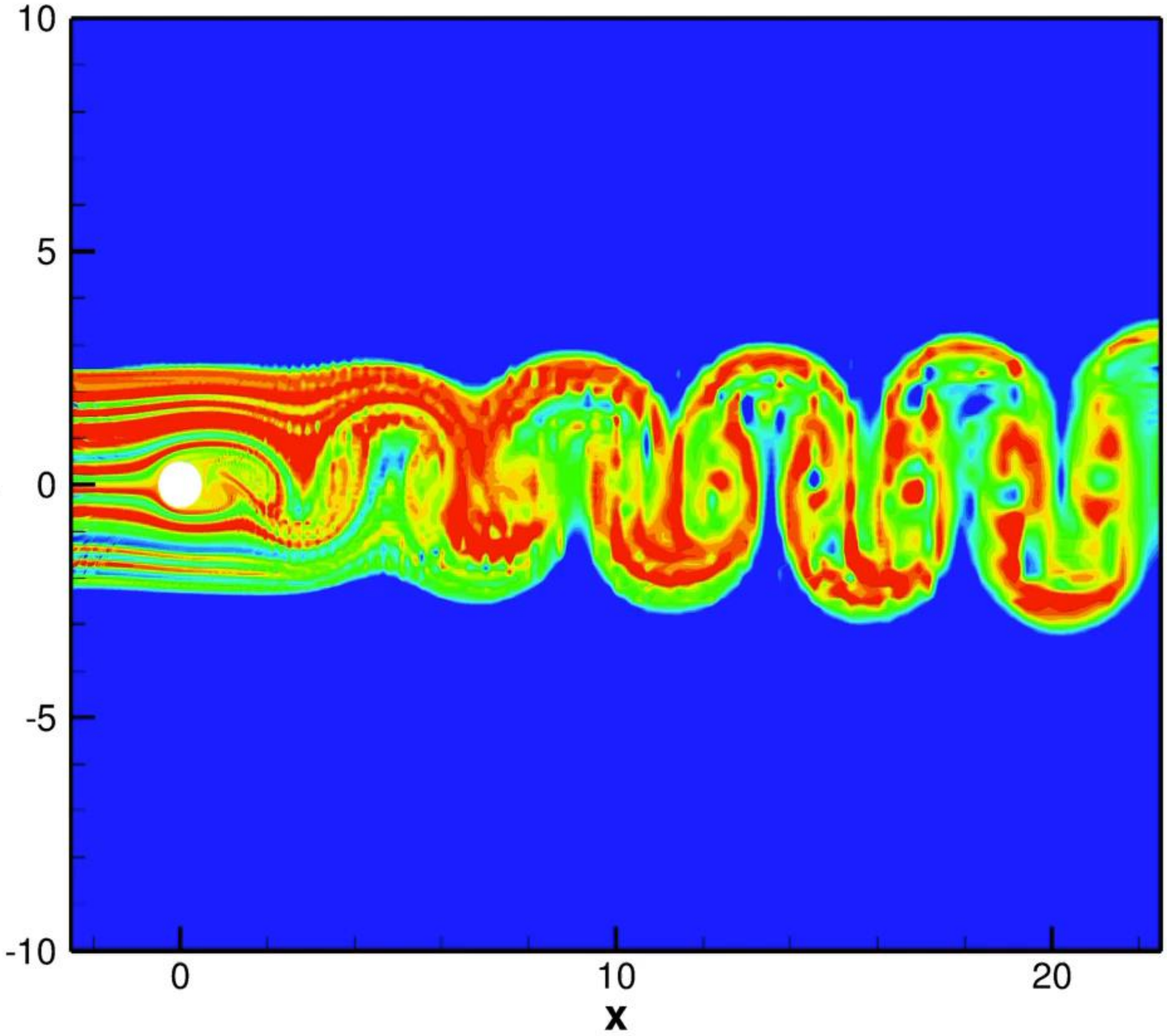
Aim: Extract quantitative information from videos for rescue operations and beyond!



Hidden Fluid Mechanics

Can we compute the body forces based on flow viz only?

Science
AAAS



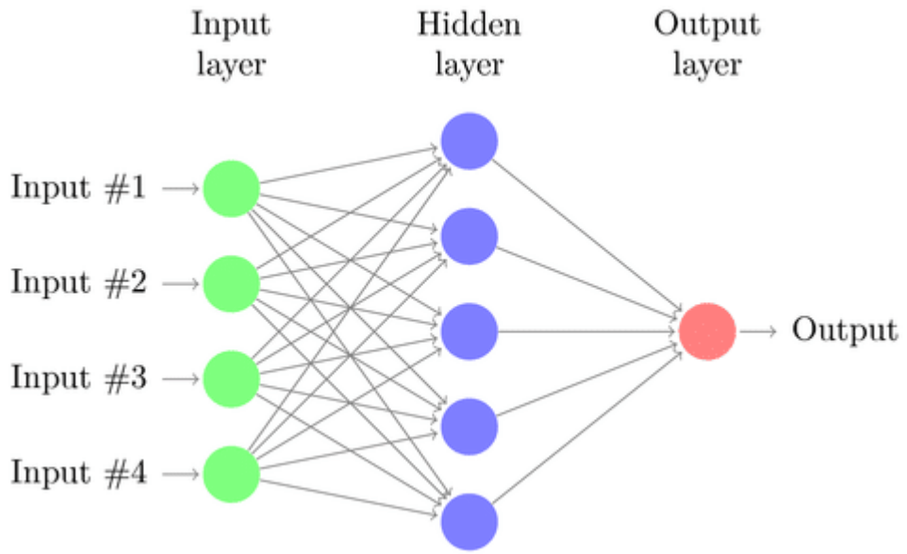
Neural Networks are Function Approximators, Not Predictors!

Theorem (*Cybenko, 1989*)

Let σ be any continuous sigmoidal function. Then, the finite sums of the form

$$G(x) = \sum_{j=1}^{N} \alpha_j \sigma\left(y_j \cdot x + \theta_j\right)$$

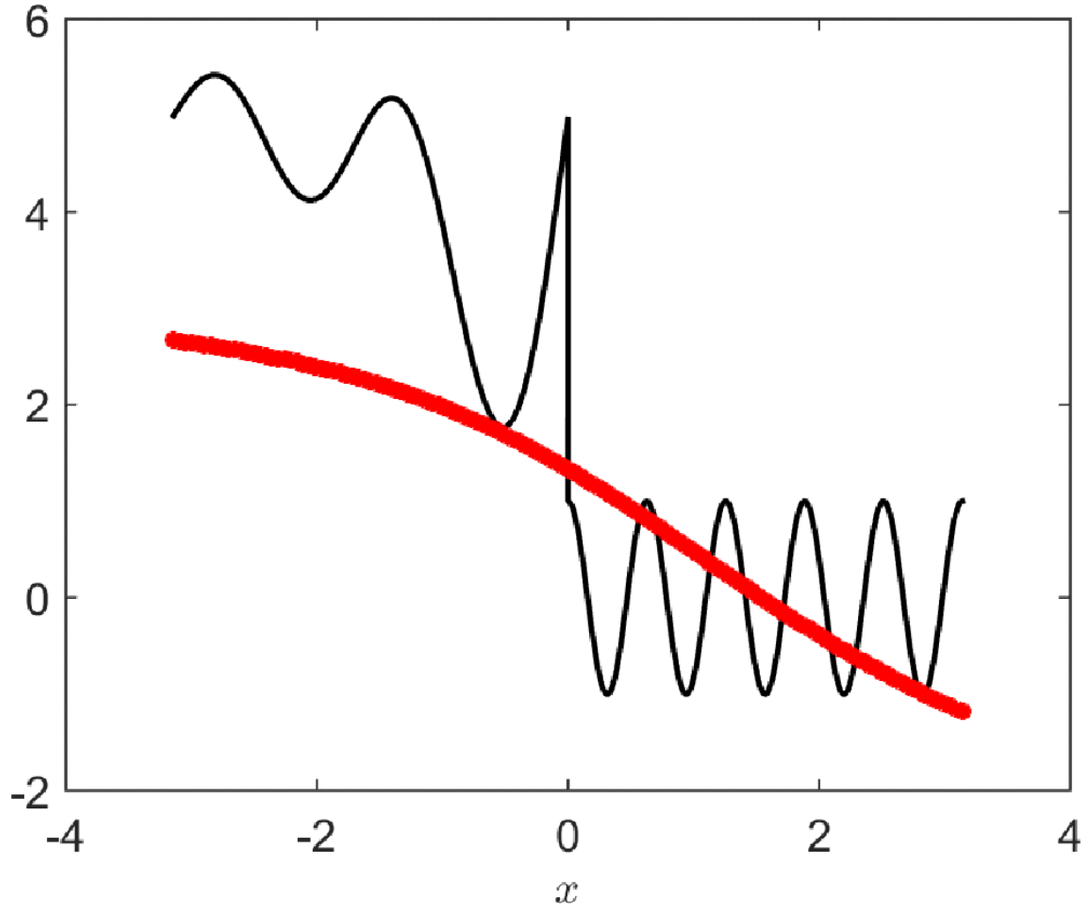
are dense in $C(I_d)$.

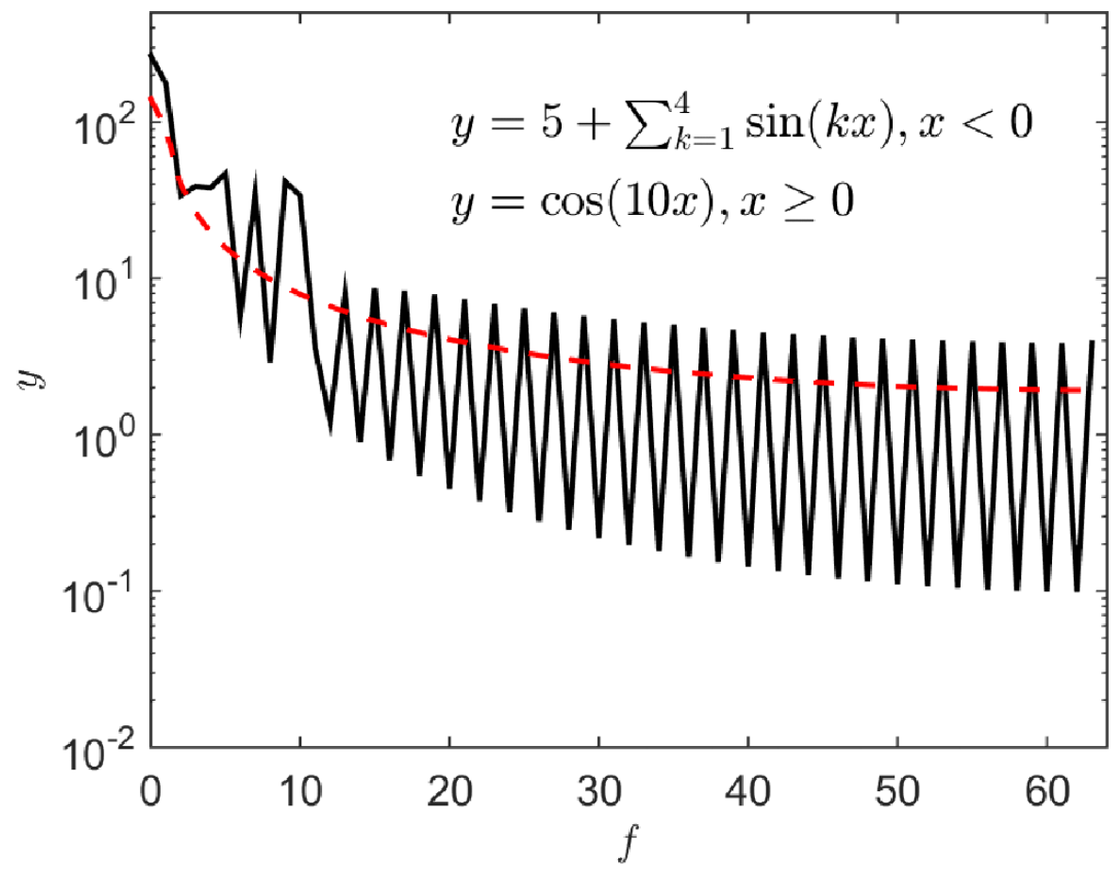


- G. Cybenko, "Approximation by superpositions of a sigmoidal function", Mathematics of Control, Signals and Systems, 303-314, 2(4), 1989

- Hornik et. al., 1989; Barron (1994); Mhaskar (1996)

Learning a Discontinuous/Oscillatory Function in Physical & Fourier Domains

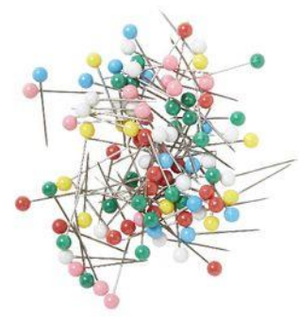




DNN: Fully Connected 2 x 20

Data + Neural Networks + Physical Laws

*Physics-Informed Neural Networks (PINNs)



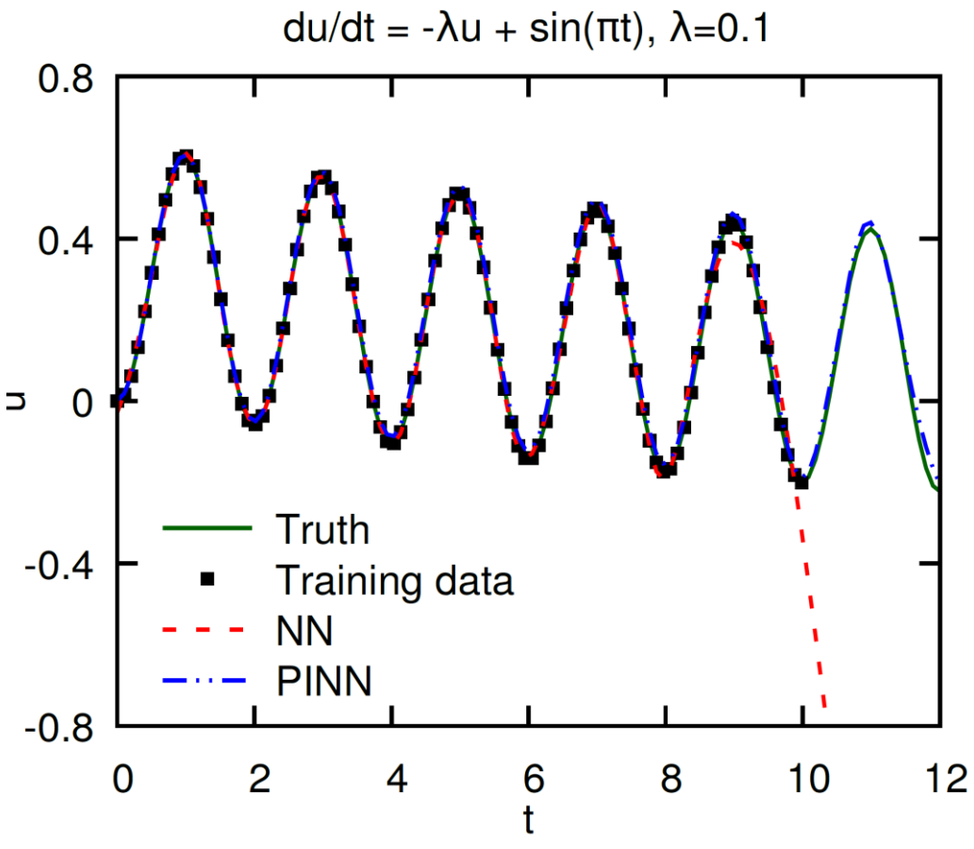
Physics-informed neural networks: A deep learning framework for solving forward and inverse problems involving nonlinear partial differential equations

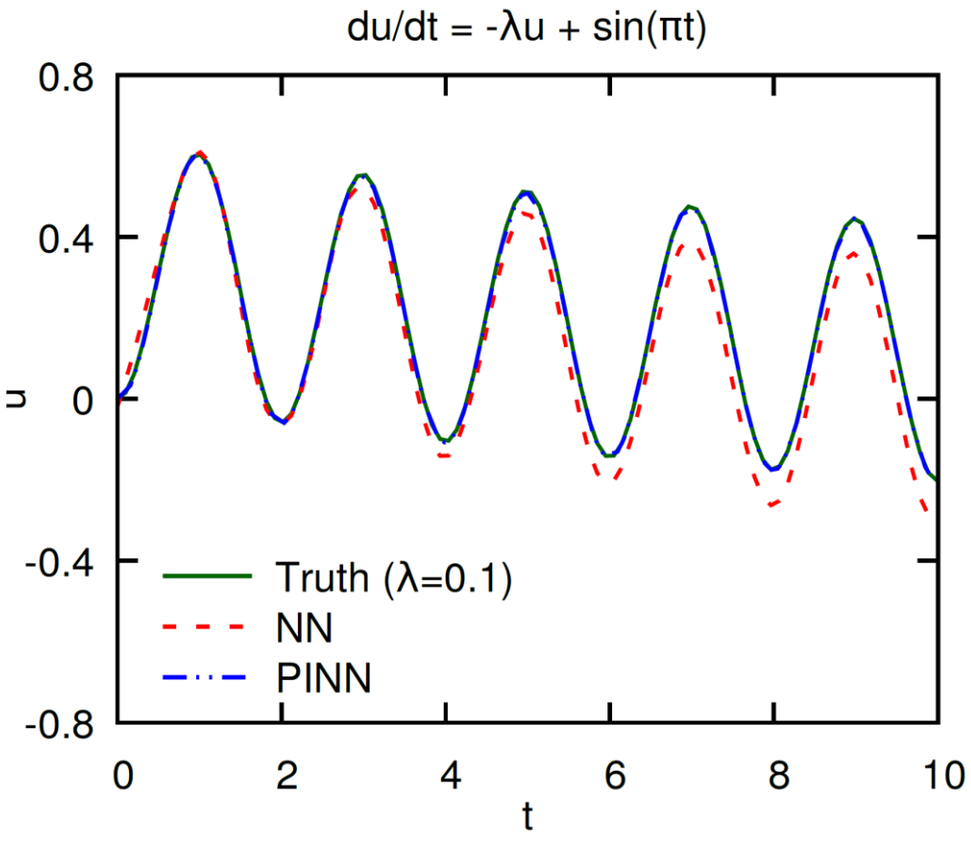
M Raissi, P Perdikaris, GE Karniadakis
Journal of Computational Physics 378, 686-707

arXiv:1711.10561; arXiv:1711.10566 (2017)

Physics-Informed Neural Networks (PINN) vs Neural Networks (NN)

❖ to PINN or not to PINN?



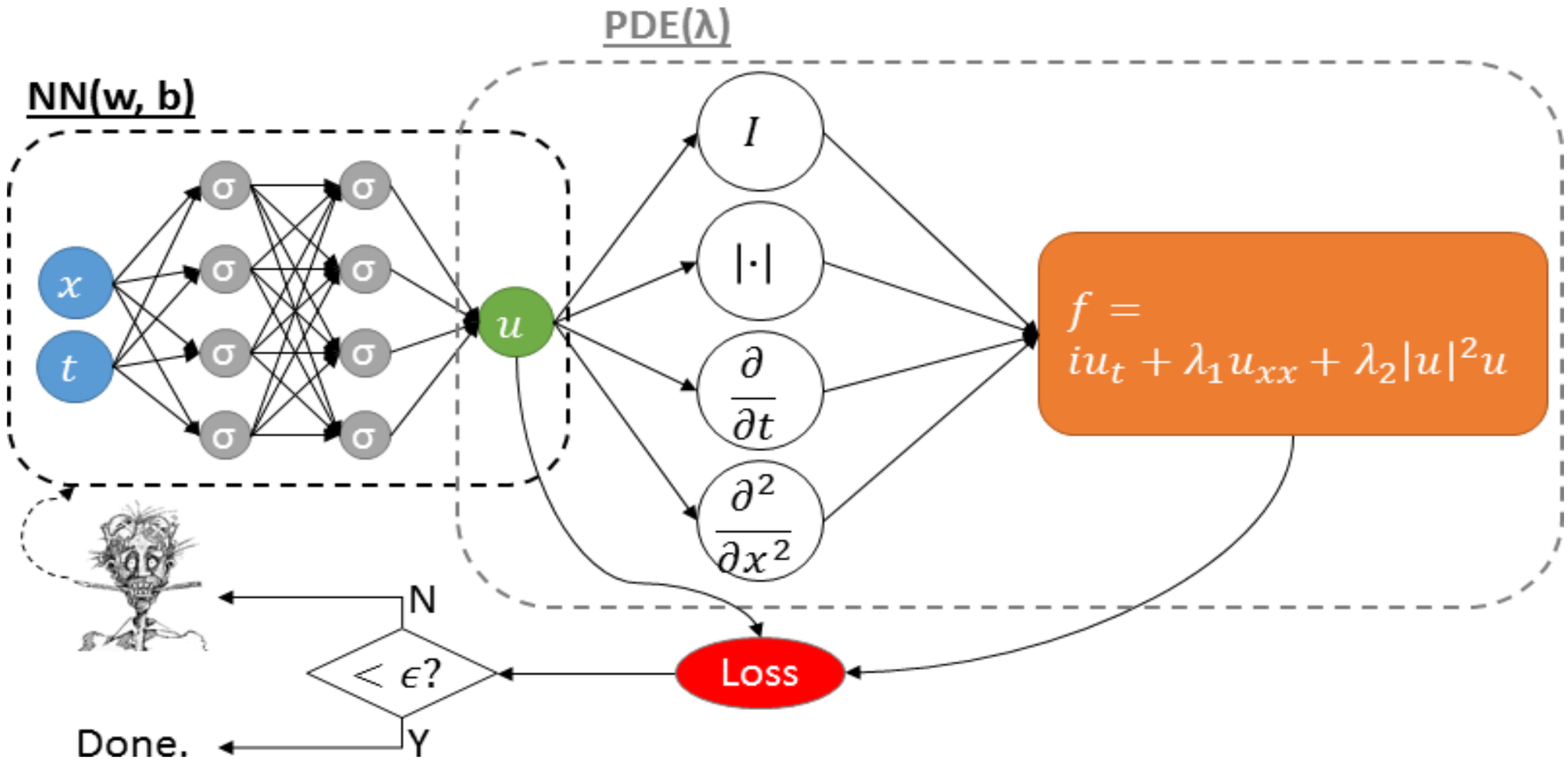


- train for lambda = 0.3, 0.4, ..., 1 and t = [0, 10]

- NN: 4 layers, 20 neurons per layer
- Adam optimizer, learning rate 0.001, optimized for 10000 steps.

