End-to-End AI for Science Bootcamp

Dr. CK Lee, Johnson Sun, Jay Chen | NCHC x NVIDIA Joint Lab

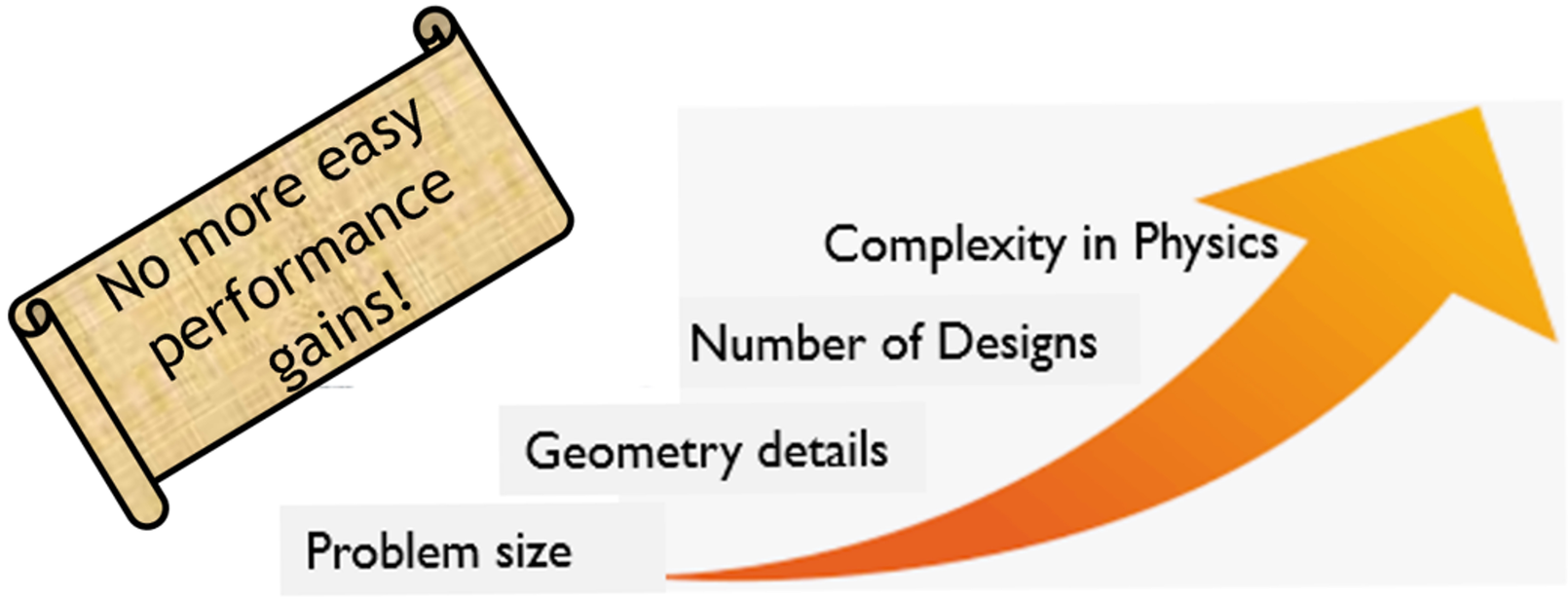
Taiwan NVAITC Solution Architect Team, June 2024.

Saturating Performance in Traditional HPC

Simulations are getting larger and more complex

Traditional solution methods are:

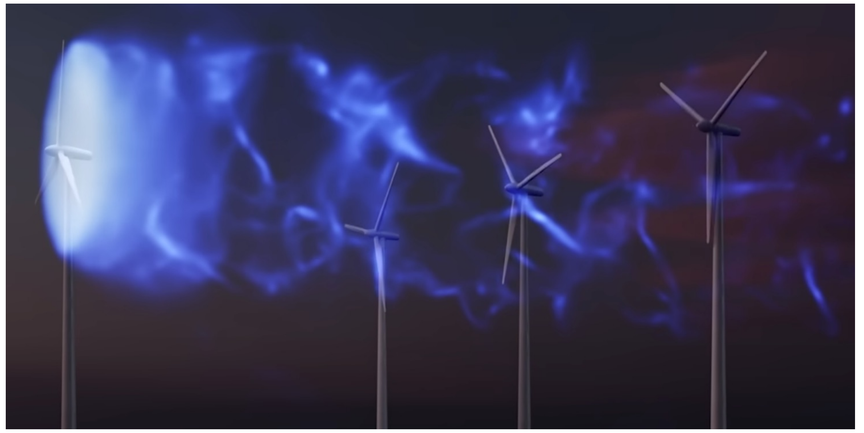
- Computationally Expensive
- Plagued by Domain Discretization Techniques
- Not suitable for Data-assimilation or Inverse problems