What is a PINN? Physics-Informed Neural Network

We employ two (or more) NNs that share the same parameters



- Minimize:

$$MSE = MSE_u + MSE_f,$$

L-BFGS

$$MSE_u = \frac{1}{N_u}\sum_{i=1}^{N_u} |u(t_u^i, x_u^i) - u^i|^2,$$

$$MSE_f = \frac{1}{N_f}\sum_{i=1}^{N_f} |f(t_f^i, x_f^i)|^2.$$

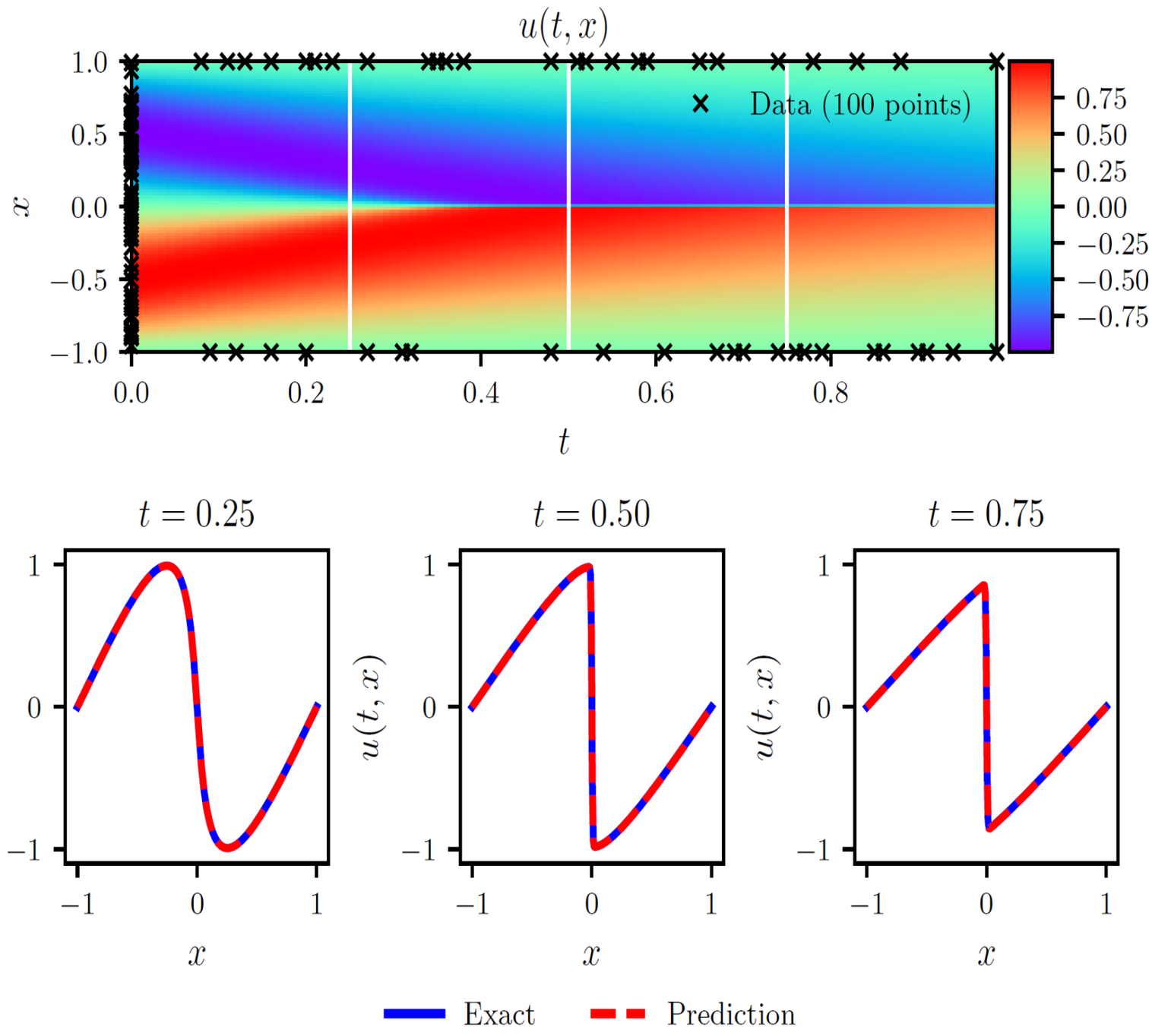
Physics Informed Neural Networks

$$u_t + uu_x - (0.01/\pi)u_{xx} = 0$$

```
def u(t, x):
    u = neural_net(tf.concat([t,x],1), weights, biases)
    return u
```

```
def f(t, x):
    u = u(t, x)
    u_t = tf.gradients(u, t)[0]
    u_x = tf.gradients(u, x)[0]
    u_xx = tf.gradients(u_x, x)[0]
    f = u_t + u*u_x - (0.01/tf.pi)*u_xx
    return f
```

$$\sum_{i=1}^{N_u} |u(t_u^i, x_u^i) - u^i|^2 + \sum_{i=1}^{N_f} |f(t_f^i, x_f^i)|^2$$



PINNs Change the Learning Mode of DNNs

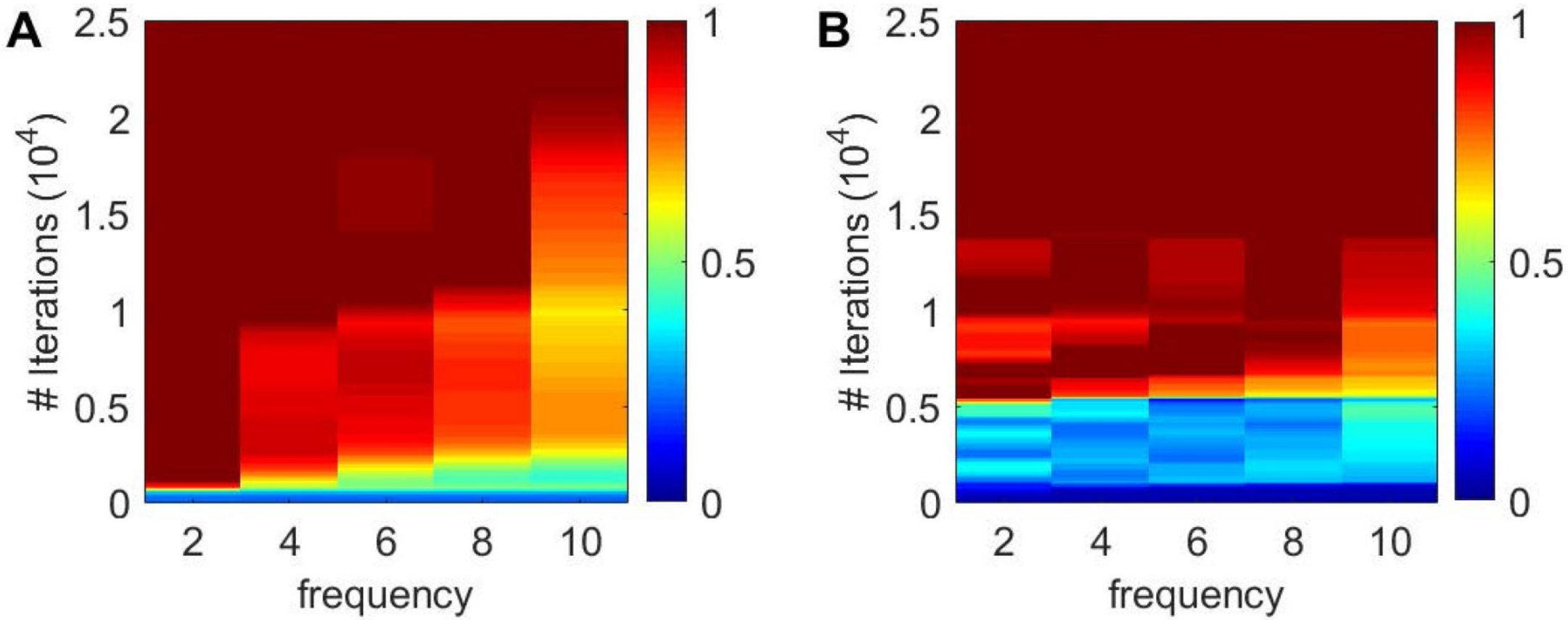


FIG. 2. *Convergence of the amplitude for each frequency during the training process.* (***A***) *A neural network is trained to approximate the function* $f(x) = \sum_{k=1}^{5} \sin(2kx)/(2k)$. *The color represents amplitude values with the maximum amplitude for each frequency normalized to 1.* (***B***) *A PINN is used to solve the Poisson equation* $-f_{xx} = \sum_{k=1}^{5} 2k \sin(2kx)$ *with zero boundary conditions. We use a neural network of 4 hidden layers and 20 neurons per layer. The learning rate is chosen as* 10^{-4}, *and 500 random points are sampled for training.*

PINN Approximation Theorem

A. Pinkus, Acta Numerica, (1999)

we set $|\mathbf{m}| := m_1 + \cdots + m_d$, and

$$D^{\mathbf{m}} := \frac{\partial^{|\mathbf{m}|}}{\partial x_1^{m_1} \dots \partial x_d^{m_d}}.$$

THEOREM 2.1. *Let $\mathbf{m}^i \in \mathbb{Z}_+^d$, $i = 1, \dots, s$, and set $m = \max_{i=1,\dots,s} |\mathbf{m}^i|$. Assume $\sigma \in C^m(\mathbb{R})$ and σ is not a polynomial. Then the space of single hidden layer neural nets*

$$\mathcal{M}(\sigma) := span\{\sigma(\mathbf{w} \cdot \mathbf{x} + b) : \mathbf{w} \in \mathbb{R}^d, b \in \mathbb{R}\}$$

is dense in

$$C^{\mathbf{m}^1,\dots,\mathbf{m}^s}(\mathbb{R}^d) := \cap_{i=1}^s C^{\mathbf{m}^i}(\mathbb{R}^d),$$

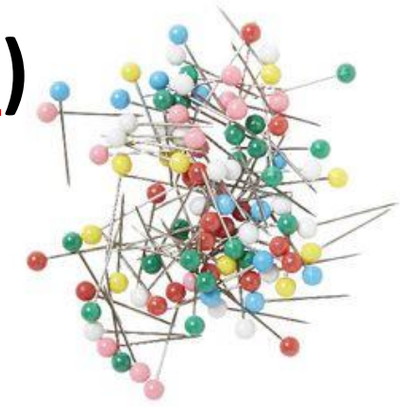
i.e., for any $f \in C^{\mathbf{m}^1,\dots,\mathbf{m}^s}(\mathbb{R}^d)$, any compact $K \subset \mathbb{R}^d$, and any $\varepsilon > 0$, there exists a $g \in \mathcal{M}(\sigma)$ satisfying

$$\max_{\mathbf{x} \in K} |D^{\mathbf{k}} f(\mathbf{x}) - D^{\mathbf{k}} g(\mathbf{x})| < \varepsilon,$$

for all $\mathbf{k} \in \mathbb{Z}_+^d$ for which $\mathbf{k} \leq \mathbf{m}^i$ for some i.

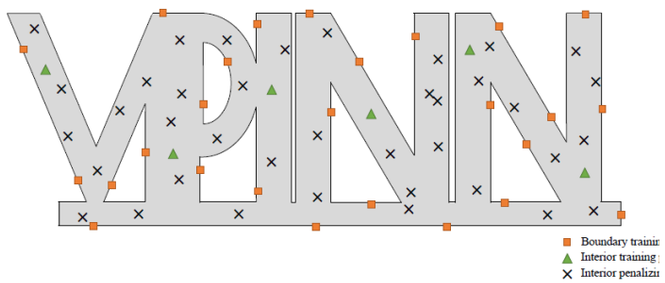
Theorem 2.1 shows that feed-forward neural nets with enough neurons can simultaneously and uniformly approximate any function and its partial derivatives.

Physics-Informed Neural Networks (PINNs)



- cPINNs: conservative PINNs
- vPINNs: variational PINNs
- pPINNs: parareal PINNs
- sPINNs: stochastic PINNs
- fPINNs: fractional PINNs
- LePINNs: Levy process PINNs
- nPINNs: Nonlocal PINNs…

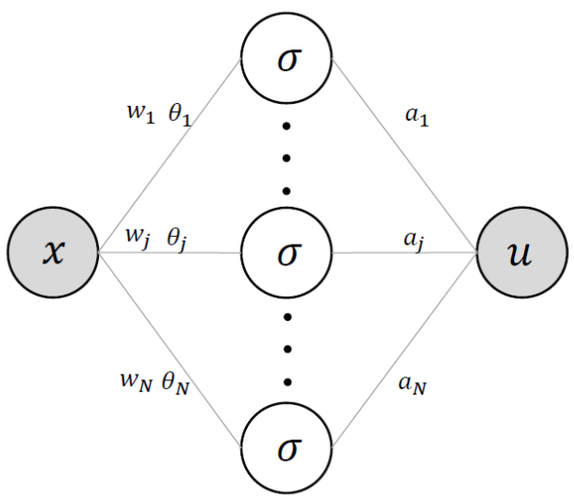
Variational PINNs



$$\mathcal{R}_k^{(1)} = -\left(\frac{d^2 u_{NN}(x)}{dx^2}, v_k(x)\right)_\Omega,$$

$$\mathcal{R}_k^{(2)} = \left(\frac{du_{NN}(x)}{dx}, \frac{dv_k(x)}{dx}\right)_\Omega - \frac{du_{NN}(x)}{dx} v_k(x)\Big|_{\partial\Omega},$$

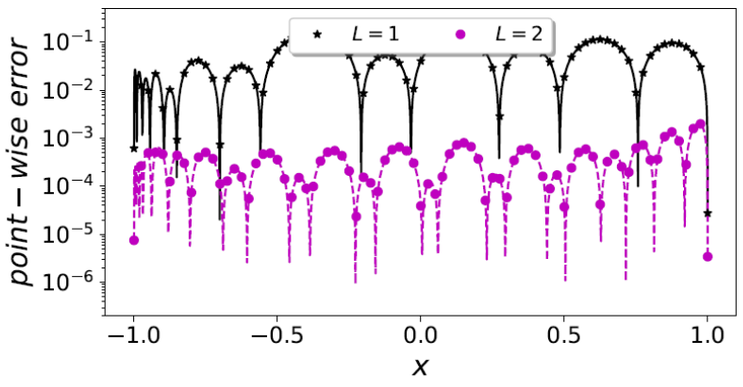
$$\mathcal{R}_k^{(3)} = -\left(u_{NN}(x), \frac{d^2 v_k(x)}{dx^2}\right)_\Omega - \frac{du_{NN}(x)}{dx} v_k(x)\Big|_{\partial\Omega} + u_{NN}(x)\frac{dv_k(x)}{dx}\Big|_{\partial\Omega}.$$



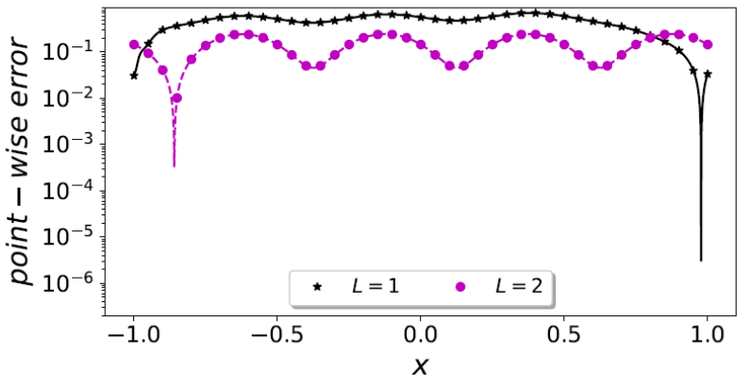
$$\mathcal{R}_k^{(1)} = \mathcal{R}_k^{(2)} = 2(-1)^k k\pi \sum_{j=1}^{N} \frac{a_j w_j^2 \cos(\theta_j)\sin(w_j)}{w_j^2 - k^2\pi^2},$$

$$\mathcal{R}_k^{(3)} = 2(-1)^k k\pi \sum_{j=1}^{N} \frac{a_j k^2 \pi^2 \cos(\theta_j)\sin(w_j)}{w_j^2 - k^2\pi^2} + (-1)^k k\pi (h-g).$$

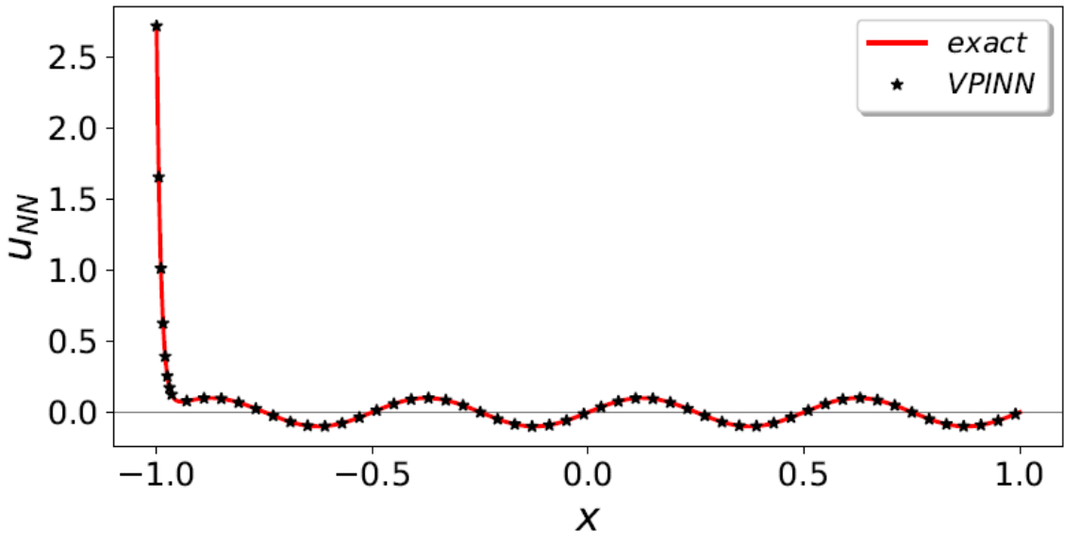
VPINN

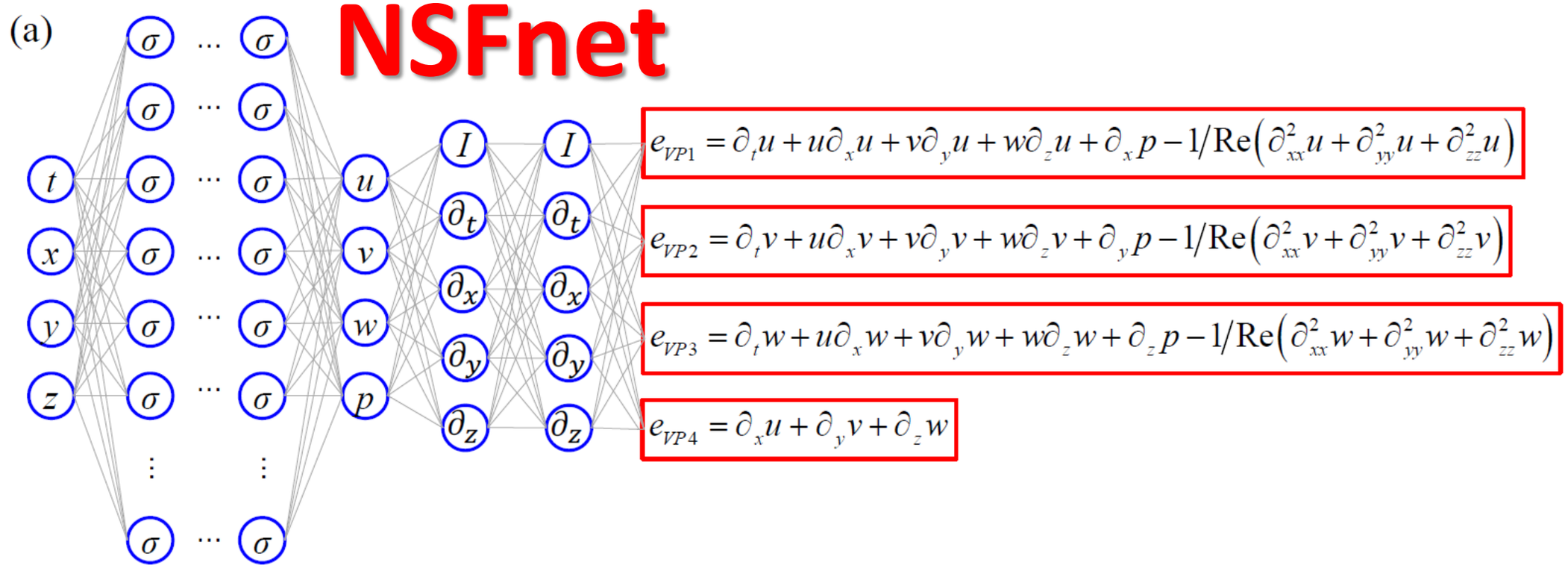


PINN



$$u^{exact} = 0.1\sin(4\pi x) + e^{\frac{0.01-(x+1)}{0.01}}$$





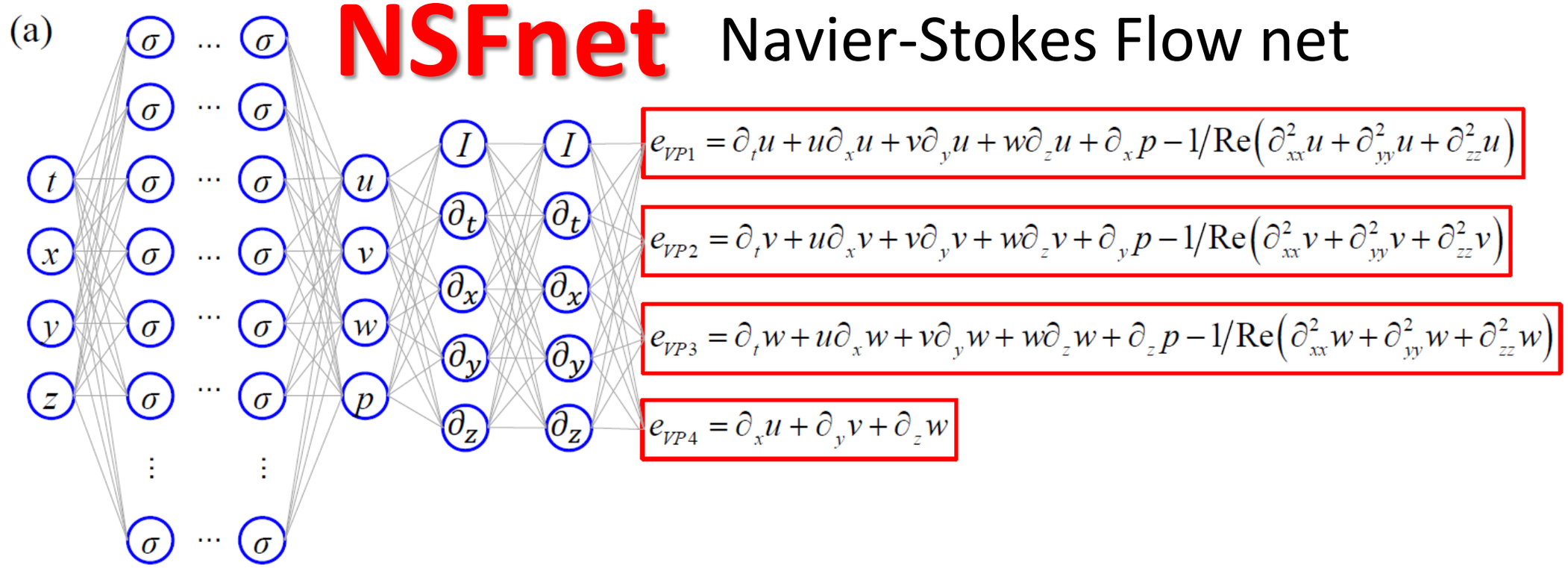
The loss function for training the parameters of PINNs to obtain the solutions of equations (1) is defined as follows:

$$L = L_e + \alpha L_b + \beta L_i, \tag{2a}$$

$$L_e = \frac{1}{N_e}\sum_{i=1}^{4}\sum_{n=1}^{N_e}|e_{VPi}|^2, \tag{2b}$$

$$L_b = \frac{1}{N_b}\sum_{n=1}^{N_b}\left(|u-u^b|^2 + |v-v^b|^2 + |w-w^b|^2\right), \tag{2c}$$

$$L_i = \frac{1}{N_i}\sum_{n=1}^{N_i}\left(|u-u^i|^2 + |v-v^i|^2 + |w-w^i|^2\right), \tag{2d}$$



The loss function for training the parameters of PINNs to obtain the solutions of equations (1) is defined as follows:

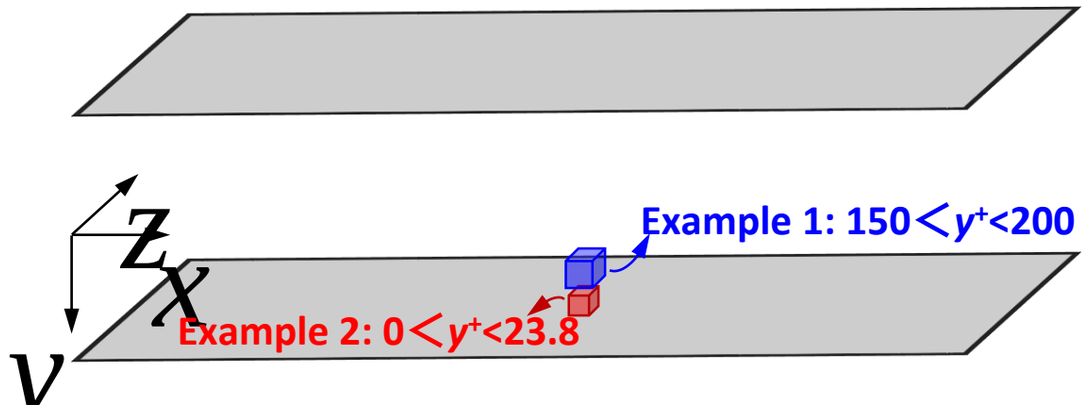
$$L = L_e + \alpha L_b + \beta L_i, \tag{2a}$$

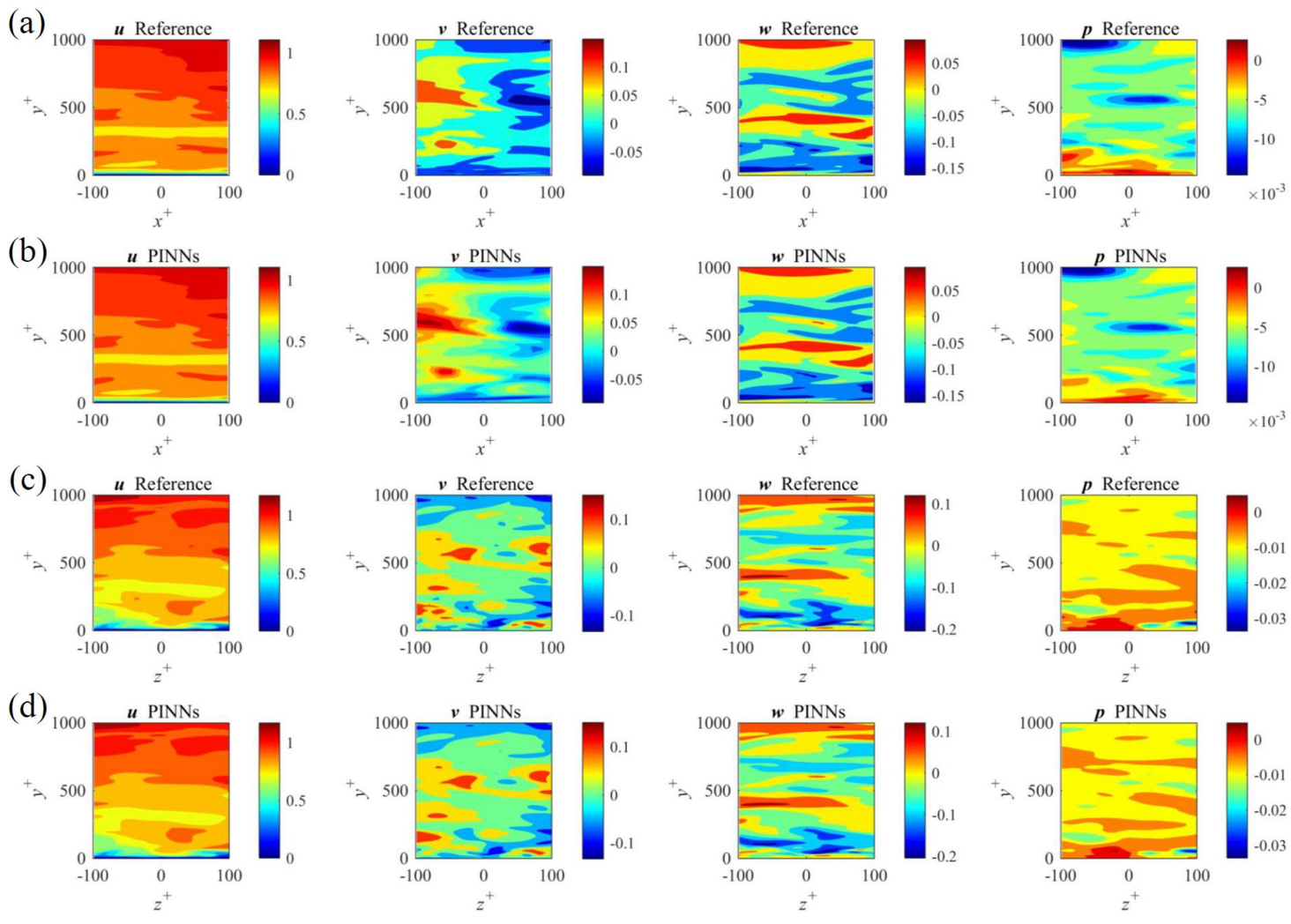
$$L_e = \frac{1}{N_e}\sum_{i=1}^{4}\sum_{n=1}^{N_e}|e_{VPi}|^2, \tag{2b}$$

$$L_b = \frac{1}{N_b}\sum_{n=1}^{N_b}\left(|u-u^b|^2 + |v-v^b|^2 + |w-w^b|^2\right), \tag{2c}$$

$$L_i = \frac{1}{N_i}\sum_{n=1}^{N_i}\left(|u-u^i|^2 + |v-v^i|^2 + |w-w^i|^2\right), \tag{2d}$$

Can PINNs Sustain Turbulence?





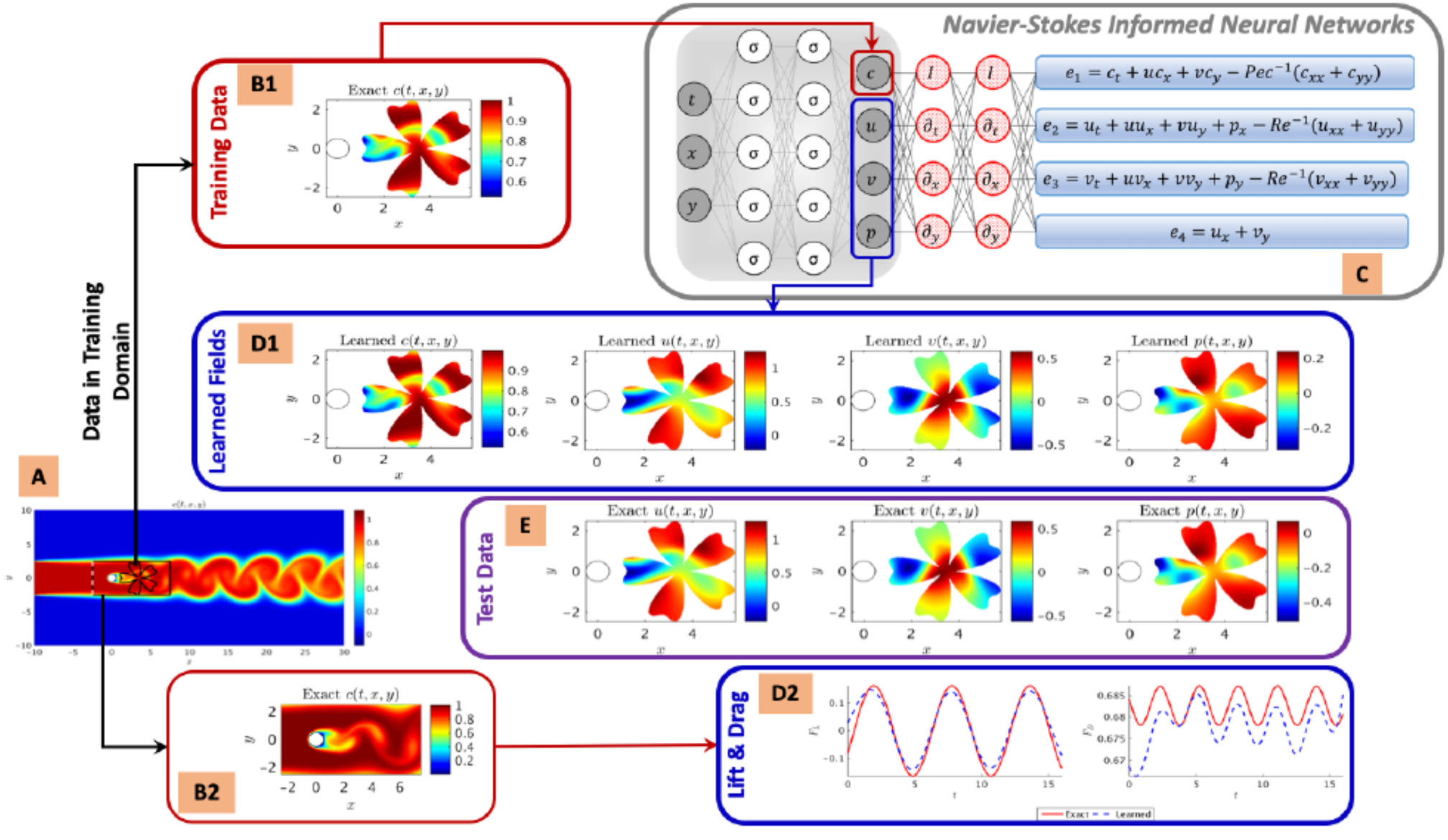
Location: $[12.47,12.66] \times [-1.00, -0.0031] \times [4.61,4.82]$

Ne = 100 000 per time step; Nb = 26 048 and Ni = 147 968

Hidden Fluid Mechanics

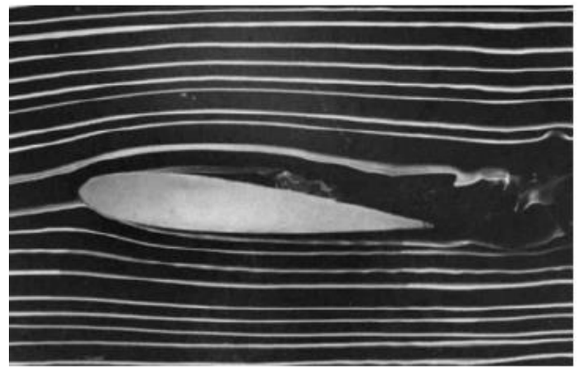
M. Raissi, A. Yazdani and G.E. Karniadakis, arXiv:1808.04327

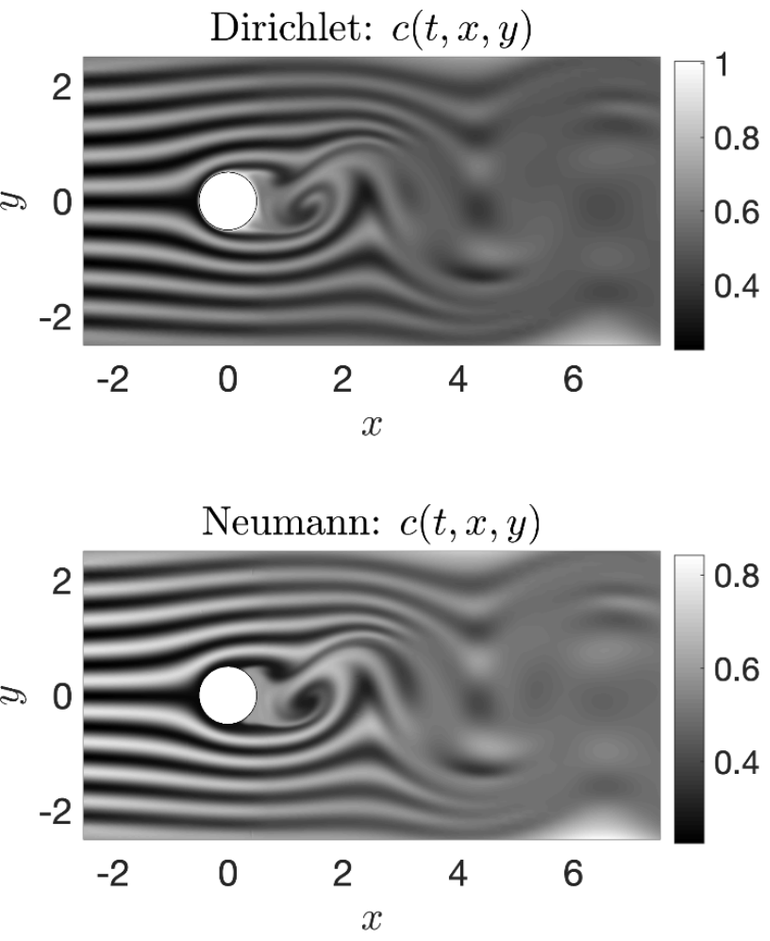
Science
AAAS

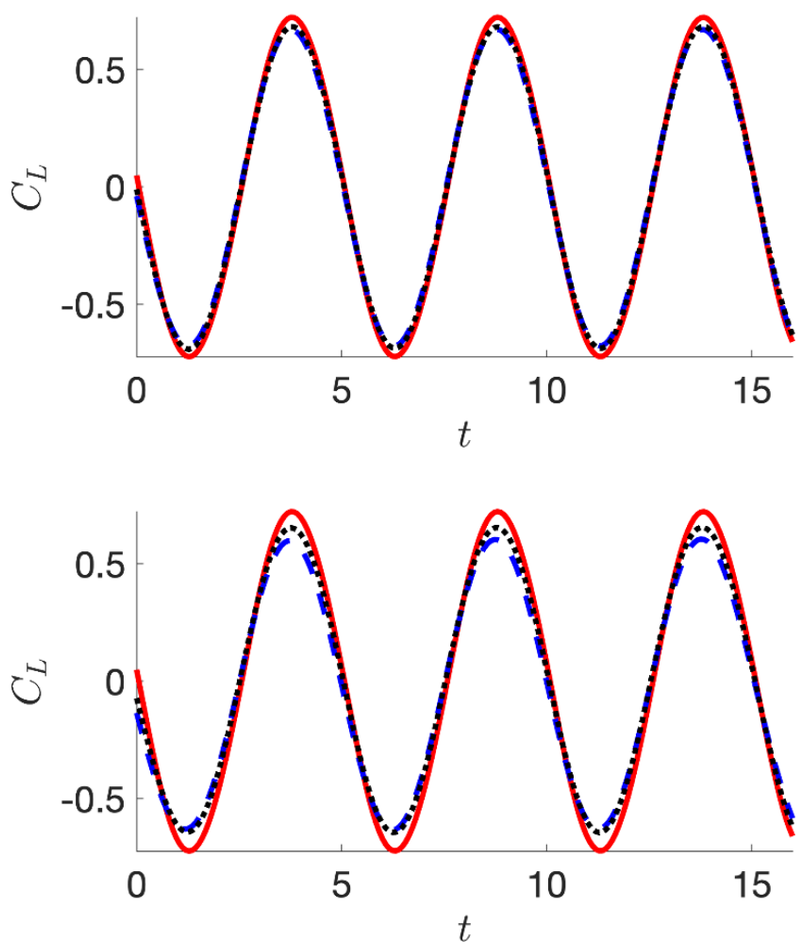


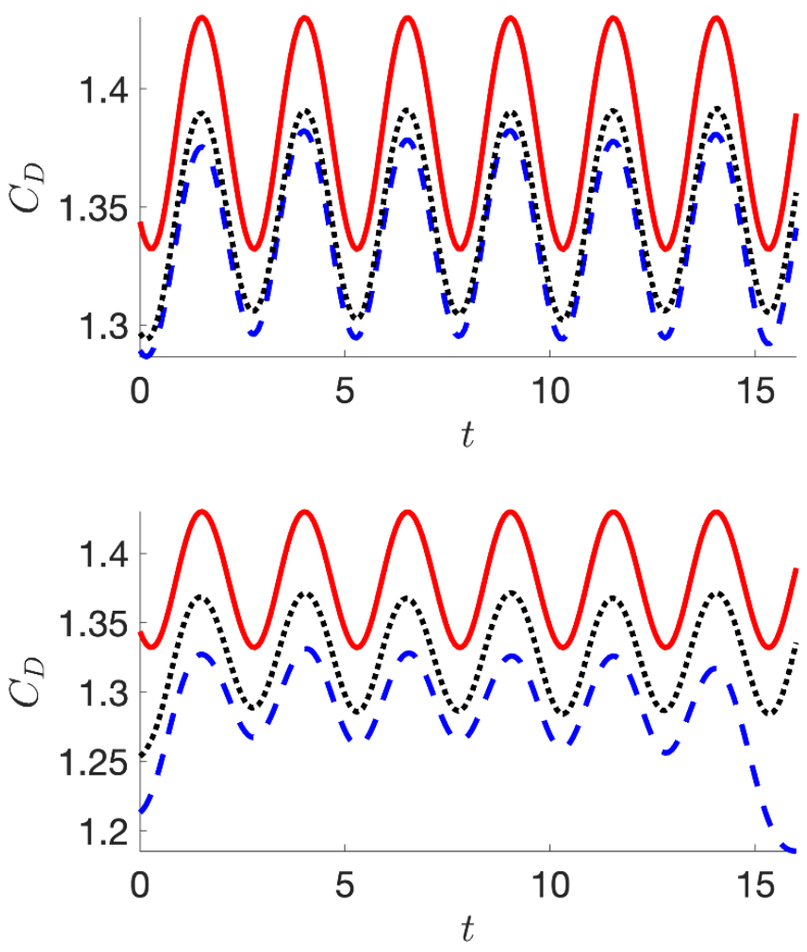
Science
AAAS

Smoke-type Visualizations







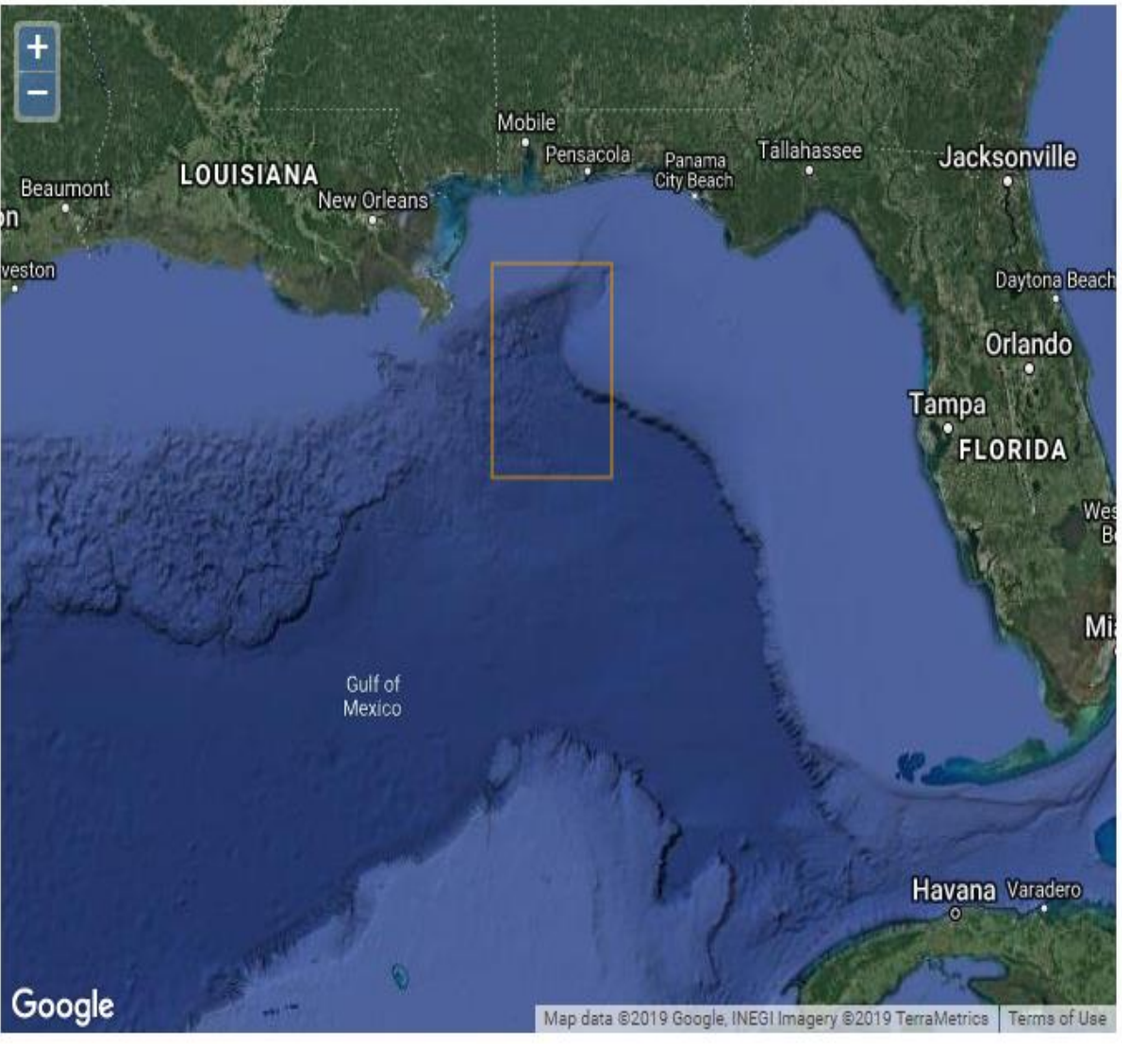


Reference — Regressed (Without No-slip B.C.) - - Regressed (With No-slip B.C.) ····

Re=200; Pe=2,000

Gulf of Mexico Sea Surface Temperature

- Regional Ocean Modeling System (ROMS) simulation in the LASER region during a winter month, 150 m horizontal resolution (from GRIIDC provided by Roy Barkan)



Regional Ocean Modeling System (ROMS) simulation in the LASER region during a winter month, 150 m horizontal resolution

Consortium for Advanced Research on Transport of Hydrocarbon in the Environment II (CARTHE II)

DOI:
10.7266/N75H7DQ5

UDI:
R4.x265.000:0037

Last Update:
Jul 08 2017 00:04 UTC

Dataset Author(s):
Roy Barkan

Funding Source:
RFP-IV

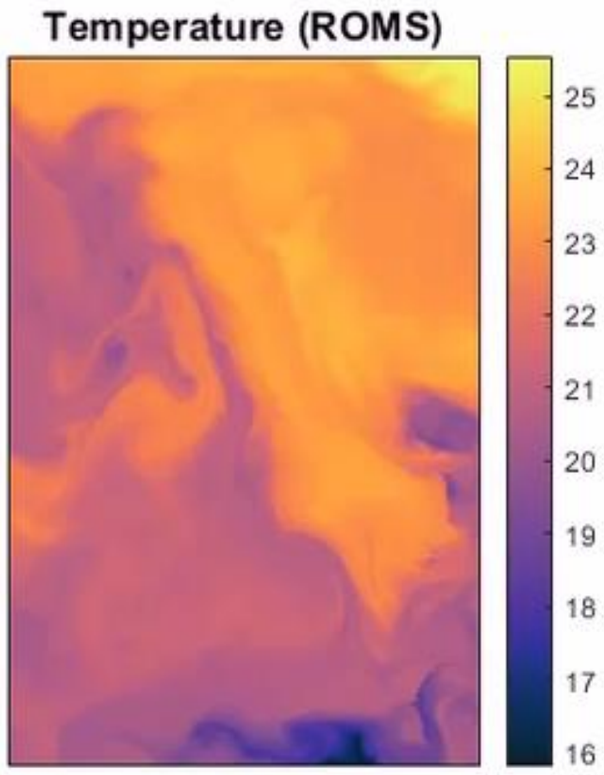
Data Collection Period:
2010-02-01 to 2010-02-19

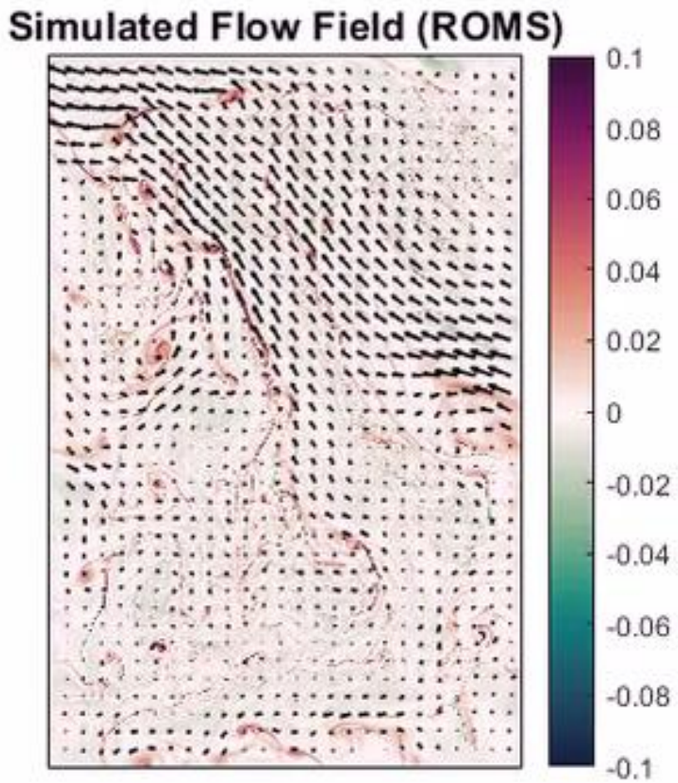
Point of Contact:
Barkan, Roy
University of California Los Angeles / Department of Atmospheric and Oceanic Sciences
Box 957228
Los Angeles, California 90095
USA
rbarkan@atmos.ucla.edu

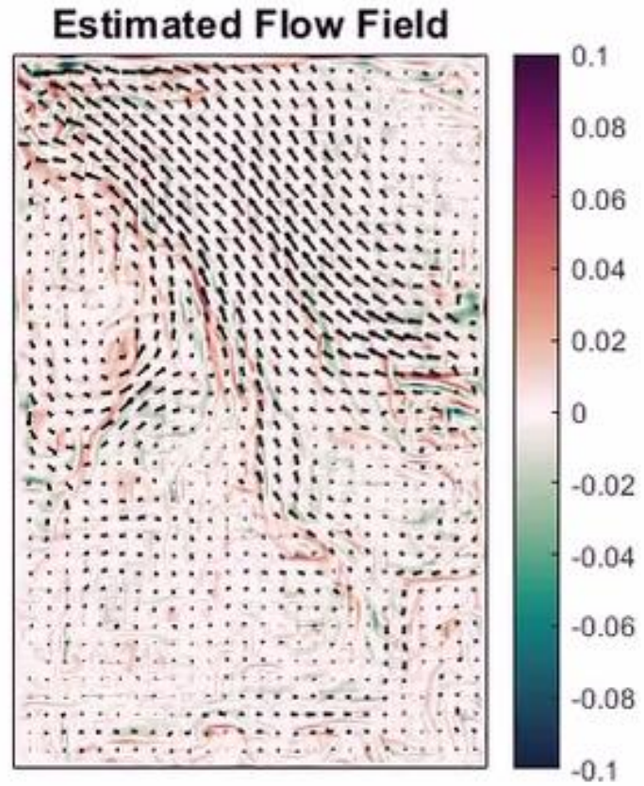
Identified	Submitted	In-Review	Available
✓	✓	✓	✓

PINNs Estimate Velocity from SS Temperature

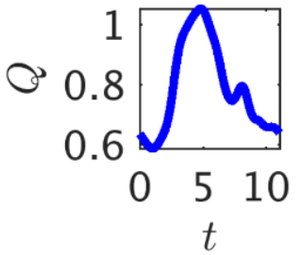
- Velocity field and vorticity map (February)
- Movie (ten hours per frame)