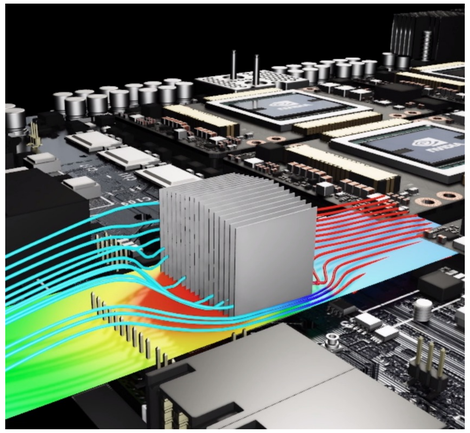
NVIDIA Modulus

NVIDIA Modulus is a neural network framework that blends the power of physics in form of governing partial differential equations (PDEs) with data to build high-fidelity, parameterized surrogate models with near-real-time latency. It offers:

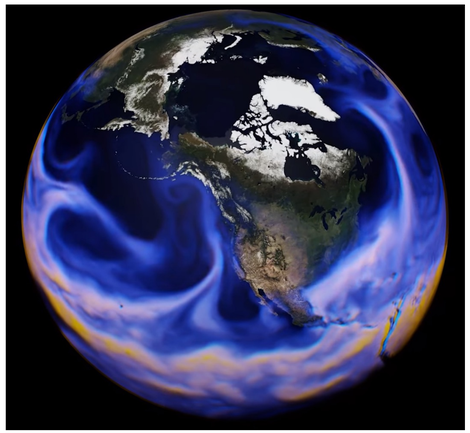
- **AI Toolkit**
  - Offers building blocks for developing physics-ML surrogate models
- **Scalable Performance**
  - Solves larger problems faster by scaling from single GPU to multi-node implementation
- **Near-Real-Time Inference**
  - Provides parameterized system representation that solves for multiple scenarios in near real time, trains once offline to infer in real time repeatedly
- **Easy Adoptability**
  - Includes APIs for domain experts to work at a higher level of abstraction. Extensible to new applications with reference applications serving as starting points