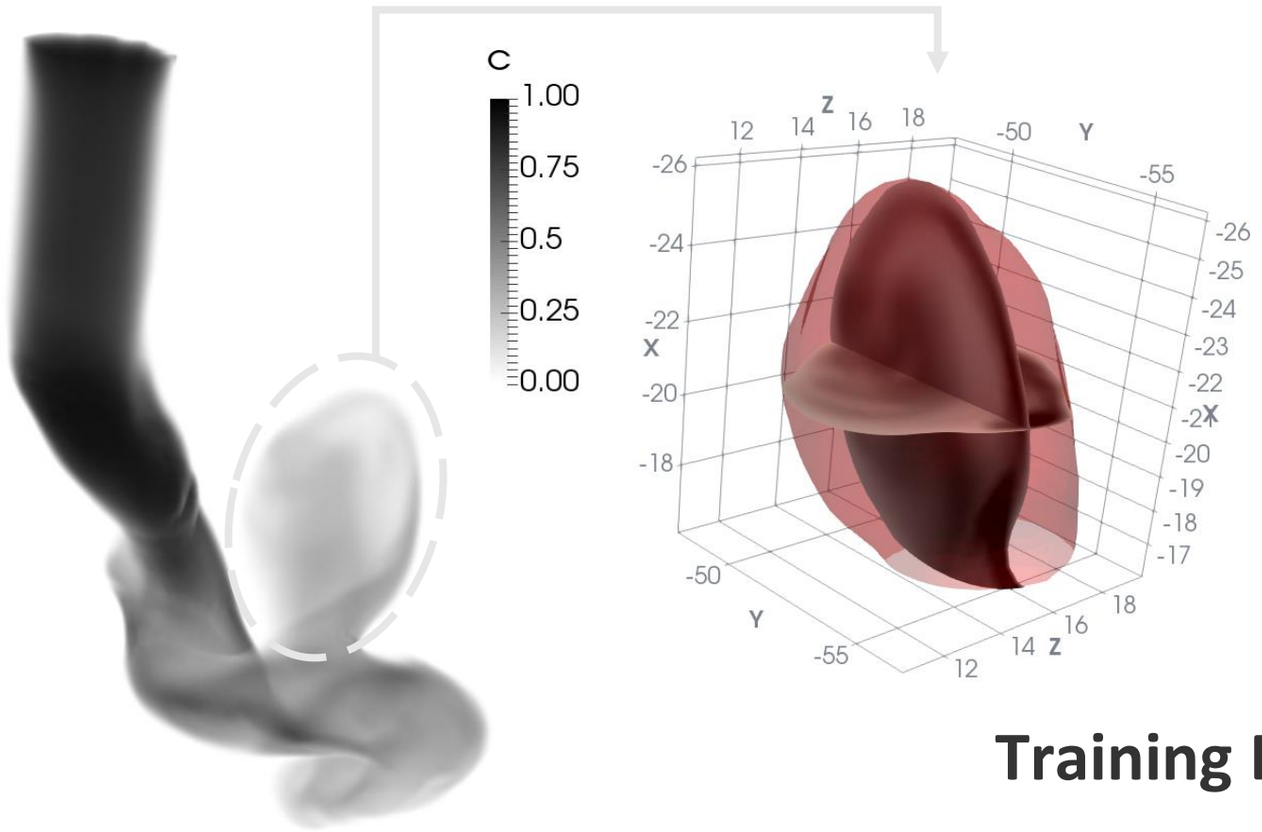






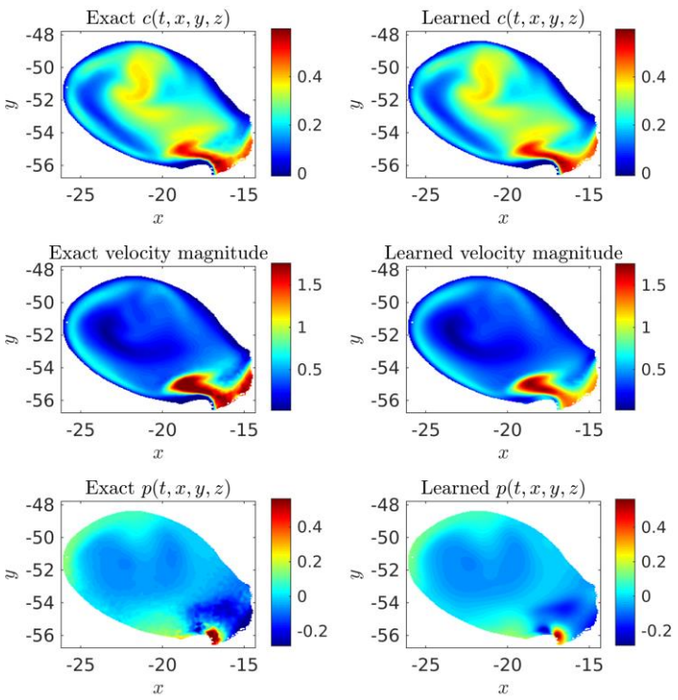
Hidden Fluid Mechanics:
flow dynamics in an intracranial aneurysm

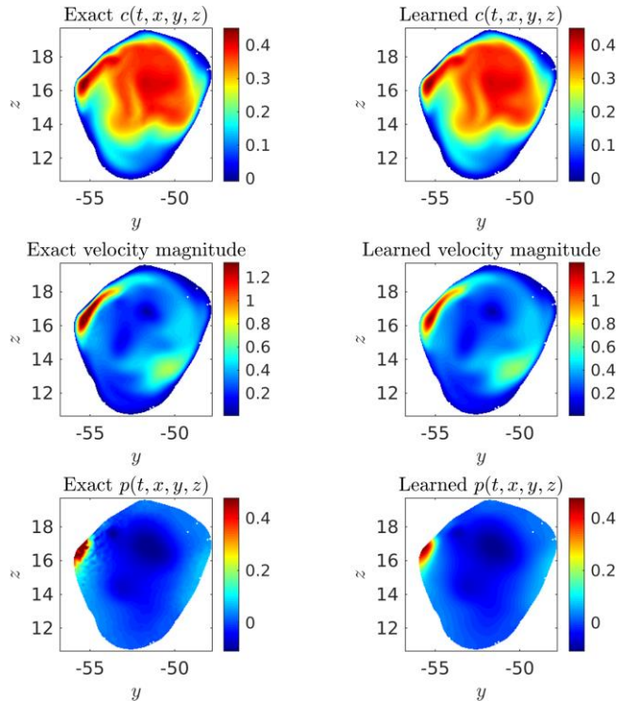


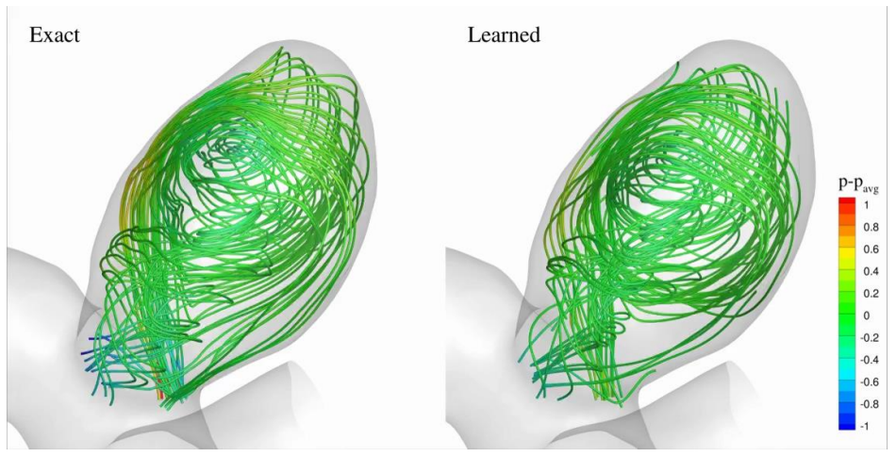


Training Data

Inferred Hidden Physics







Heat Transfer: Hidden Thermal Boundary Conditions

- We can simulate the flow but not HT because of lack of thermal BC.
- In applications the thermal BC is neither constant T nor constant flux.
- We assume that we have some thermocouple measurements available.
- This problem is ill-posed with classical methods.

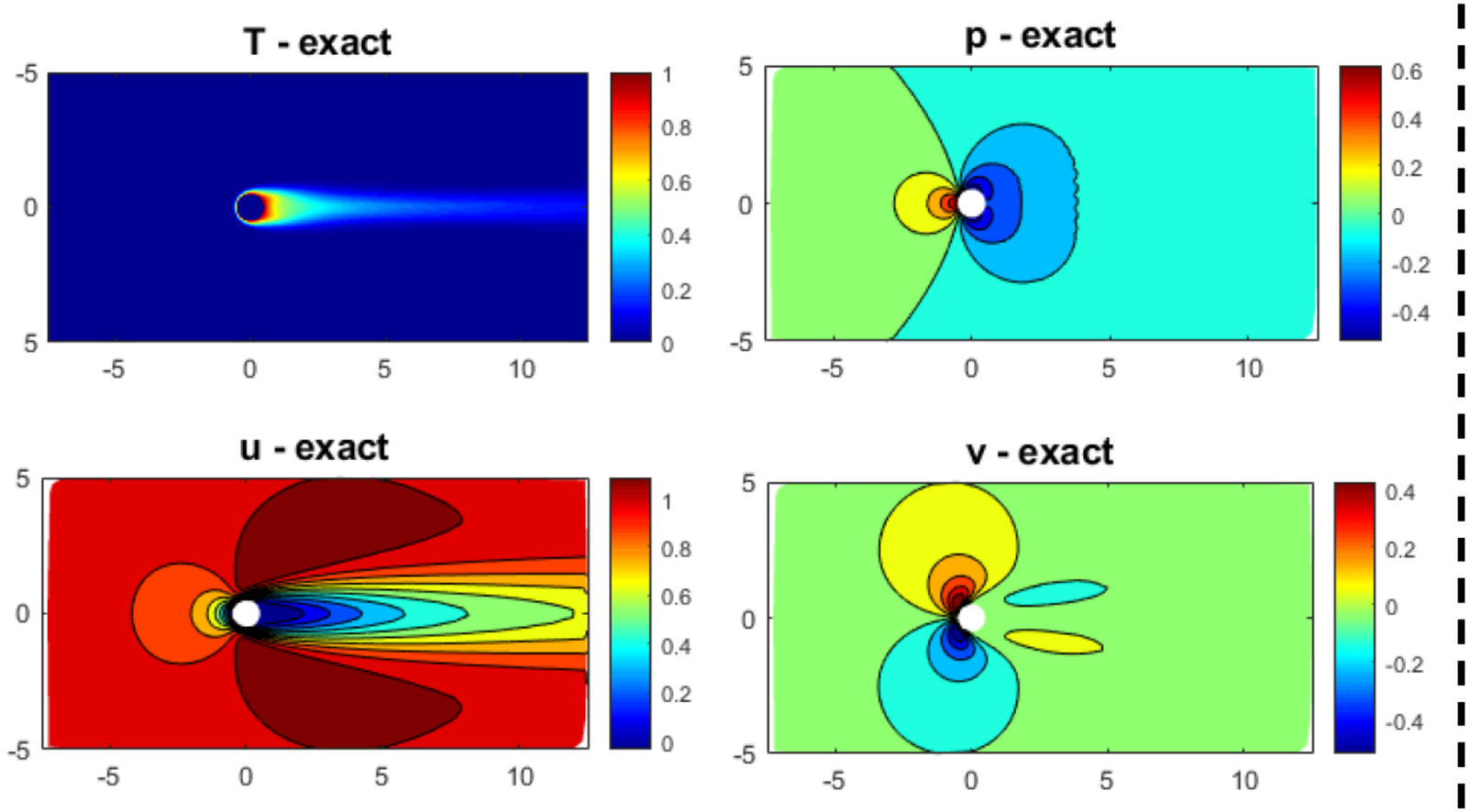
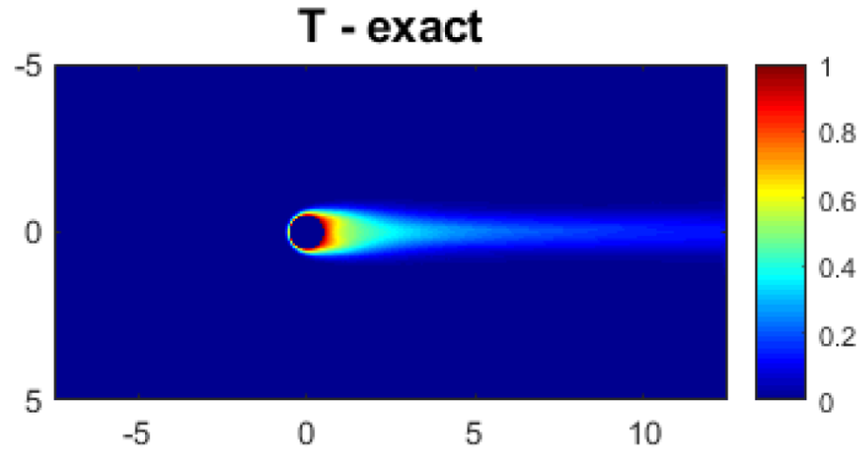


Figure: Flow and heat past a circular cylinder (NekTar).

PINN Formulation for Convective Heat Transfer

Experimental setup



- Hidden Fluid Mechanics (NN)
- 10 hidden layers and 200 neurons per hidden layer
- Loss function

$$\text{Loss} = \sum_i \|e_i\| + \sum_i [(u_{NN} - u_b)^2 + (v_{NN} - v_b)^2] + \sum_i (T_{NN} - T_b)^2 + \sum_i (T_{NN} - T)^2$$

(N_e, N_{b1}, N_{b2}, N_T)

Physical constraints (NS equations and heat transfer)
$(N_e \approx 25000)$

Boundary conditions for velocity field, including the no-slip boundary condition of the cylinder
$(N_{b1} \approx 900)$

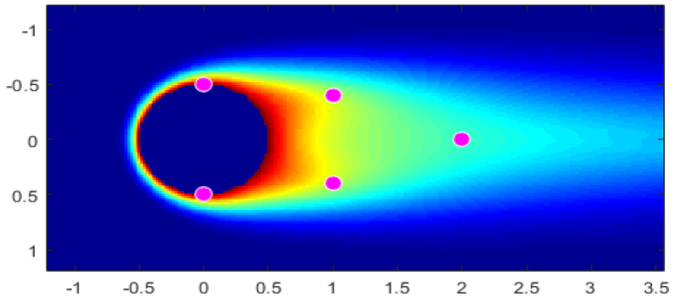
Boundary conditions for temperature
$(N_{b2} \approx 520)$

Temperature measurements in the domain
$(N_T < 10)$

Discover constant Temperature on Cylinder

Experimental results

- Case A



L2 error - u: 0.45 %
L2 error - v: 0.63 %
L2 error - p: 1.10 %

L2 error - c: 19.45 %

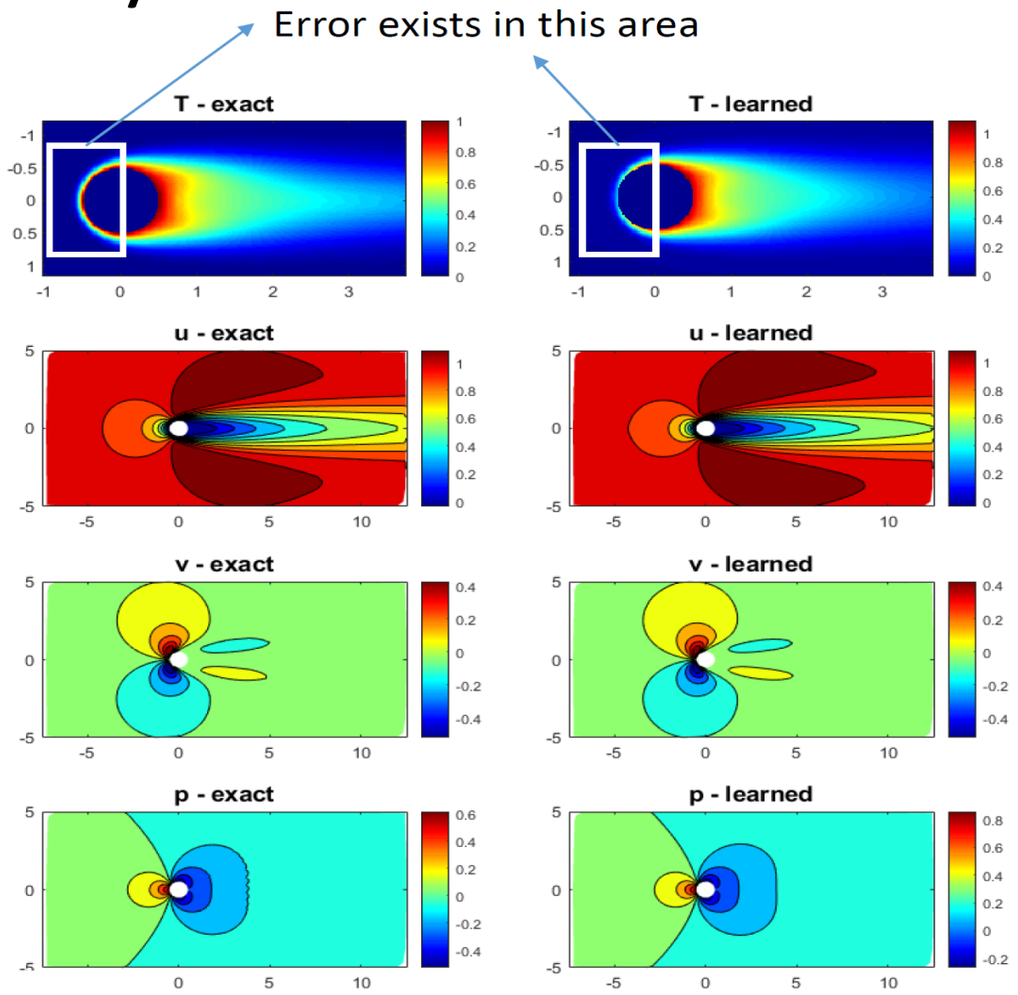
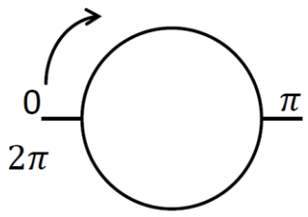
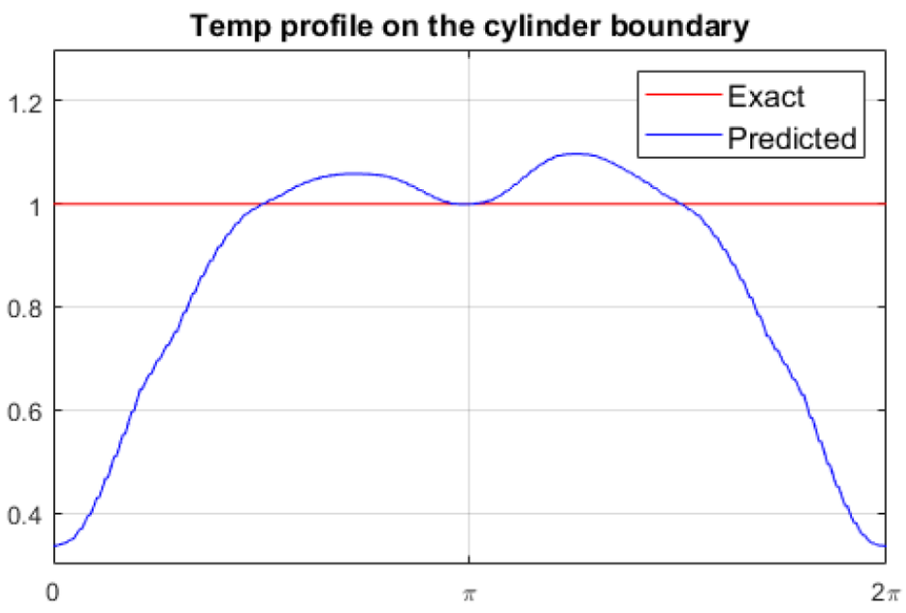
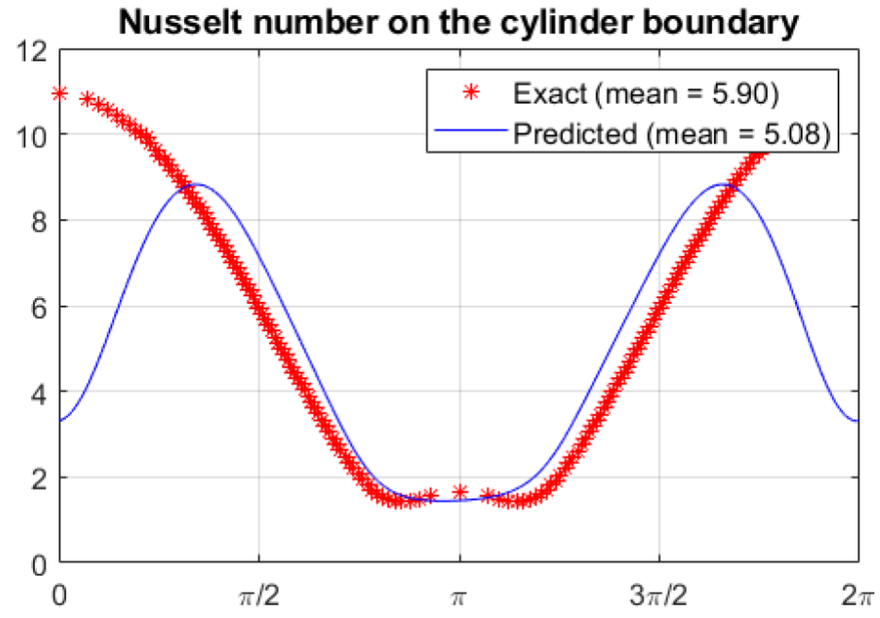


Figure: Results of 2D flow past a circular cylinder





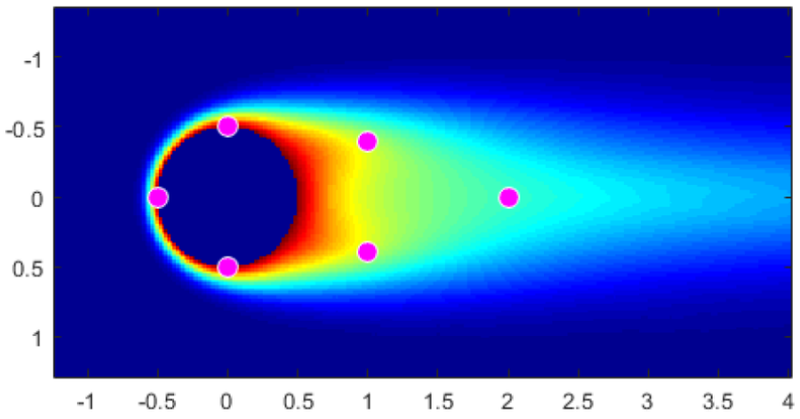


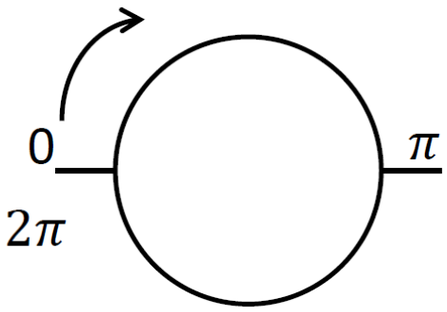
Discover constant Temperature on Cylinder

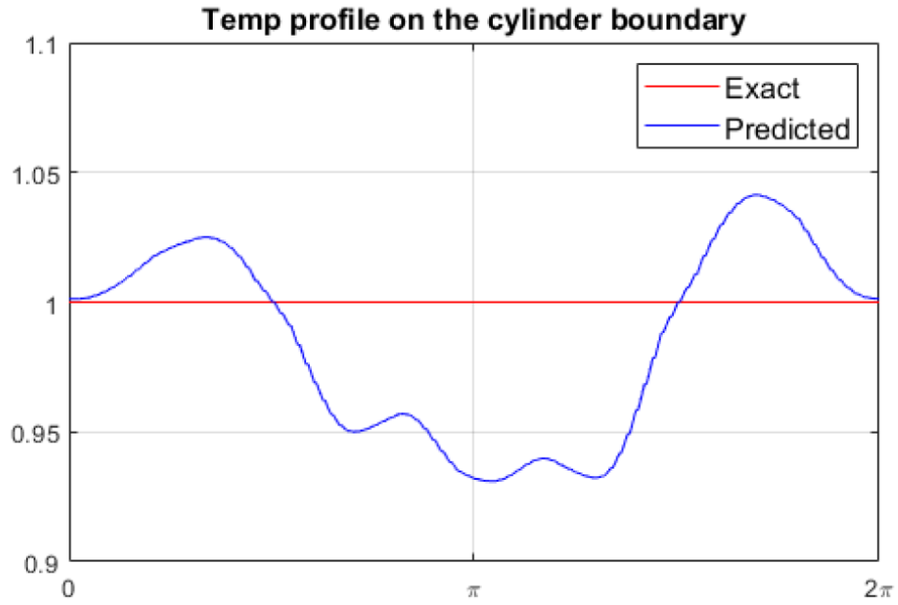
- ❑ Place a thermocouple on the front stagnation point