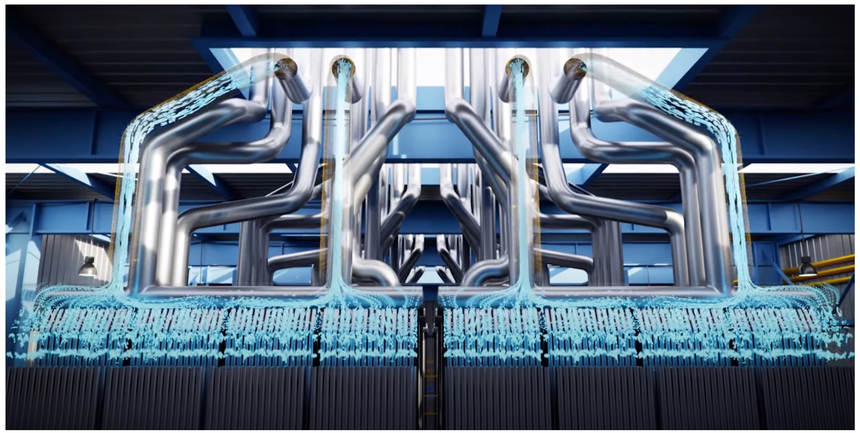
Wind Farm Super Resolution



FPGA Heatsink Design Optimization



Extreme Weather Prediction

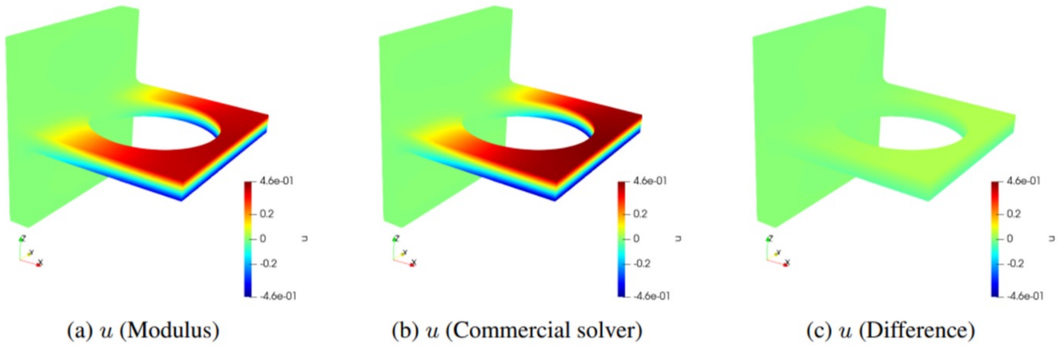


Industrial Digital Twin

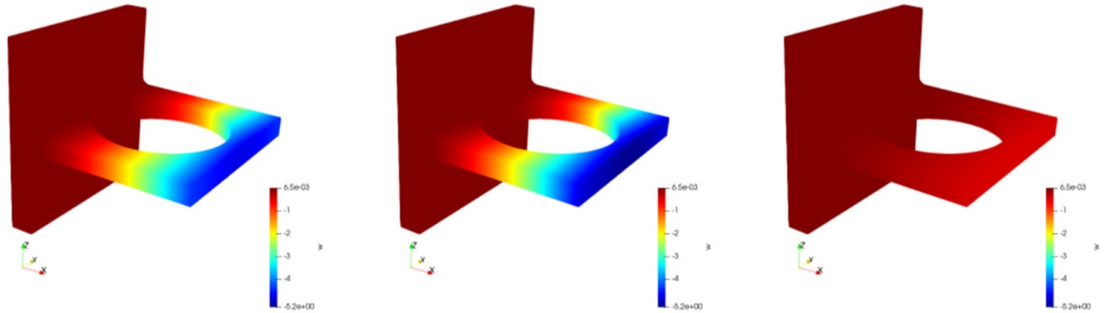
What is Modulus?

Modulus is a PDE solver

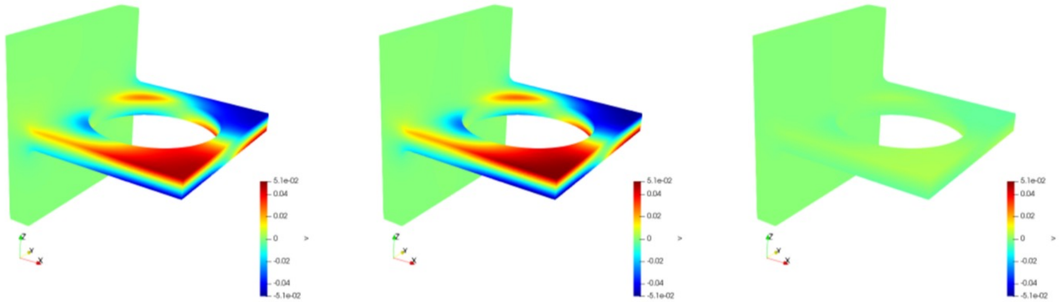
- Like traditional solvers such as Finite Element, Finite Difference, Finite Volume, and Spectral solvers, Modulus can solve PDEs



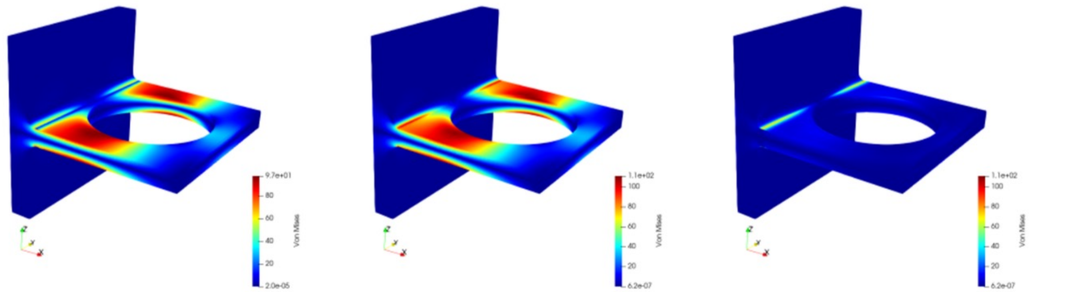
(a) u (Modulus) (b) u (Commercial solver) (c) u (Difference)