Experimental results

- Case B

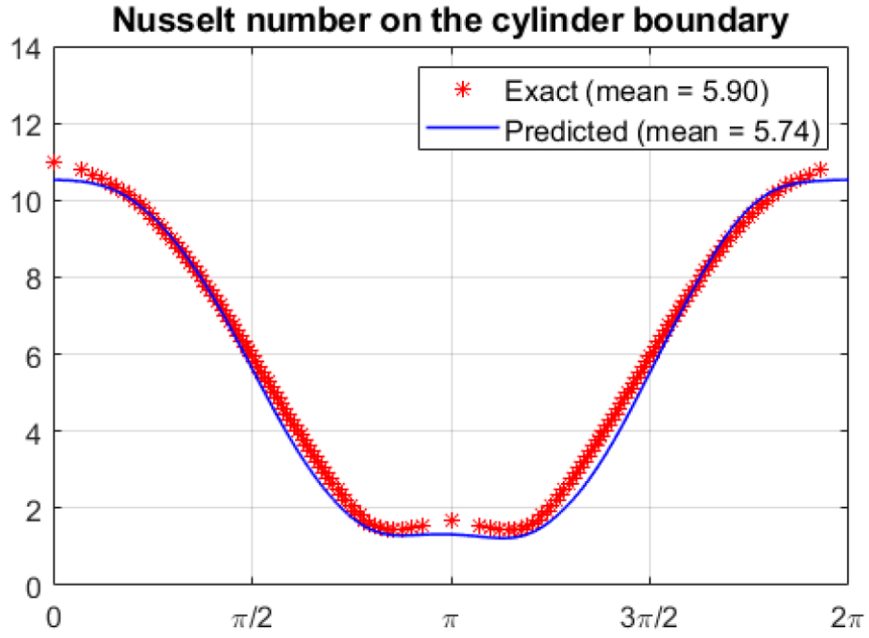






L2 error - u: 0.45 %
L2 error - v: 0.64 %
L2 error - p: 1.11 %

L2 error - T: 4.14 %

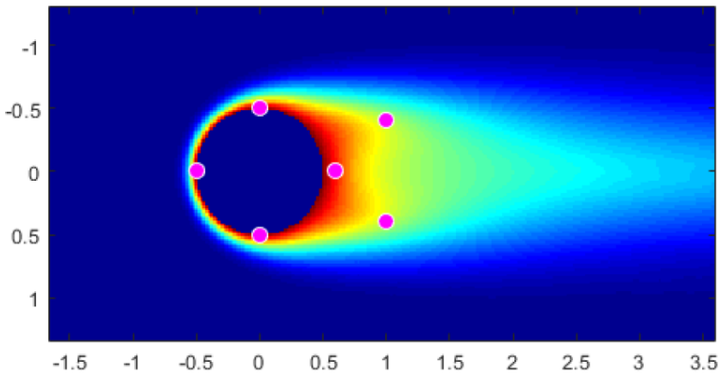


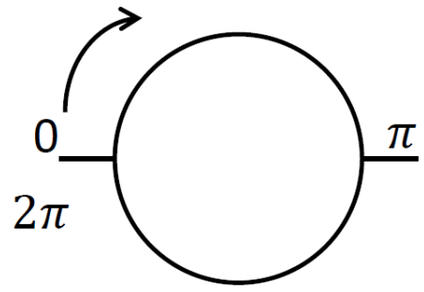
Discover constant Temperature on Cylinder

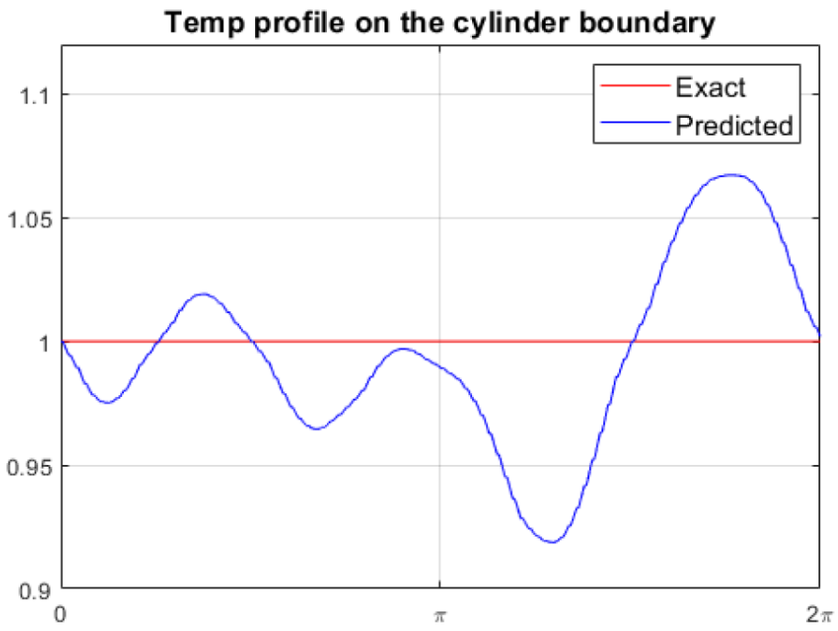
- ❑ Place a thermocouple on rear stagnation point

Experimental results

- Case C

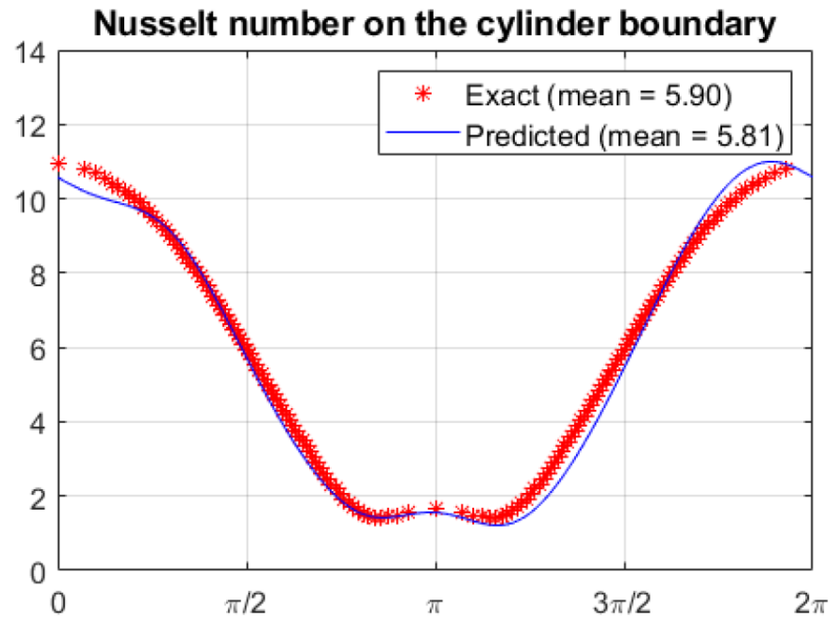






L2 error - u: 0.66 %
L2 error - v: 1.06 %
L2 error - p: 1.73 %

L2 error - T: 3.30 %



Discover constant Temperature on Cylinder

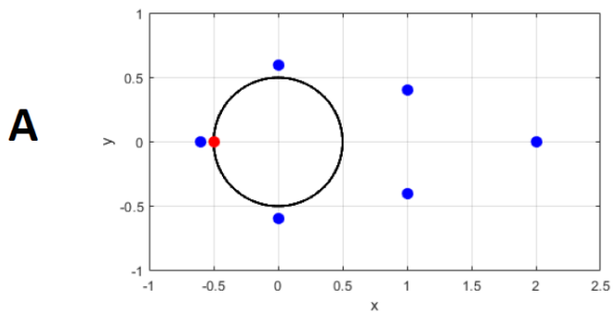
- ❑ Place two thermocouples close to front stagnation point

Experimental setup

Remark:
One measurement on the boundary is enough to get a decent result

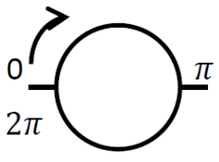
- Second group: Adding one sensor on the boundary

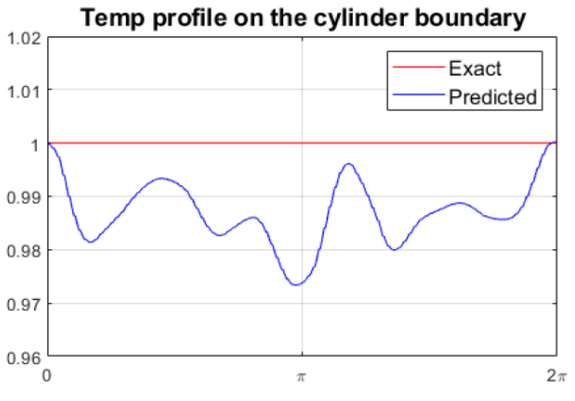
Sensor placement

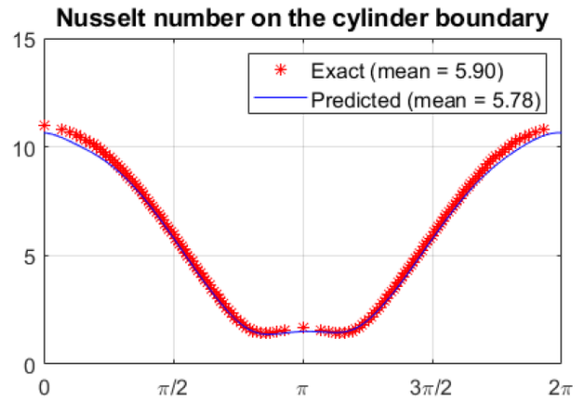


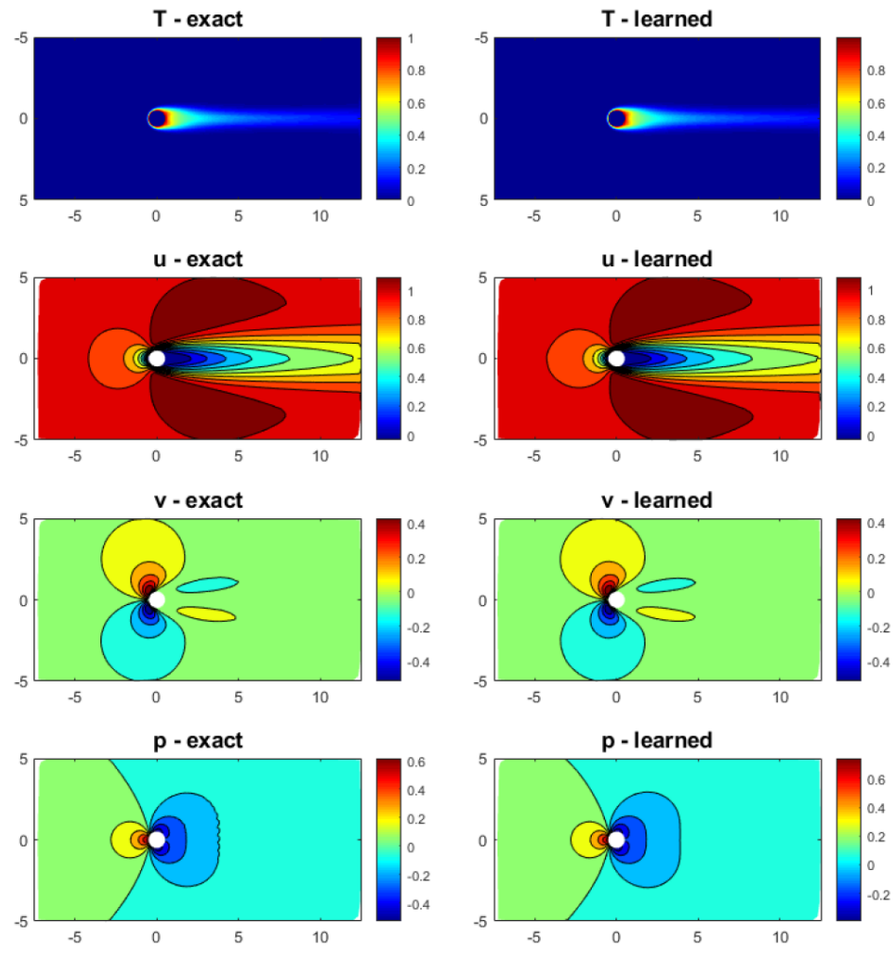
Result

L2 error - u: 0.31 %
L2 error - v: 0.50 %
L2 error - p: 0.85 %
L2 error - T: 1.28 %









Active Sensor Placement

Criterion:

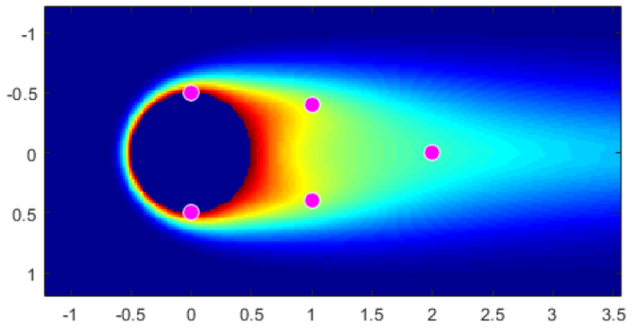
$$Res(x,y) = u\frac{\partial T}{\partial x} + v\frac{\partial T}{\partial y} - \kappa\Delta T$$

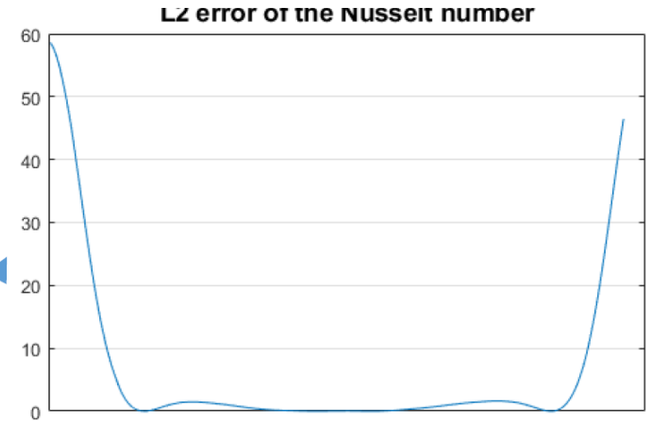
Algorithm:

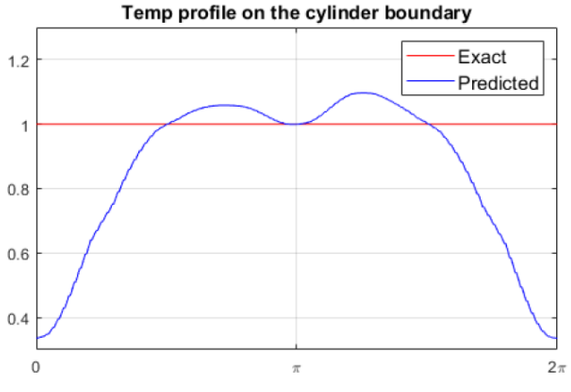
- **Step 1:** Giving initial configuration
- **Step 2:** HFM network training
- **Step 3:** Compute the residual on the boundary and find the position with maximum
- **Step 4:** Updating sensor placement and data, back to Step 2

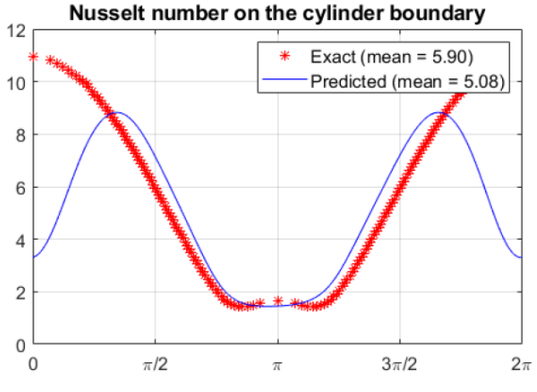
Case A

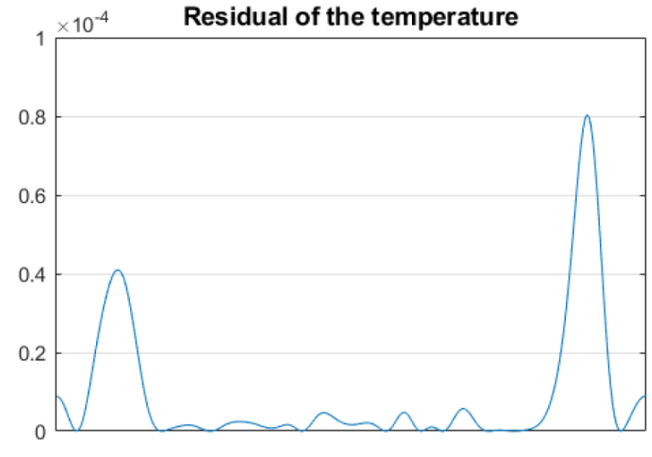
L2 error - u: 0.45 %
L2 error - v: 0.63 %
L2 error - p: 1.10 %
L2 error - T: 19.45 %







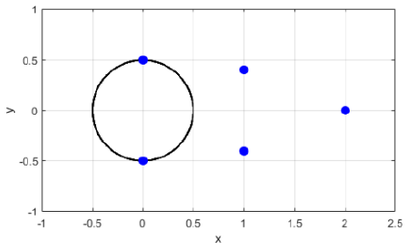


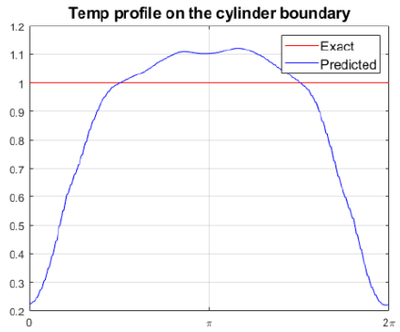


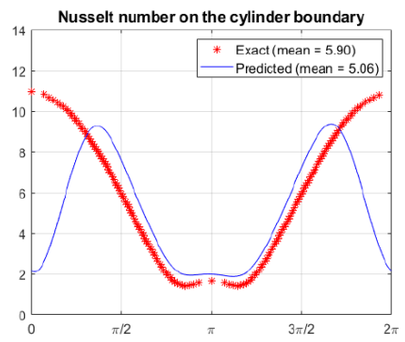
This figure shows that:
There is correlation between the prediction error and the residual