(g) w (Modulus) (h) w (Commercial Solver) (i) w (Difference)



(d) v (Modulus) (e) v (Commercial solver) (f) v (Difference)



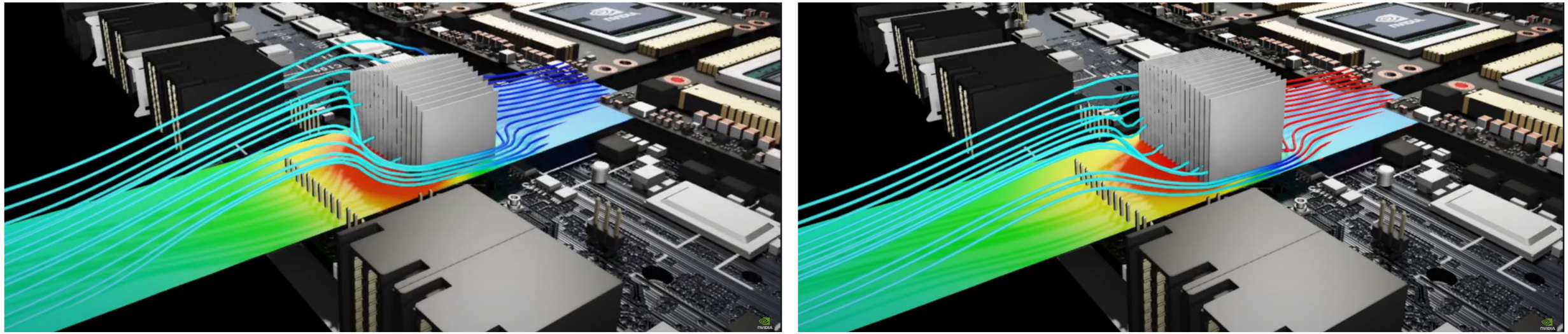
(j) σ_{VM} (Modulus) (k) σ_{VM} (Commercial Solver) (l) σ_{VM} (Difference)

A comparison between Modulus and commercial solver results for a bracket deflection example. Linear elasticity equations are solved here.

What is Modulus?

Modulus is a tool for efficient design optimization for multi-physics problems

- With Modulus, professionals in Manufacturing and Product Development can explore different configurations and scenarios of a model, in near-real time by, changing its parameters, allowing them to gain deeper insights about the system or product, and to perform efficient design optimization of their products.

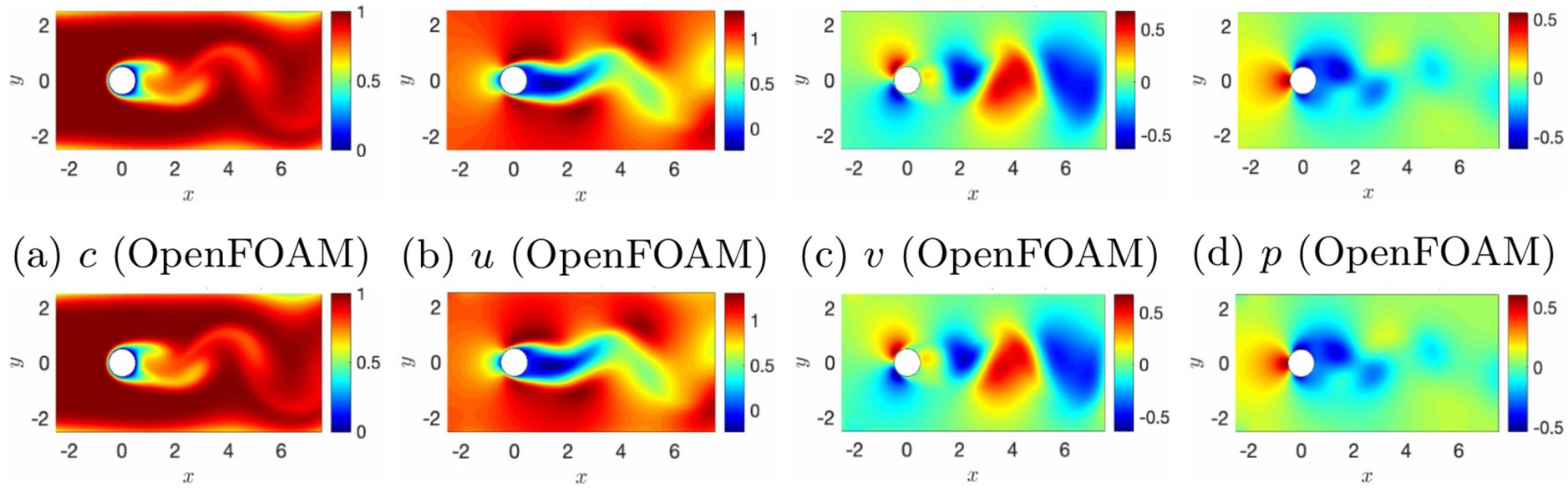


Efficient design space exploration of the heatsink of a Field-Programmable Gate Array (FPGA) using Modulus.

What is Modulus?

Modulus is a solver for inverse problems

- Many applications in science and engineering involve inferring unknown system characteristics given measured data from sensors or imaging.
- By combining data and physics, Modulus can effectively solve inverse problems.



A comparison between Modulus and OpenFOAM results for the flow velocity, pressure and passive scalar concentration fields. Modulus has inferred the velocity and pressure fields using scattered data from passive scalar concentration.

What is Modulus?

Modulus is a tool for developing digital twins

- A digital twin is a virtual representation (a true-to-reality simulation of physics) of a real-world physical asset or system, which is continuously updated via stream of data.
- Digital twin predicts the future state of the real-world system under varying conditions.

What is Modulus?

Modulus is a tool for developing data-driven solutions to engineering problems

- Modulus contains a variety of APIs for developing data-driven machine learning solutions to challenging engineering systems, including:
  - Data-driven modeling of physical systems
  - Super resolution of low-fidelity results computed by traditional solvers

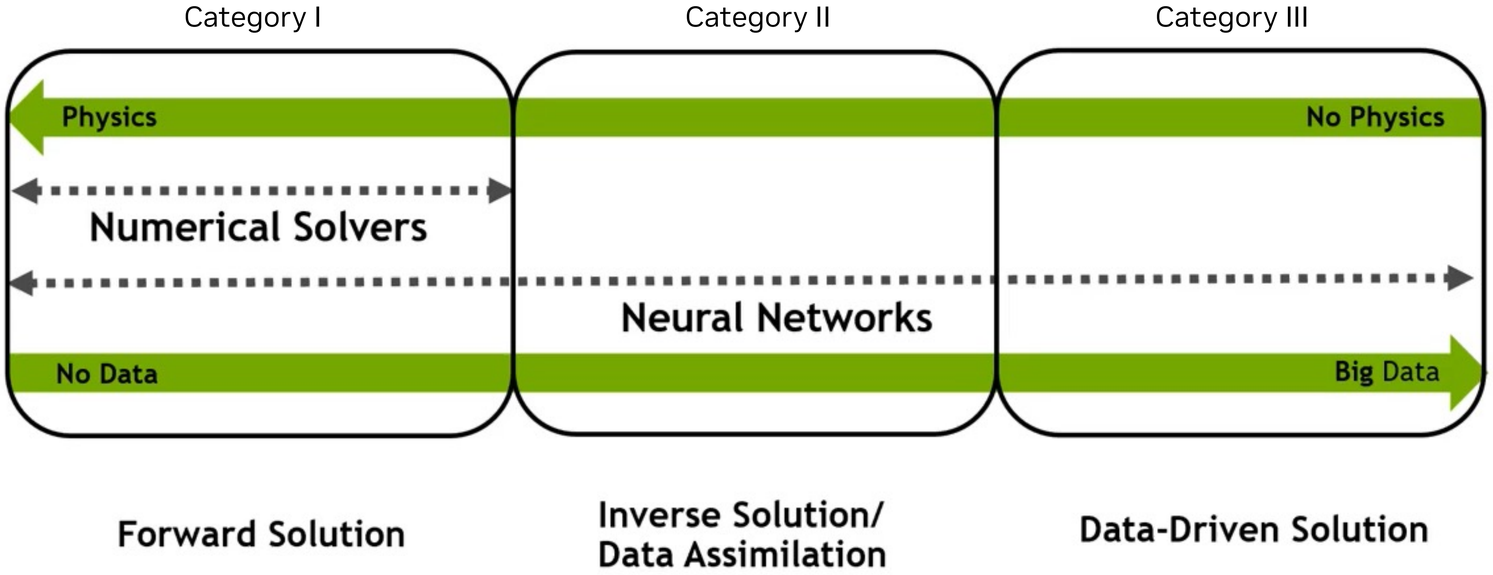
Super-resolution of flow in a wind farm using Modulus

What is Modulus?