Active sensor placement

Initial placement



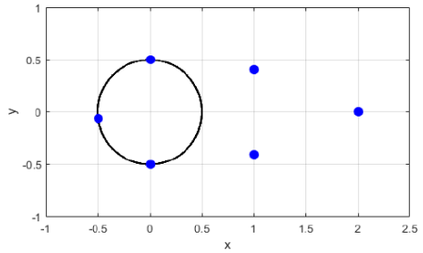


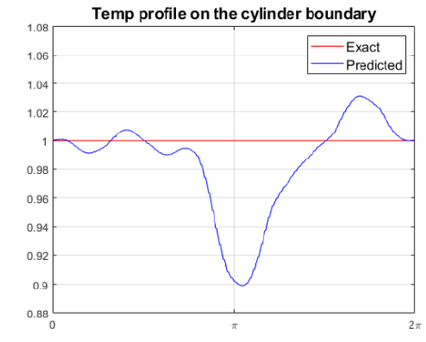


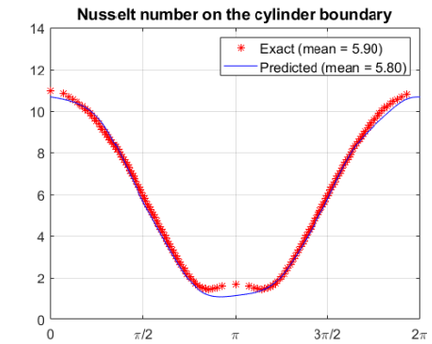
L2 error - T: 23.70 %

L2 error - Tb: 33.37 %

Iteration 1



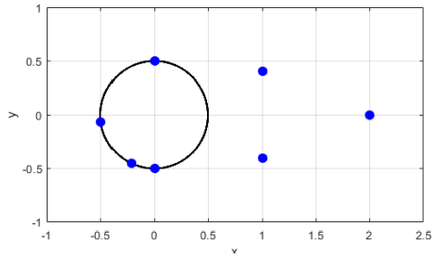


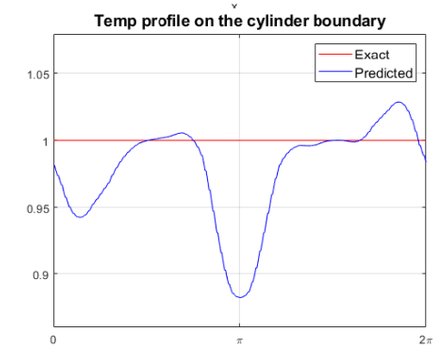


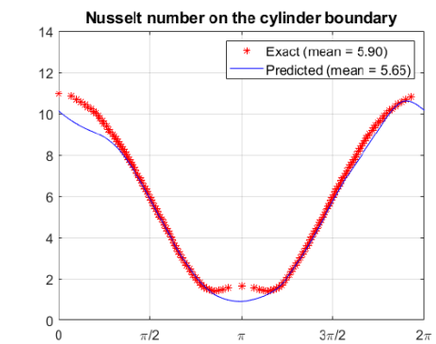
L2 error - T: 3.69 %

L2 error - Tb: 3.71 %

Iteration 2



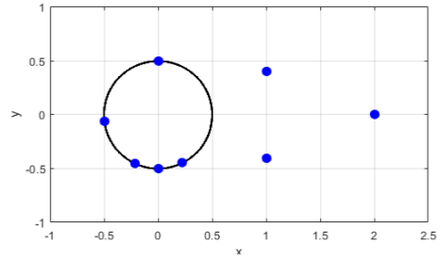


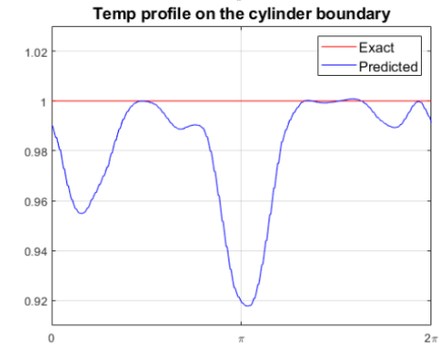


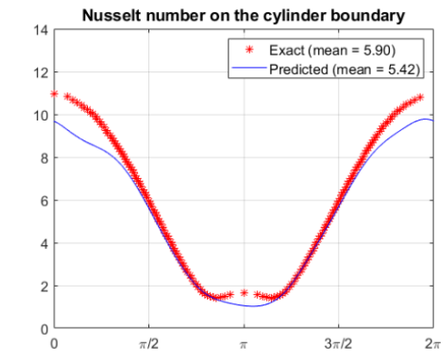
L2 error - T: 4.47 %

L2 error - Tb: 4.32 %

Iteration 3





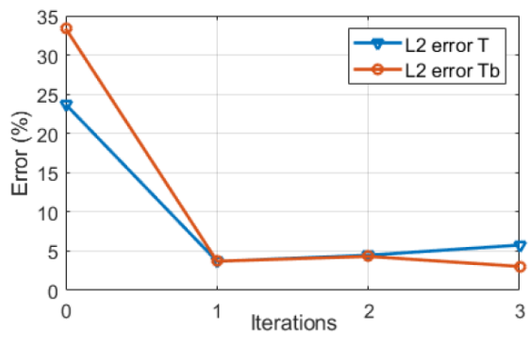


L2 error - T: 5.76 %

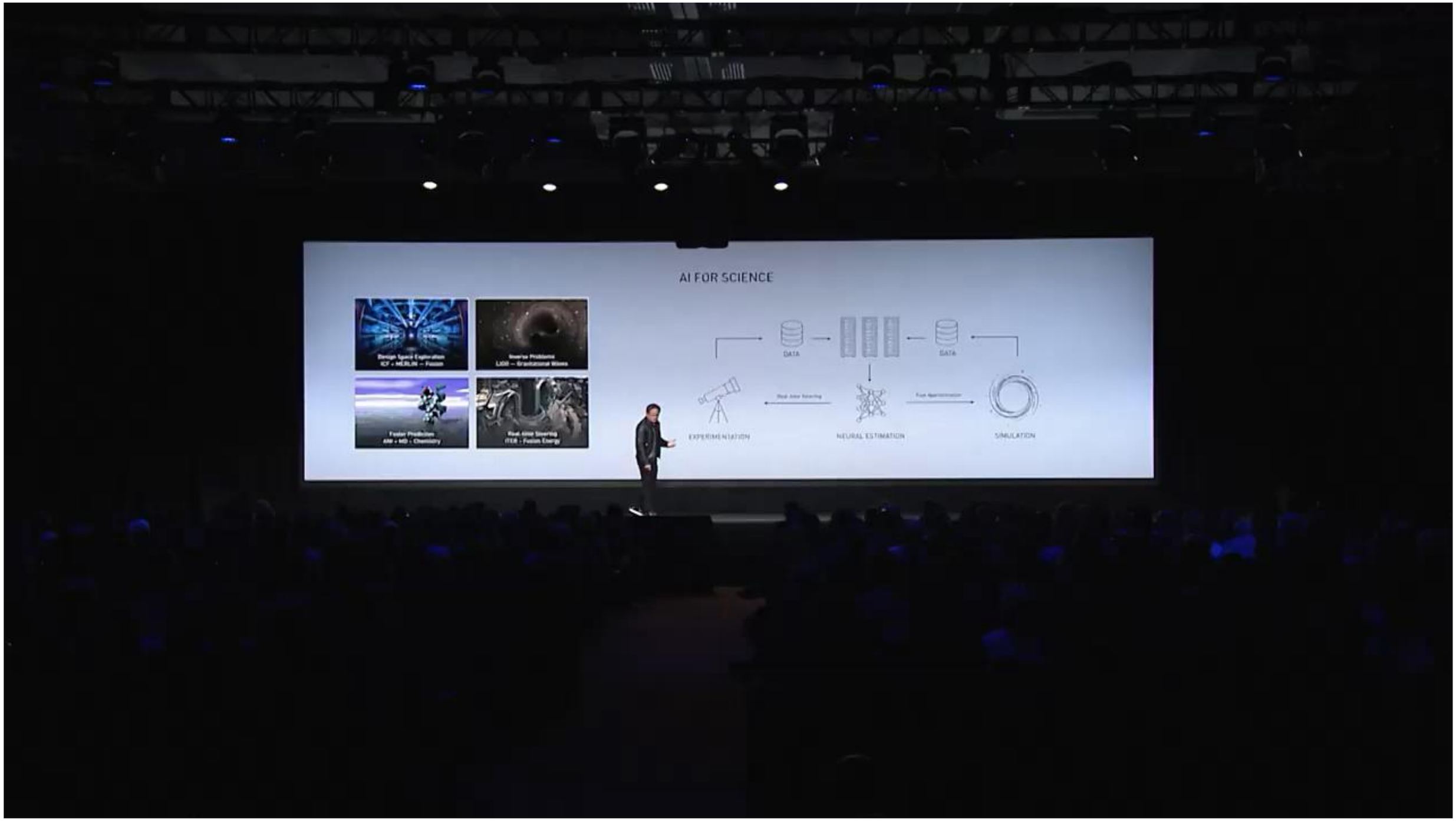
L2 error - Tb: 3.04 %

L2 error - T: mean error over the whole domain

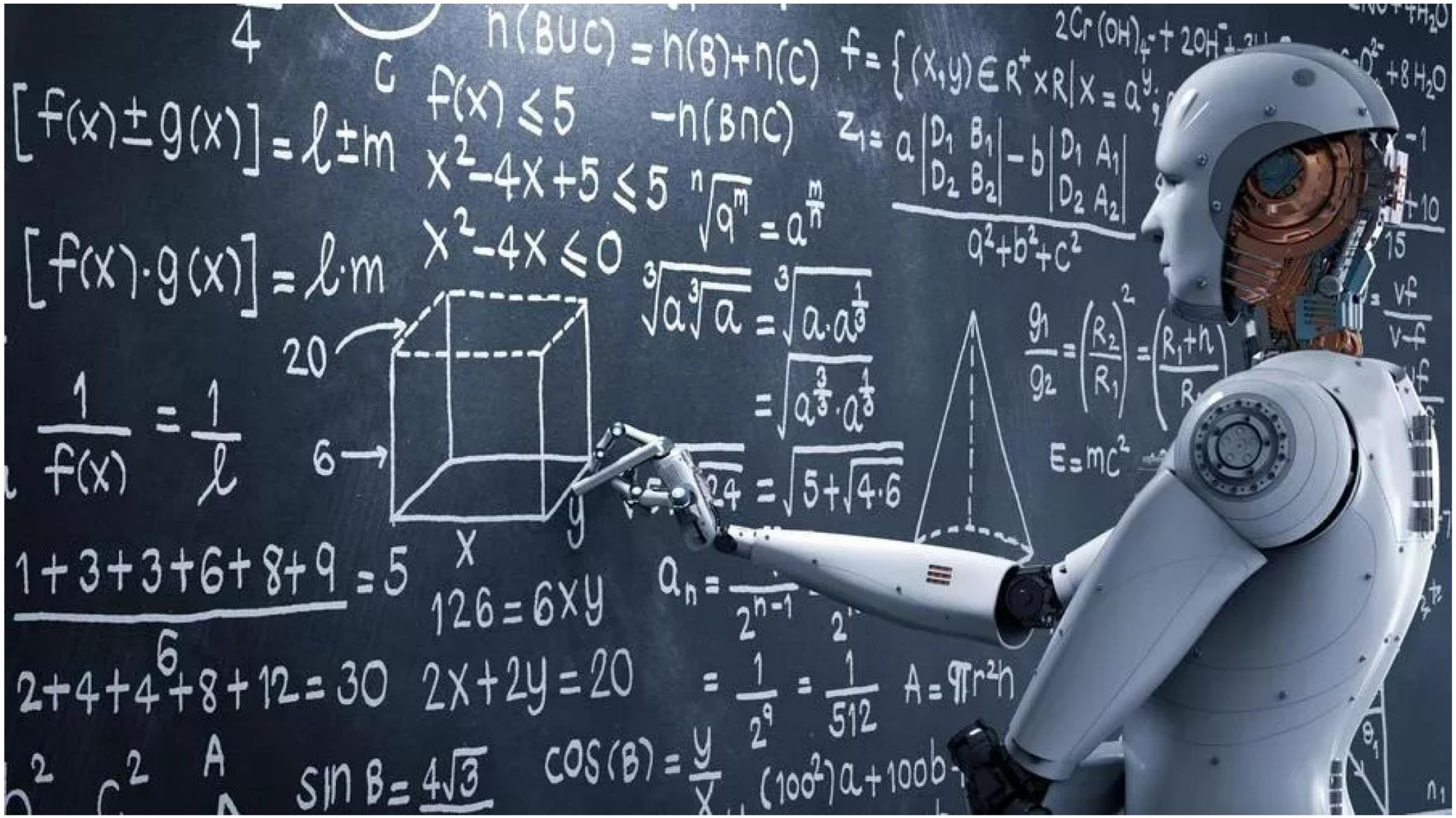
L2 error - Tb: mean error on the cylinder boundary



NVIDIA uses PINNs to design the new GPU chips



Do we need to teach Robots calculus?



Universal Approximation Theorem for Operator

$G : u \mapsto G(u)$, $G(u) : y \in \mathbb{R}^d \to \mathbb{R}$

Theorem (Chen & Chen, IEEE Trans. Neural Netw., 1995)

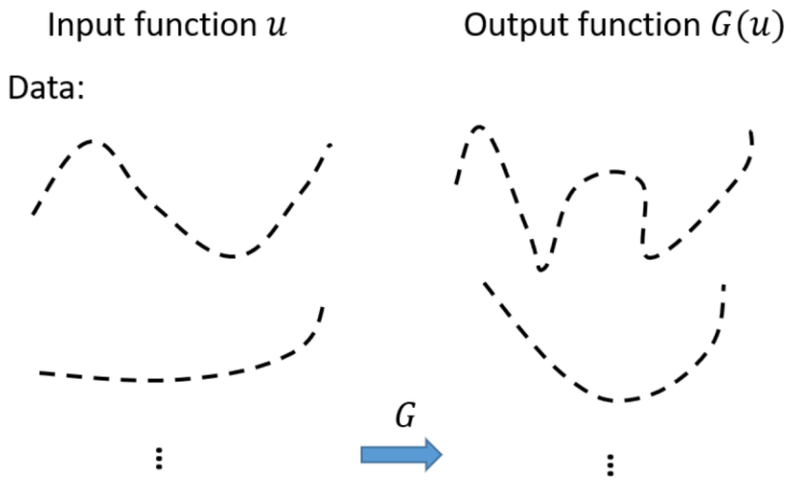
Suppose that σ is a continuous non-polynomial function, X is a Banach Space, $K_1 \subset X$, $K_2 \subset \mathbb{R}^d$ are two compact sets in X and $\mathbb{R}^d$, respectively, V is a compact set in $C(K_1)$, G is a nonlinear continuous operator, which maps V into $C(K_2)$. Then for any $\epsilon > 0$, there are positive integers n, p, m, constants $c_i^k, \xi_{ij}^k, \theta_i^k, \zeta_k \in \mathbb{R}$, $w_k \in \mathbb{R}^d$, $x_j \in K_1$, $i = 1, \ldots, n$, $k = 1, \ldots, p$, $j = 1, \ldots, m$, such that

$$\left| G(u)(y) - \sum_{k=1}^{p} \underbrace{\sum_{i=1}^{n} c_i^k \sigma \left(\sum_{j=1}^{m} \xi_{ij}^k u(x_j) + \theta_i^k \right)}_{branch} \underbrace{\sigma(w_k \cdot y + \zeta_k)}_{trunk} \right| < \epsilon$$

holds for all $u \in V$ and $y \in K_2$.

Problem setup

$$G : u \mapsto G(u)$$
$$G(u) : \mathbb{R}^d \to \mathbb{R}$$

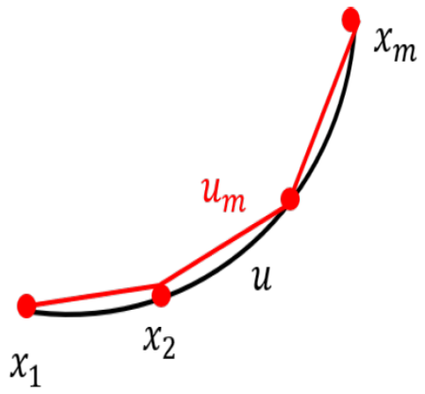


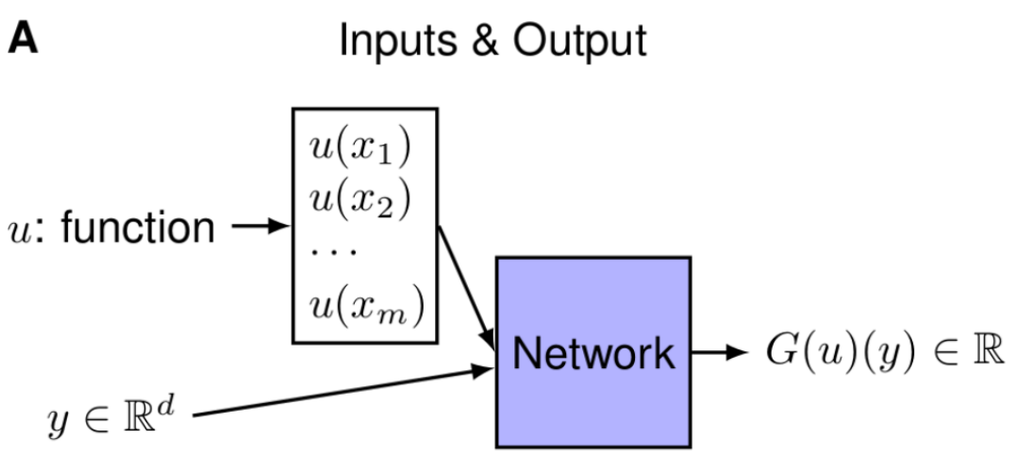
Number of sensors

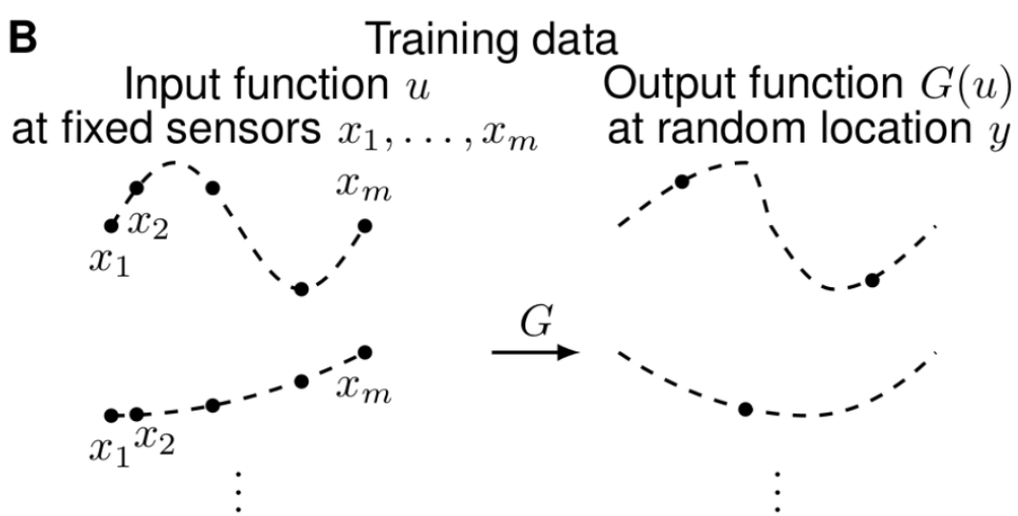
Consider $G : u(x) \mapsto \boldsymbol{s}(x)$ $(x \in [0,1])$ by ODE

$$\frac{d}{dx}\boldsymbol{s}(x) = \boldsymbol{g}(\boldsymbol{s}(x), u(x), x), \quad \boldsymbol{s}(0) = \boldsymbol{s_0}$$

$$u \in V \Rightarrow u_m \in V_m$$







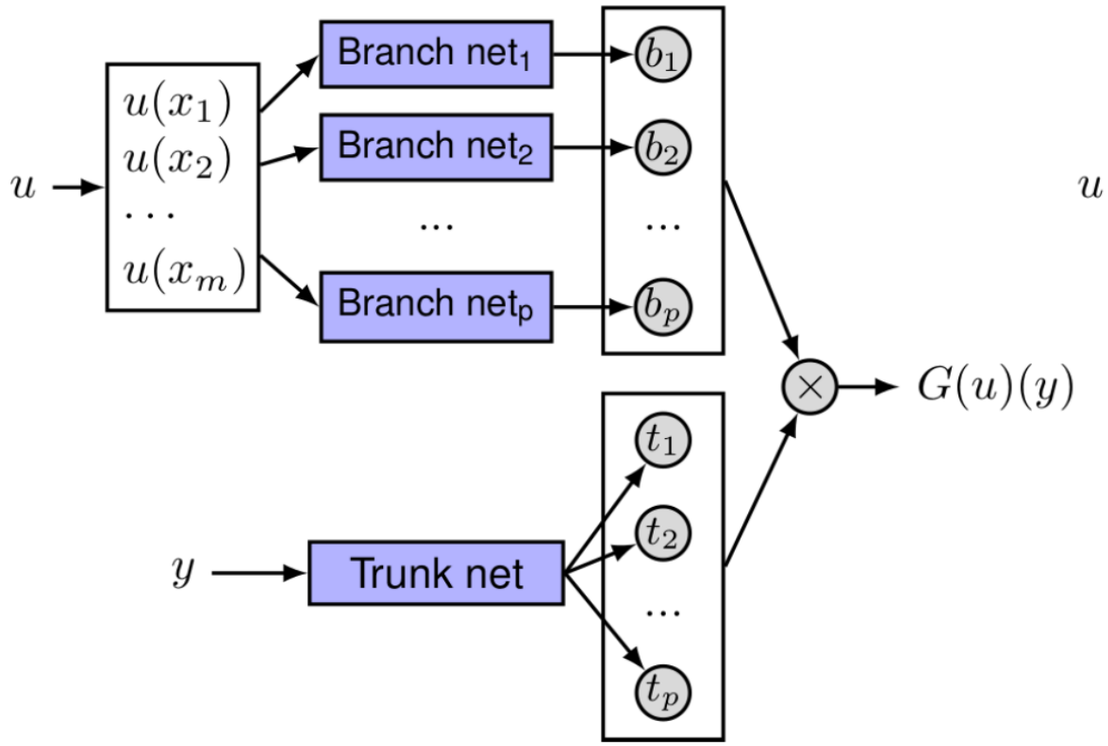
Next, how to design the network?

Deep operator network (DeepONet)

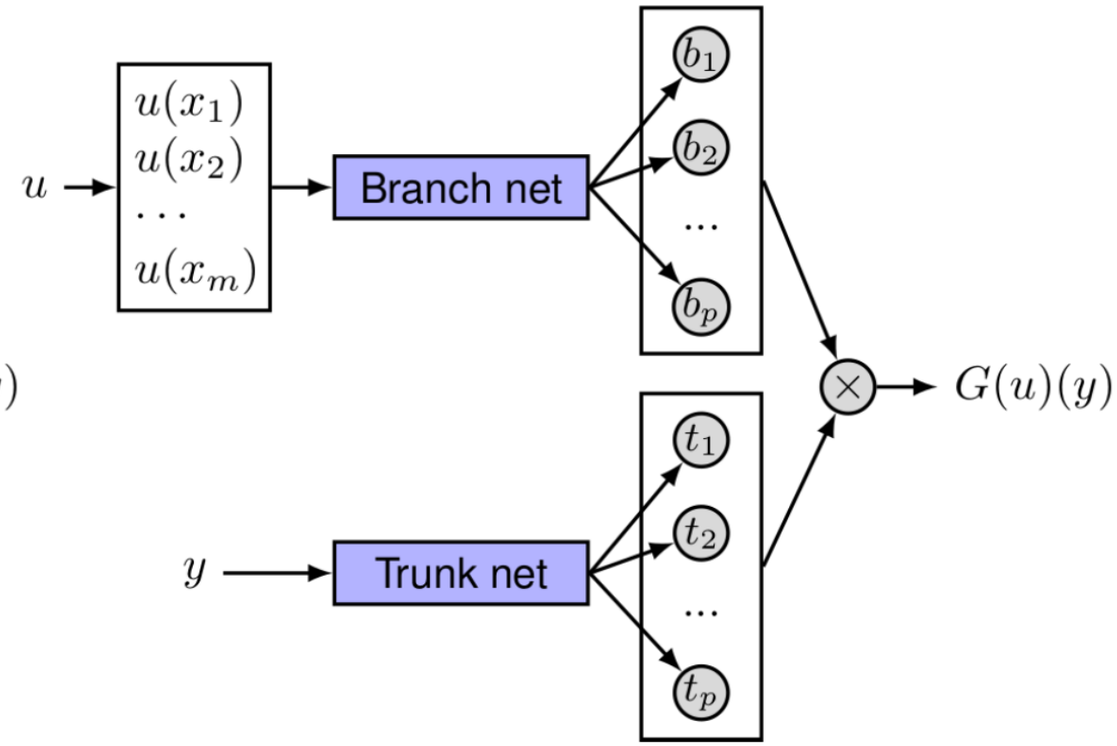
Recall the Theorem:

$$G(u)(y) \approx \sum_{k=1}^{p} \underbrace{\sum_{i=1}^{n} c_i^k \sigma \left(\sum_{j=1}^{m} \xi_{ij}^k u(x_j) + \theta_i^k \right)}_{branch} \underbrace{\sigma(w_k \cdot y + \zeta_k)}_{trunk}$$

C Stacked DeepONet



D Unstacked DeepONet



BROWN

A simple ODE case

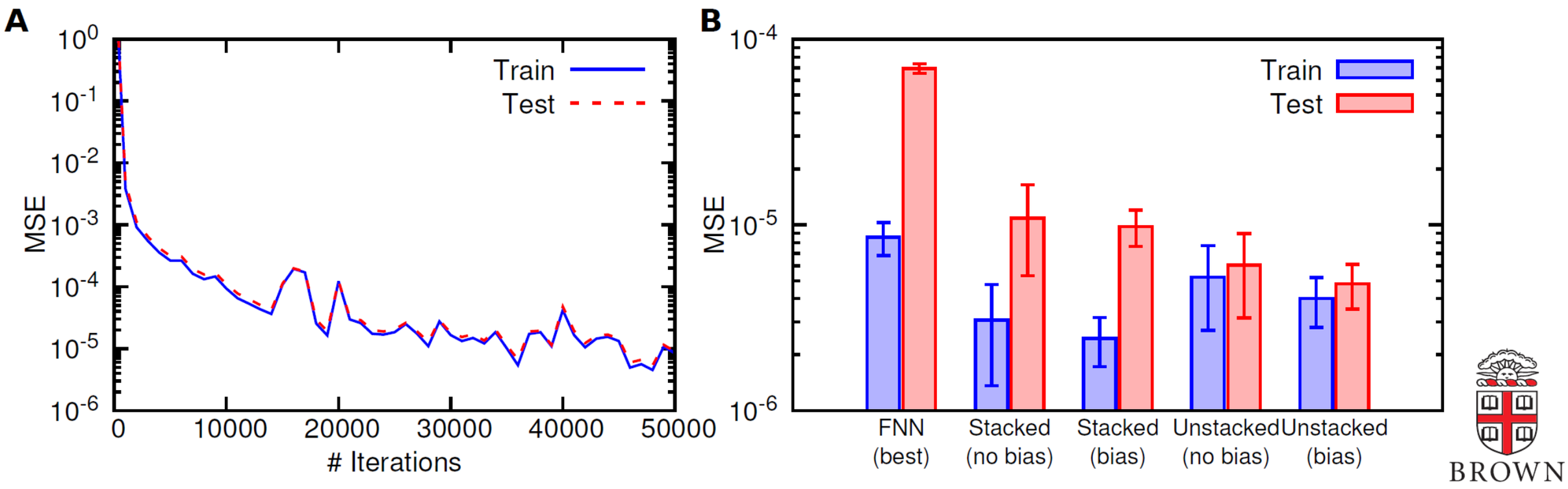
$$\frac{ds(x)}{dx} = u(x), \quad x \in [0, 1],$$

with an initial condition $s(0) = 0$.

$$G : u(x) \mapsto s(x) = \int_0^x u(\tau) d\tau.$$

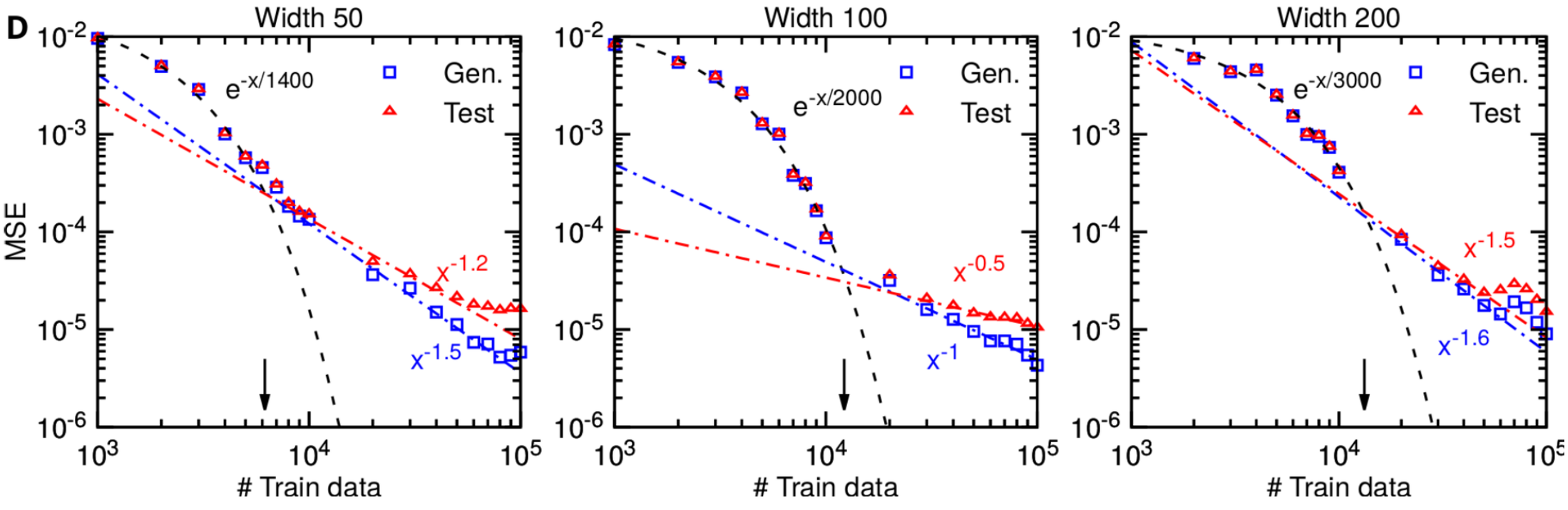
100 sensors, 10000×1 training points

Very small generalization error!