Putting it all together

- Modulus is a PDE solver (category I)
- Modulus is a tool for efficient design optimization & design space exploration (category I)
- Modulus is a solver for inverse problems (category II)
- Modulus is a tool for developing digital twins (category II)
- Modulus is a tool for developing AI solutions to engineering problems (category III)

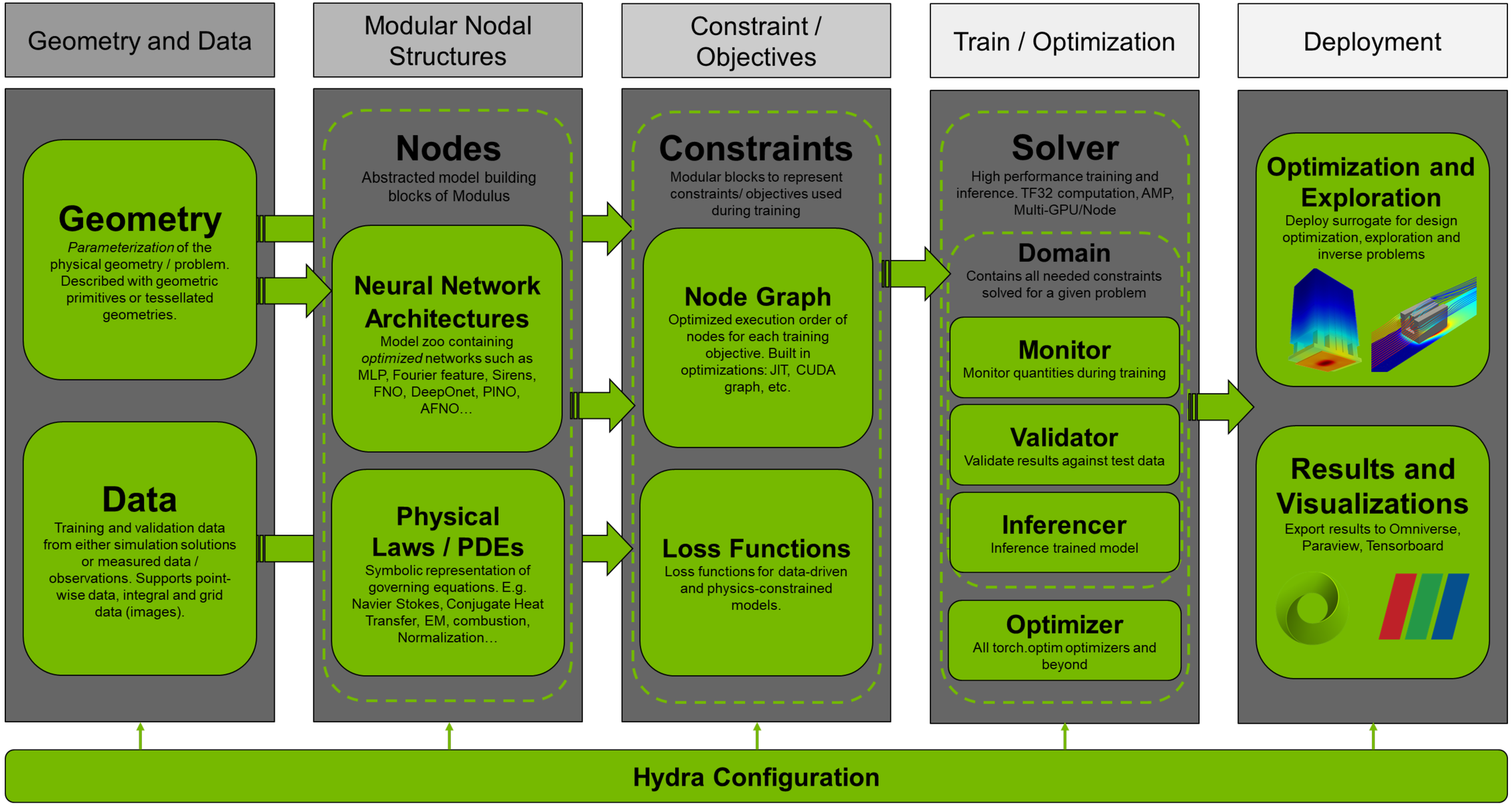
These are all done by developing deep neural network models in Modulus that are physics-informed and/or data-informed.



Agenda

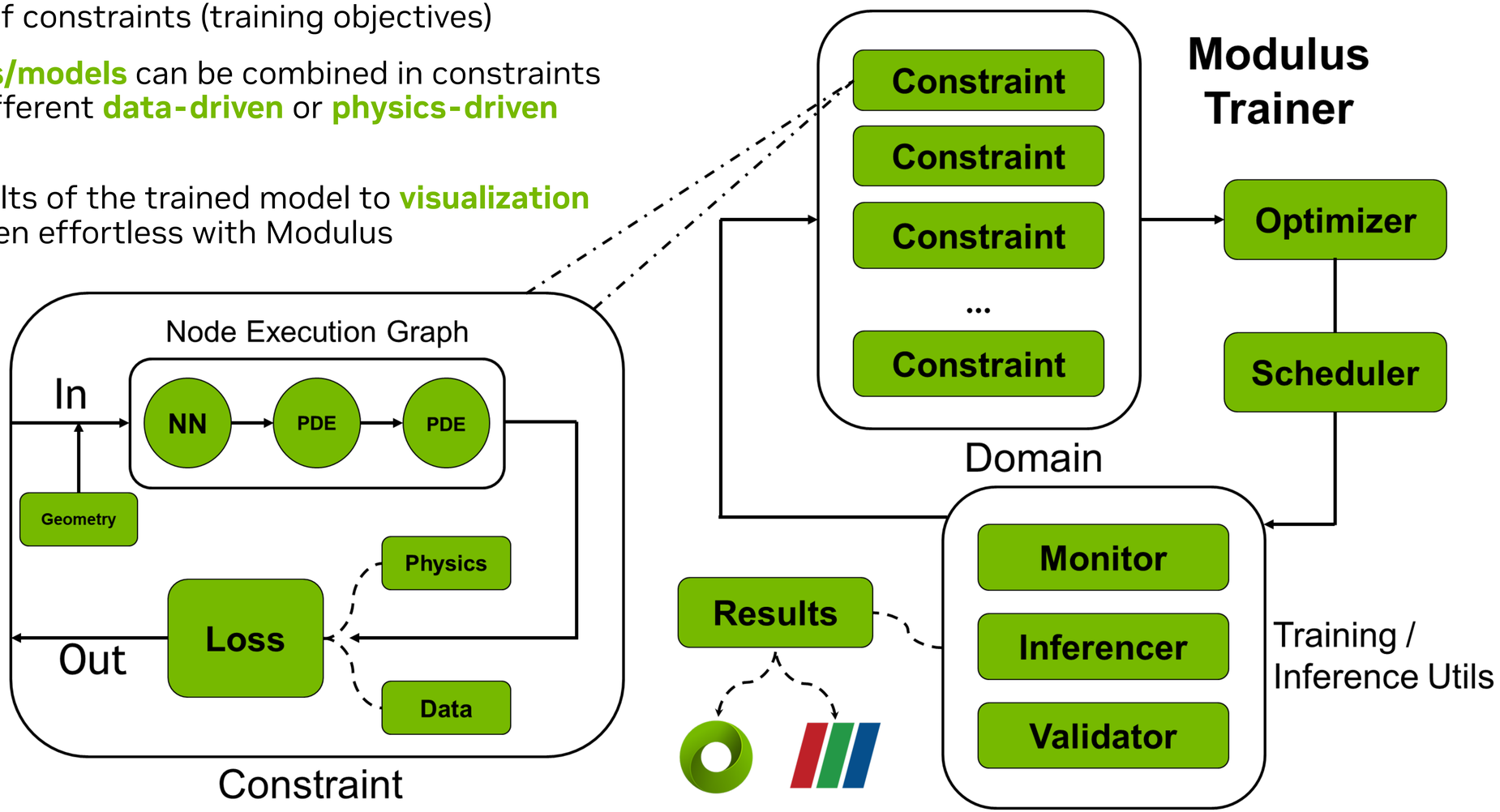
- What is Modulus?
- Modulus Architecture, Training
- Physics informed neural networks in Modulus
- Data informed neural networks in Modulus
- Modulus other features and advancements
- Introduction to Omniverse
- Hands-on with Omniverse