Gravity pendulum with an external force $u(t)$

$$\frac{ds_1}{dt} = s_2, \quad \frac{ds_2}{dt} = -k \sin s_1 + u(t)$$



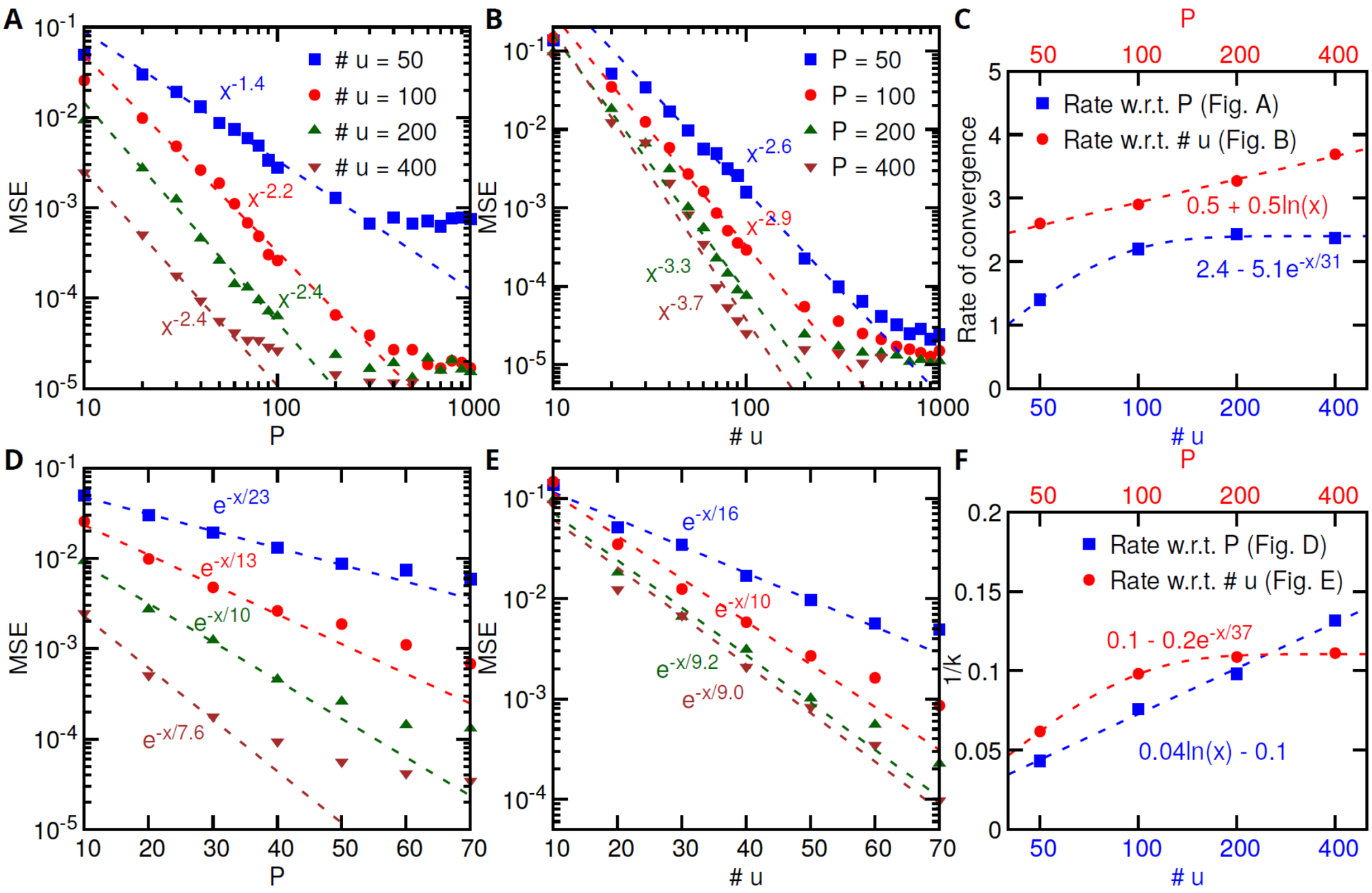
Test/generalization error:

- small dataset: exponential convergence
- large dataset: polynomial rates
- smaller network has earlier transition point

BROWN

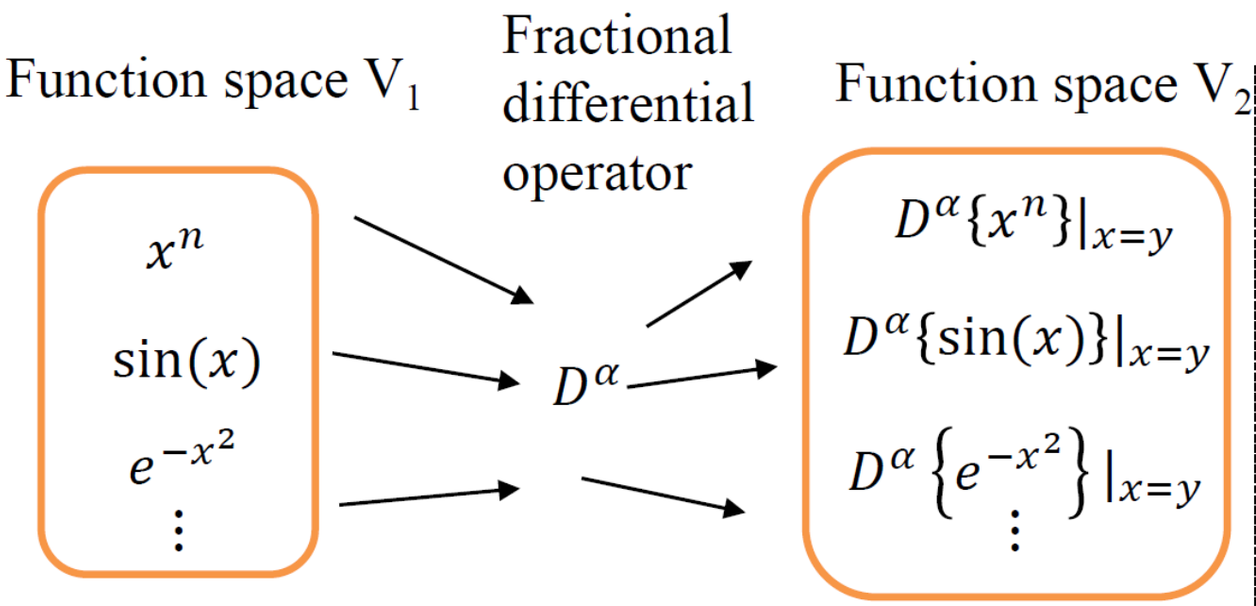
Diffusion-reaction system

exponential/polynomial convergence



BROWN

DeepONets for approximating fractional differential operators



Train a DeepONet $G()$ to approximate D^α by minimizing the following ***loss function***

$$L(\boldsymbol{\theta}_B, \boldsymbol{\theta}_T) = \left[D^\alpha\{x^n\}|_{x=y} - G\,(x^n, y, \alpha; \boldsymbol{\theta}_B, \boldsymbol{\theta}_T) \right]^2 + \left[D^\alpha\{\sin(x)\}|_{x=y} - G(\sin(x), y, \alpha; \boldsymbol{\theta}_B, \boldsymbol{\theta}_T) \right]^2 + \left[D^\alpha\left\{e^{-x^2}\right\}|_{x=y} - G(e^{-x^2}, y, \alpha; \boldsymbol{\theta}_B, \boldsymbol{\theta}_T) \right]^2 + \cdots$$

Discrete version for member functions in V_1

$$\left[\; x_1^n, \quad x_2^n, \quad \dots, \quad x_m^n \;\right]$$

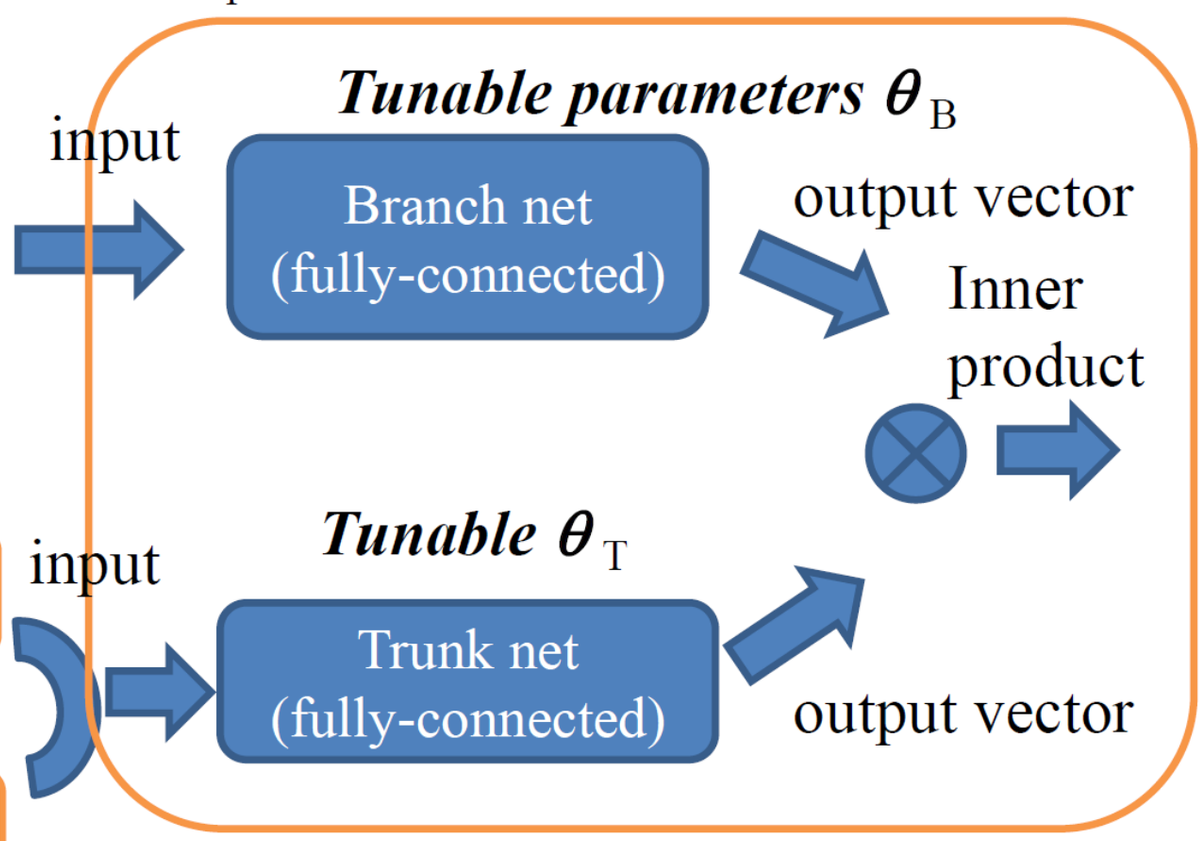
$$\left[\sin(x_1), \sin(x_2), \cdots, \sin(x_m)\right]$$

$$\begin{bmatrix} e^{-x_1^2}, & e^{-x_2^2}, & \cdots, & e^{-x_m^2} \end{bmatrix}$$

Sampled fractional order α

Selected locations where we evaluate the member functions in V_2 y

Deep ONet $G(u(x), y, \alpha; \theta_B, \theta_T)$



$G(x^n, y, \alpha)$

$G(\sin(x), y, \alpha)$

$G(e^{-x^2}, y, \alpha)$

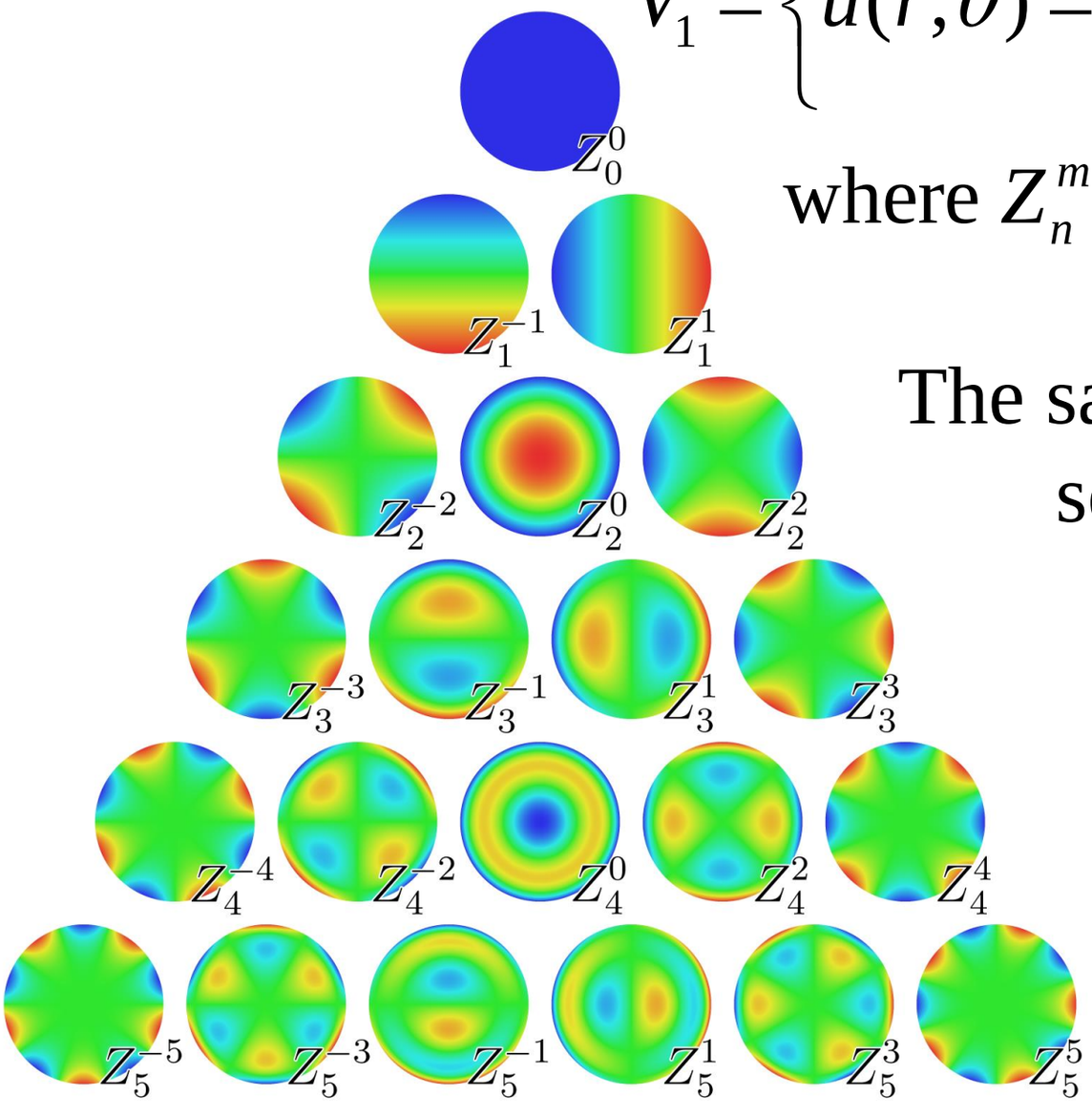
$\vdots$

The second operator we approximate is **a 2D Riesz fractional Laplacian** $D^{\alpha} = (-\Delta)^{\alpha/2}$ defined on a unit disk

The function space V_1 where u stays is now assumed to be an orthogonal polynomial space spanned by 36 Zernike polynomials:

$$V_1 = \left\{ u(r,\theta) = \sum_{n=0}^{7} \sum_{m=-n}^{n} a_{mn} Z_n^m(r,\theta), \;\; r \in [0,1], \theta \in [0,2\pi] \right\},$$

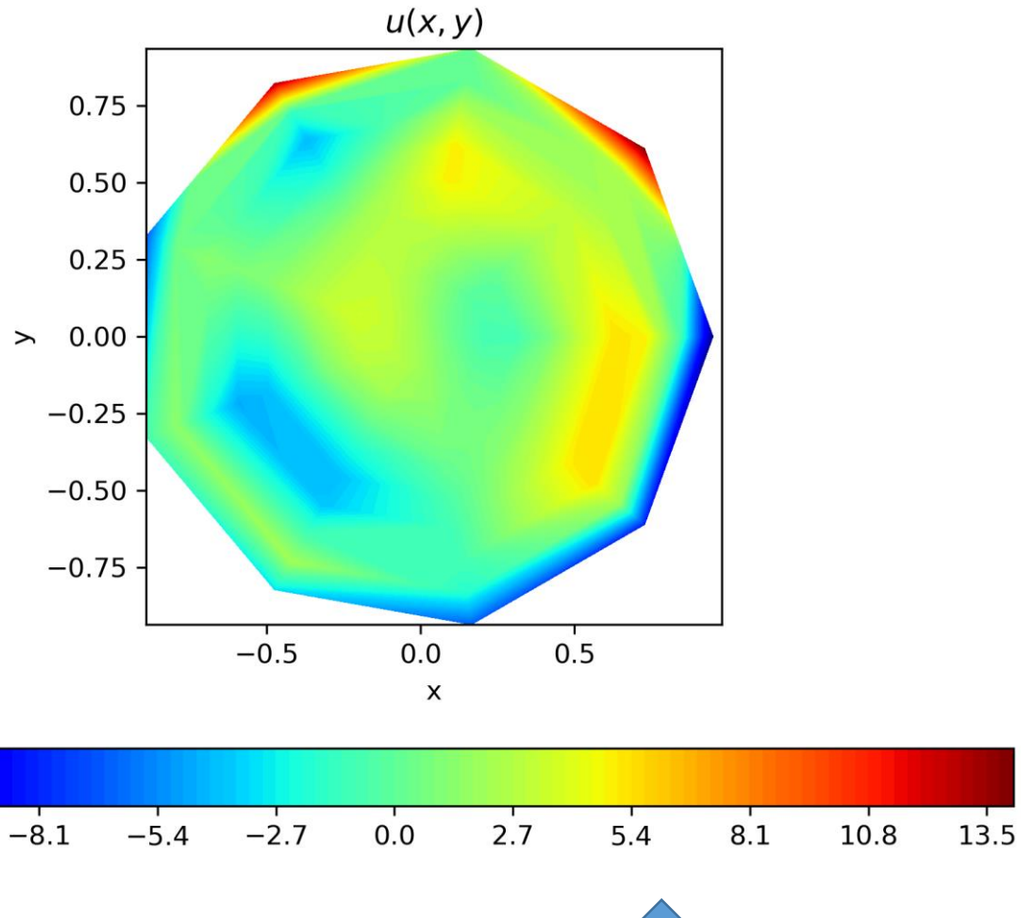
where $Z_n^m(r,\theta) \equiv 0$ for $n - m$ odd



The samples of u were generated by randomly selecting the coefficients $a_{mn} \in [-1,1]$

The first 21 Zernike polynomials $Z_{n\leq 5}^m$

Ten million training points $\{(u_i(r,\theta), y_j, \alpha_k)\}$ for i=1,…,N_u; j=1,…,N_y; k=1,…,N_a, where N_u = 1e4, N_y=100, N_a=10 (thus $N_u \times N_y \times N_a$ = 10 millions)
We employed mini-batch in training and set the batch-size to be 0.4 million.

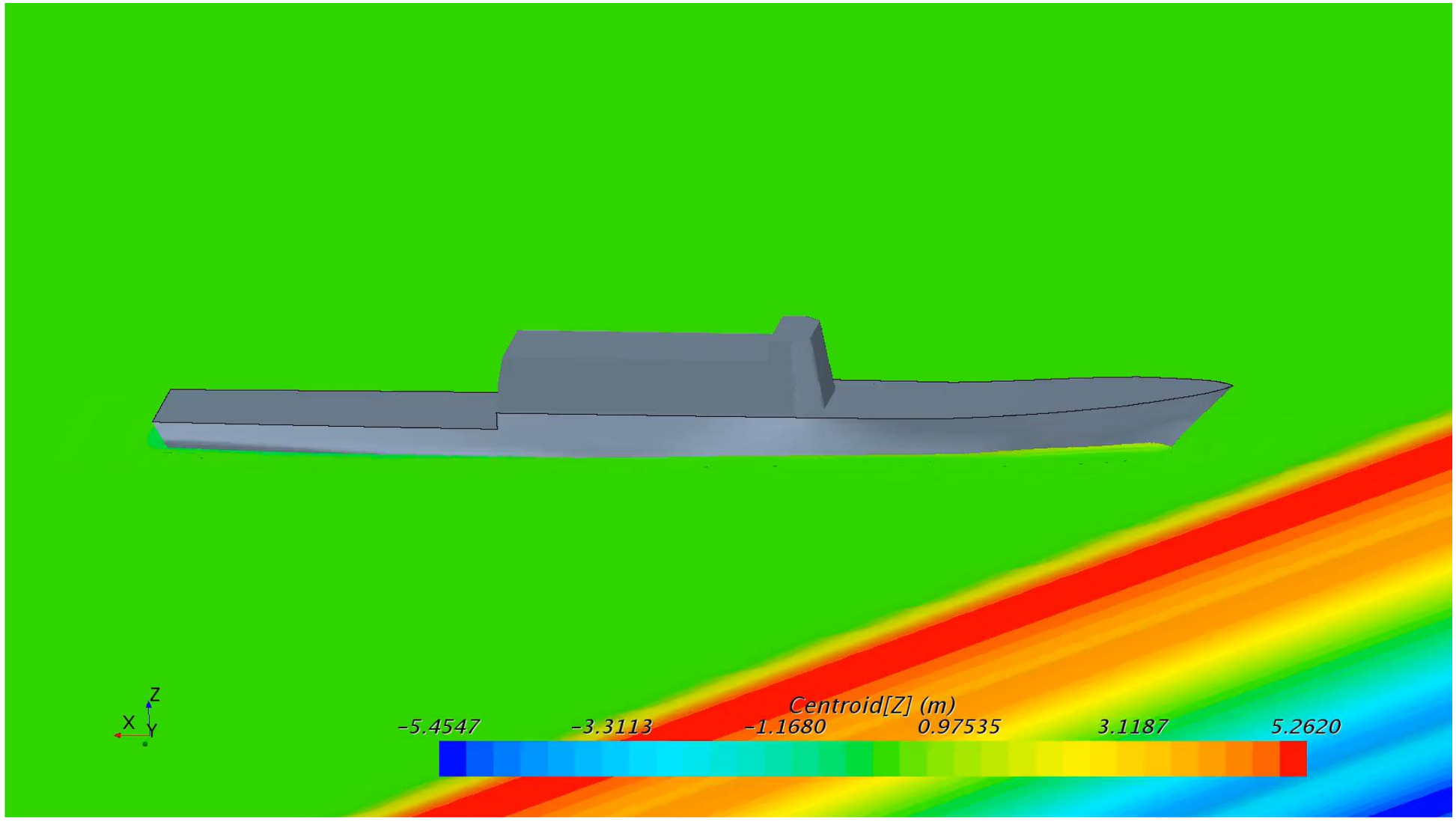


The 10000 samples of u in the training set were generated by randomly selecting the orthogonal expansion coefficients a_{mn} *in* [-1,1], while the 10000 u samples in the test set were also randomly sampled but with a different random seed.

A specific sample of u in the test set

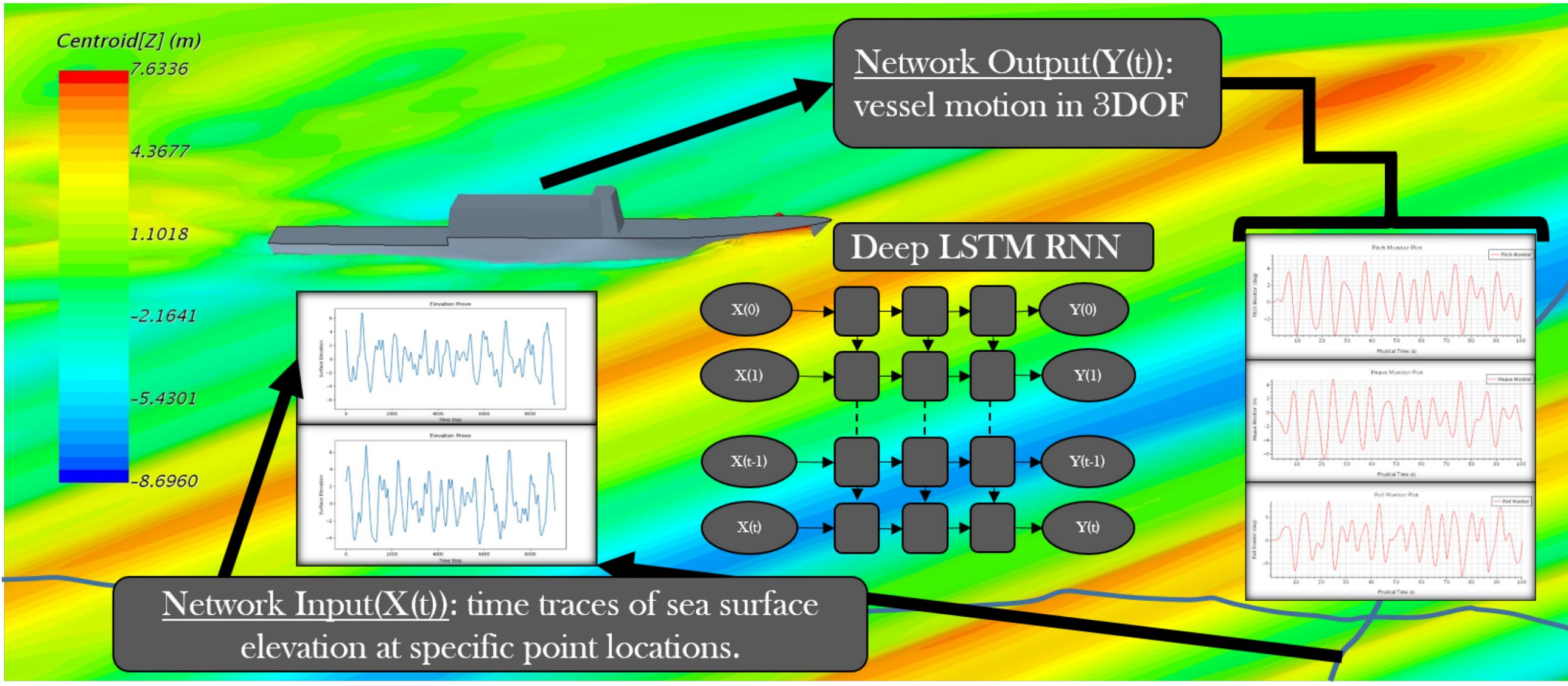
Calculus-agnostic Simulation

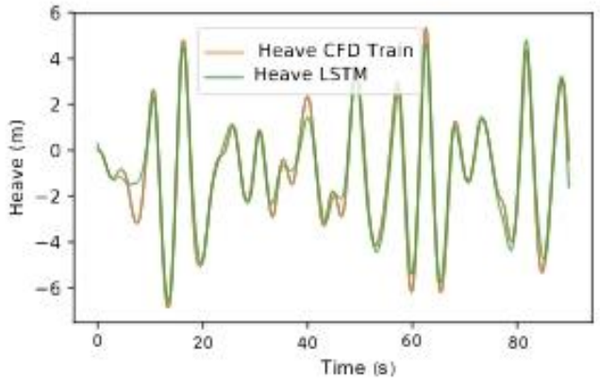
DeepFnet: Functional Approximation



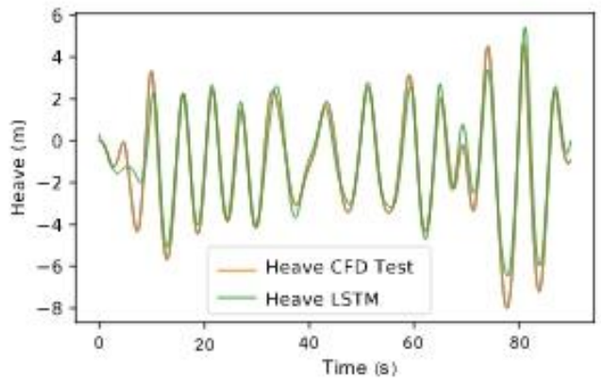
Predicting the motion of battleships in extreme sea states

DeepFnet: Functional Approximation

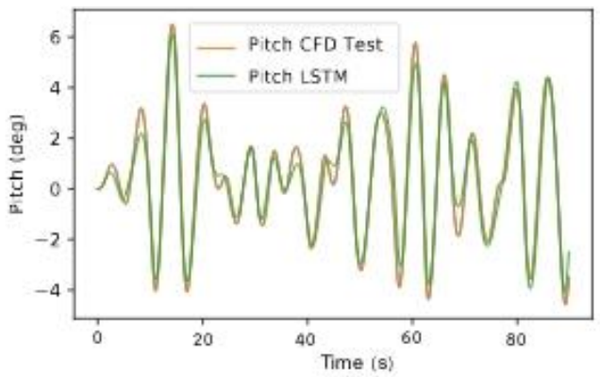




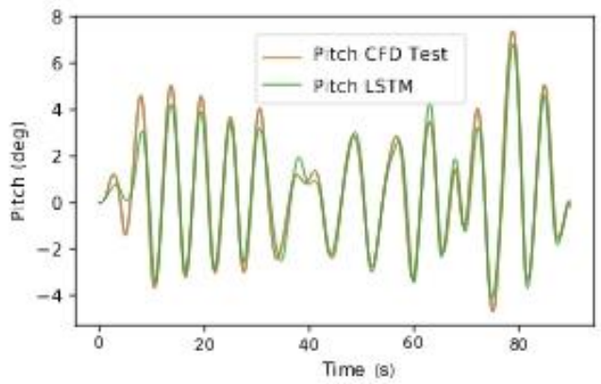
(a) Test - Relative MSE = 0.0380



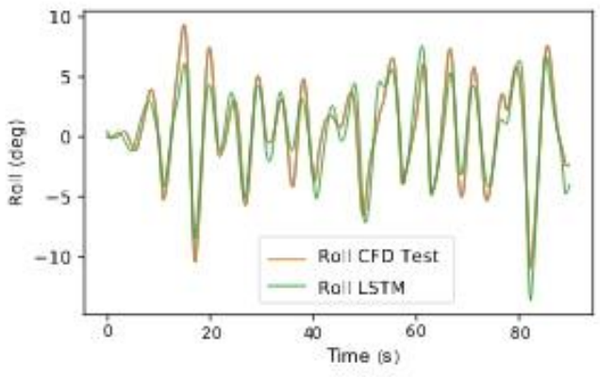
(b) Test - Relative MSE = 0.0773



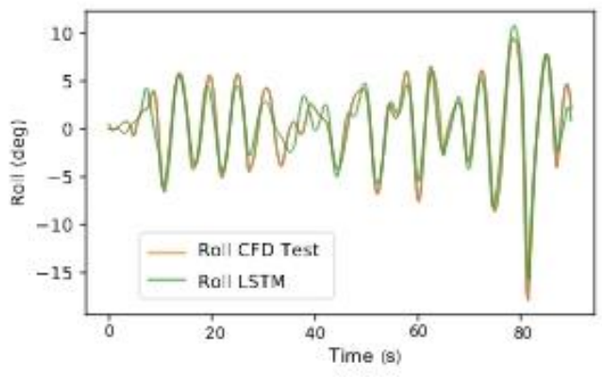
(c) Test - Relative MSE = 0.0444



(d) Test - Relative MSE = 0.0463



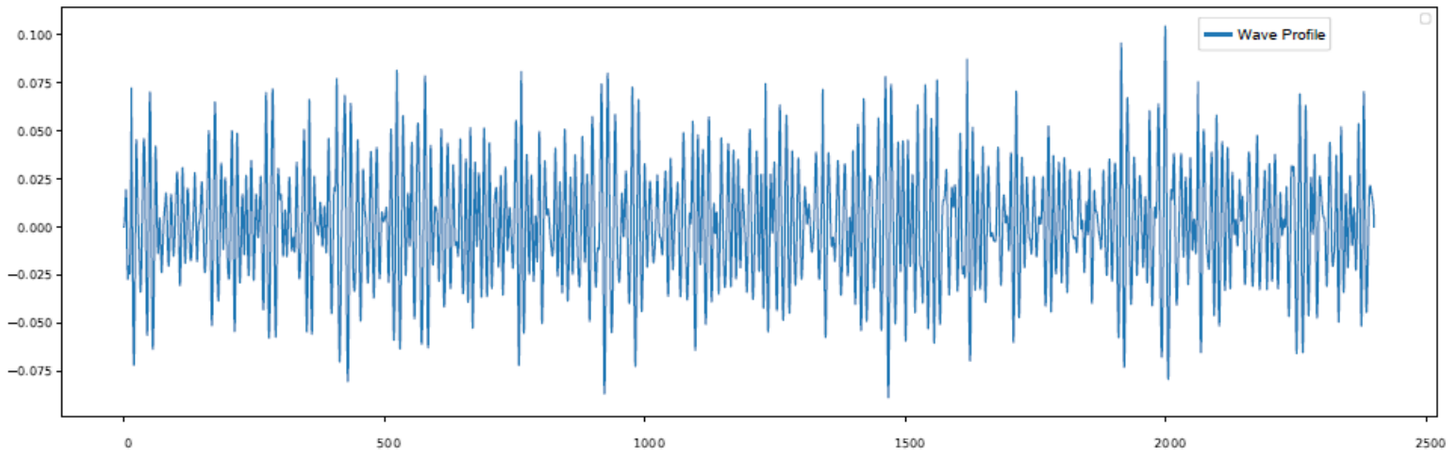
(e) Test - Relative MSE = 0.1490



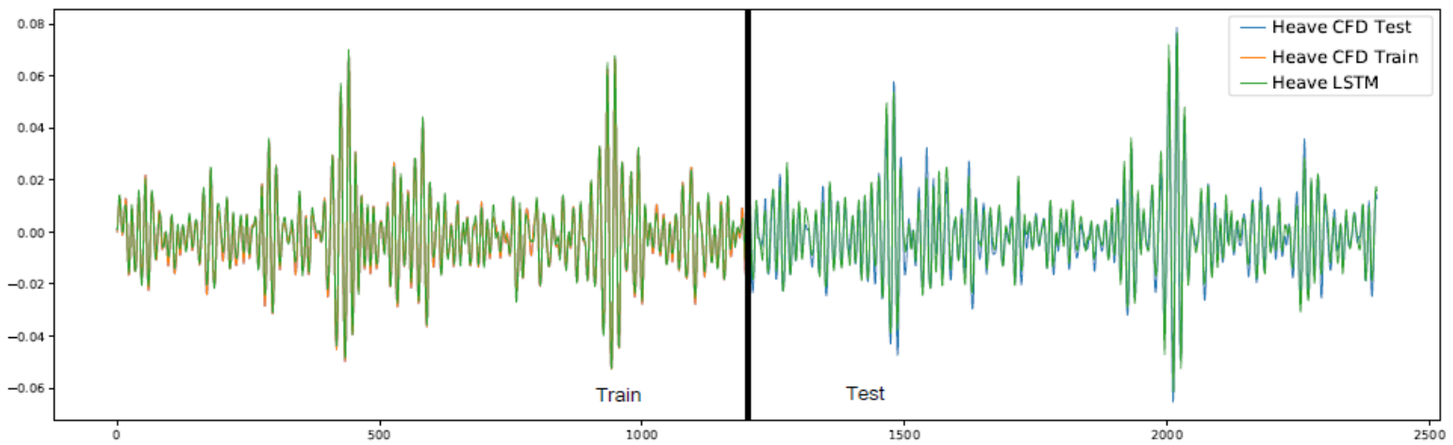
(f) Test - Relative MSE = 0.1215

Figure 10: *Test results network architecture 3, MSE=0.165.* **Heave, Pitch and Roll** motions for 3DOF motion approximation of a notional DTBM battleship sailing in WMO sea state at high Froude Number (0.4). The motion approximations provided in the unseen realizations of the sea state obtain a good performance both in amplitude and frequency approximation of the vessel response. It is interesting to notice that, at simple sight, we can see that roll spectra is very different to that of heave and pitch motions, that have more linear characteristics and are similar to each other. It is not surprising to realize that the performance of the network approximating roll motions is somewhat worse compared to heave and pitch motions. The network parameters are: {*Hidden Units, Number of Layers, Number of Train Steps*} = {90,4,5000}. Parameters that are not relevant remain constant. The inputs provided to the network are shown in Fig. 7.

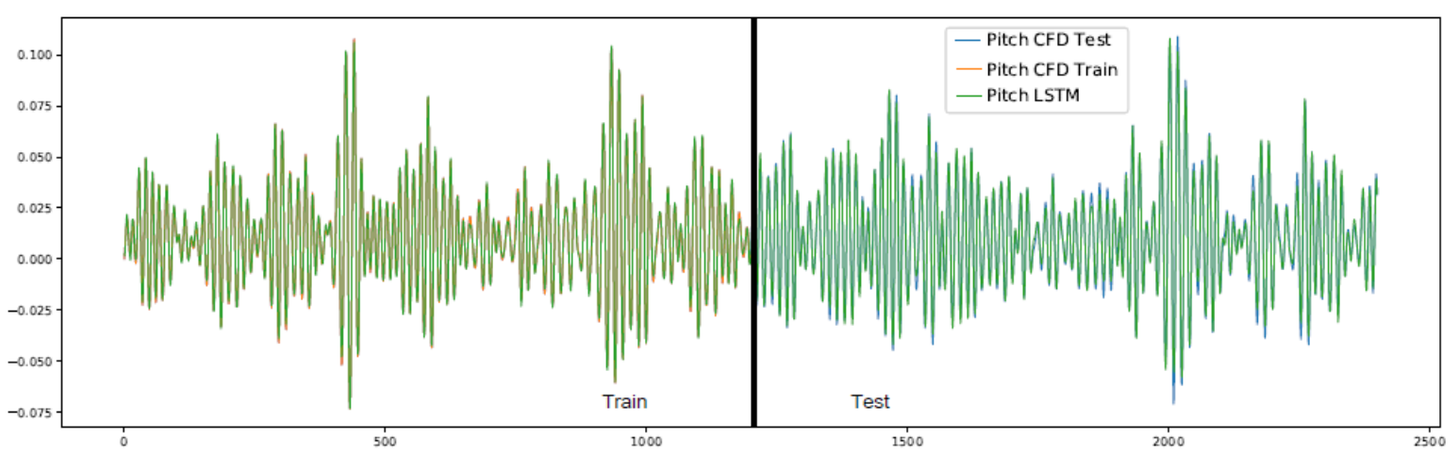
Long-Time Prediction of Catamaran SeaKeeping



(a)



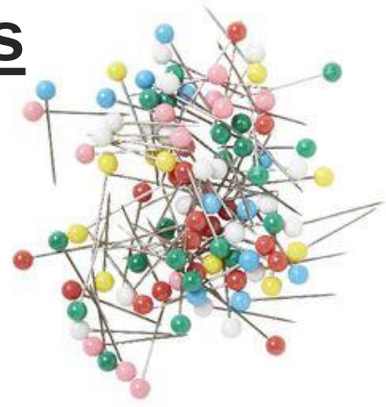
(b)



(c)

Figure 9: *Long-time predictions of vertical (b) and angular (c) motions of the catamaran vessel using a LSTM network with 2 layers and 10 neurons.* Figure (a) shows the surface elevation input to the network. The vertical line in (b),(c) denotes the beginning of testing. Each time step corresponds to $\Delta t = 0.0625s$.

SMU: Clements Scientific Computing Seminar Series



Speaker: Prof. George Karniadakis, Applied Math, Brown University, Tuesday 3:45-4:45p, February 11

Mathematics of PINNs

- Multifidelity approximation

  arXiv:1903.00104

- Nonlinear approximation theory

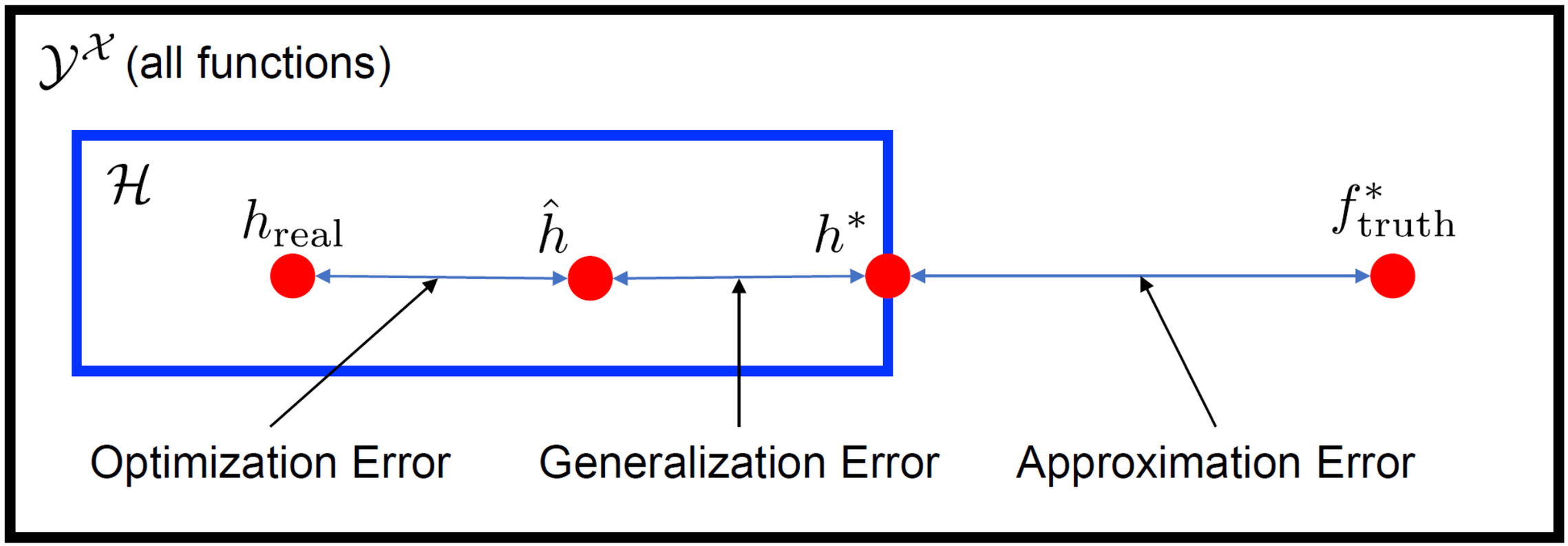
- Robust training & optimization

  arXiv:1808.04947; arXiv:1903.06733

- Learnability & small data

  arXiv:1905.01205

Overview: Statistical Learning perspective

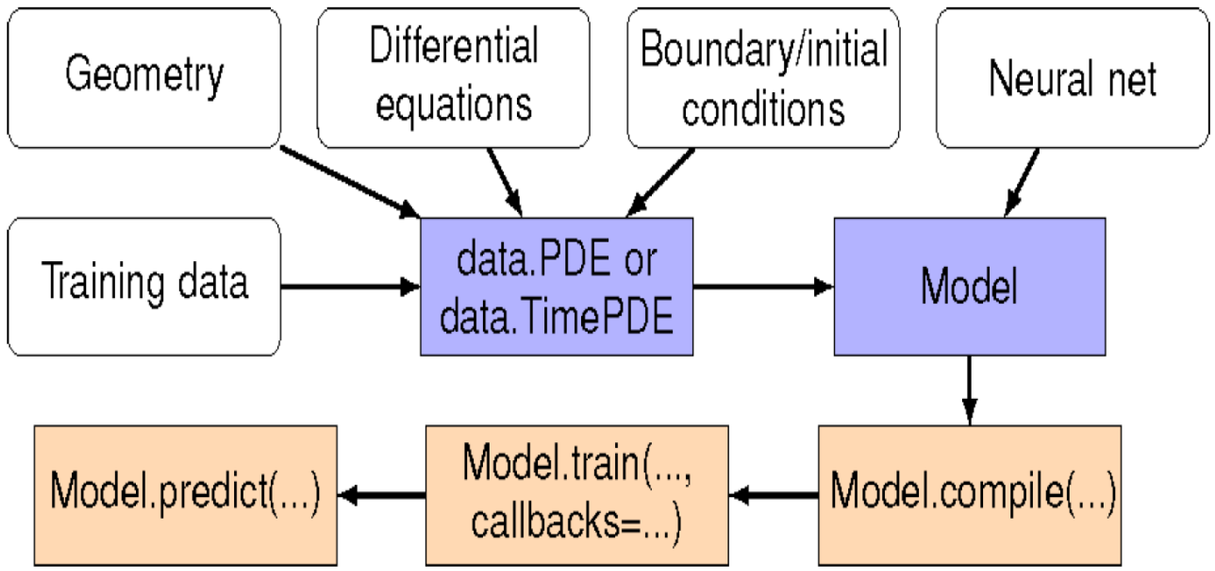


- General loss: $\mathcal{L}_{\mathcal{D}}(h) = \mathbb{E}_{(x,y)\sim\mathcal{D}}\left[\ell(h(x), y)\right] \implies h^*$
- Empirical loss: $\mathcal{L}_{\mathcal{T}_m}(h) = \frac{1}{m}\sum_{i=1}^{m} \ell(h(x_i), y_i) \implies \hat{h}$
- h_{real}: an actual approximation (e.g. after 1M gradient descent iterations)

SIAM Review – to appear

DeepXDE

arXiv:1907.0402



Pseudocode for DeepXDE

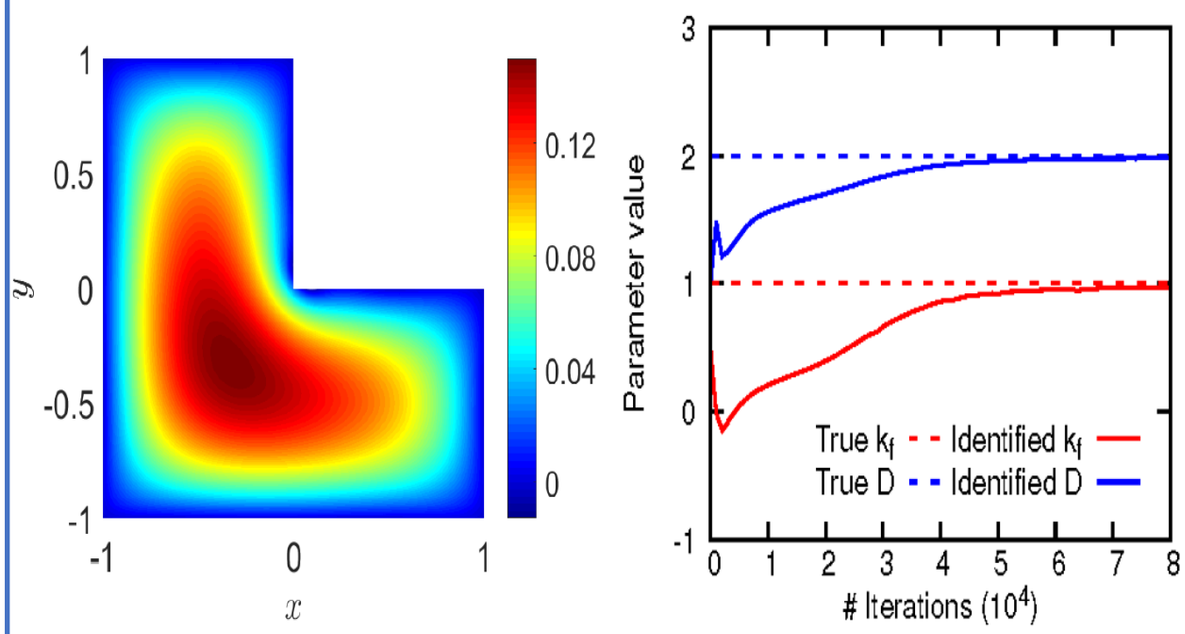
- import tensorflow/deepxde …
- def PDE/ODE/IDE — define the XDEs to be solved
- x/t_interval = deepxde.geometry.Interval(a, b)/TimeDomain(t_1, t_2)
- bcs/ics = deepxde.bc/ic — Dirichlet/Neumann/Robin…
- data = deepxde.data — select training data for PDEs
- net = deepxde.maps.FNN(architecture, activation, initialization) -- define neural networks
- model = deepxde.model(data, net) — build model
- model.compile('Adams', 'L-BFGS') -- compile
- model.train(epochs, callbacks) – train the NNs
- model.predict(…)

$$-\Delta u = 1.$$

(Forward problem)

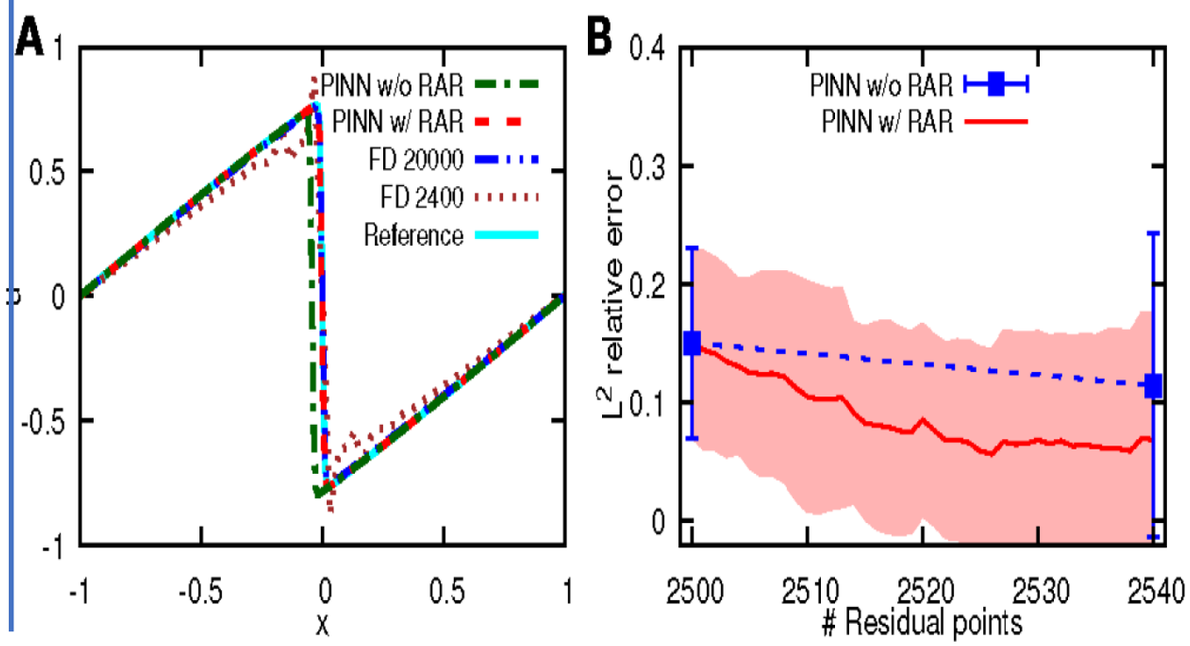
$$\partial_t C_i = D\partial_x^2 C_i - k_f C_A C_B^2,$$

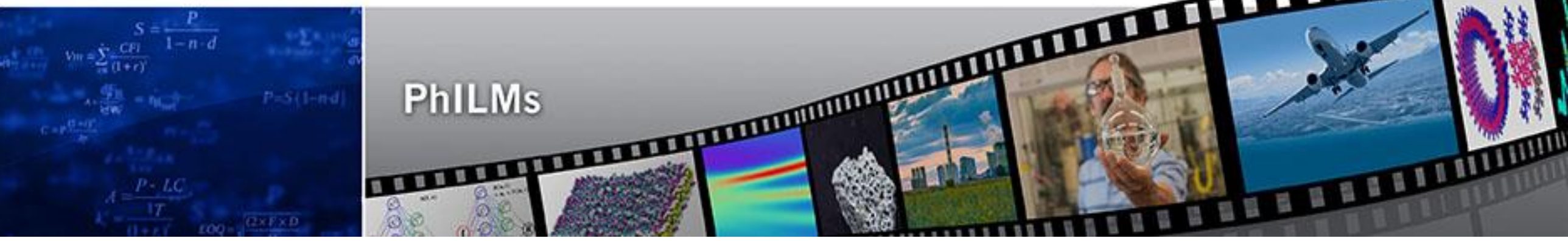
$$i = A, B.$$ (Inverse problem)



$$\partial_t u + uu_x = \nu\partial_x^2 u.$$

(Residual-based adaptive refinement (RAR))





https://www.pnnl.gov/computing/philms/

PhILMs: Collaboratory on Mathematics and Physics-Informed Learning Machines For Multiscale and Multiphysics Problems

Center Director: George Em Karniadakis
PNNL & Division of Applied Mathematics, Brown University

The **CRUNCH** group: **Home of "Math + Machine Learning + X"**
https://www.brown.edu/research/projects/crunch/home