Modulus Architecture



Modulus Training

- Modulus allows for **complex problems** to be described through sets of constraints (training objectives)
- Multiple **nodes/models** can be combined in constraints for learning different **data-driven** or **physics-driven** loss functions
- Exporting results of the trained model to **visualization software** is then effortless with Modulus



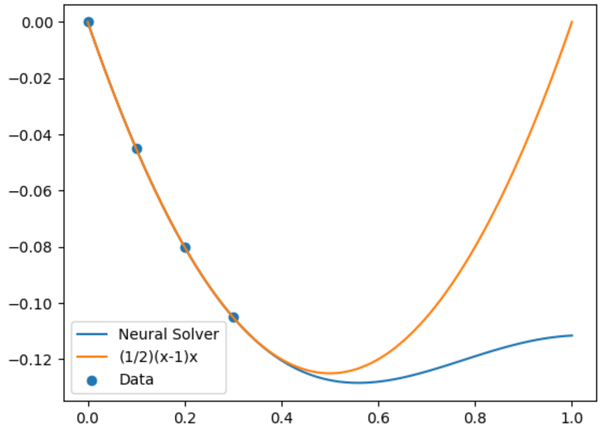
Physics Informed Neural Networks (PINNs) in Modulus

A problem with NNs and the promise of PINNs

Data Only

$$L_{data} = \sum_{x_i \in data} (u_{net}(x_i) - u_{true}(x_i))^2$$

$$L_{total} = L_{data}$$

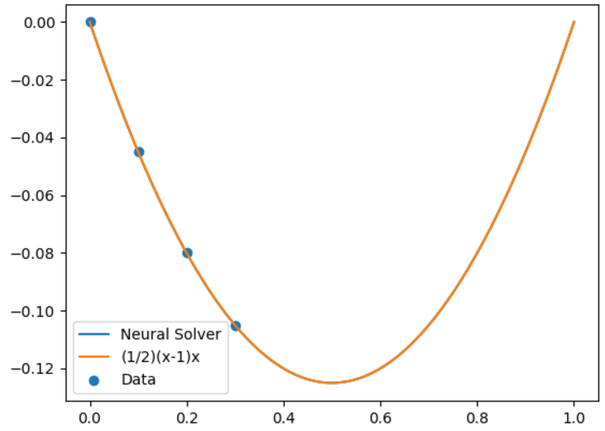


Data + Physics

$$\frac{\delta^2 u}{\delta x^2}(x) = 1$$

$$L_{physics} = \sum_{x_j \in domain} \left(\frac{\delta^2 u_{net}}{\delta x^2}(x_i) - f(x_i) \right)^2$$

$$L_{total} = L_{data} + L_{physics}$$

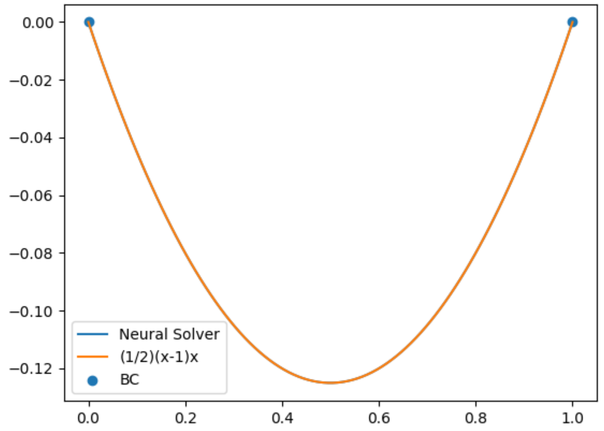


Physics only

$$\frac{\delta^2 u}{\delta x^2}(x) = 1 \qquad u(0) = u(1) = 0$$

$$L_{physics} = L_{residual} + L_{BC}$$

$$L_{total} = L_{physics}$$



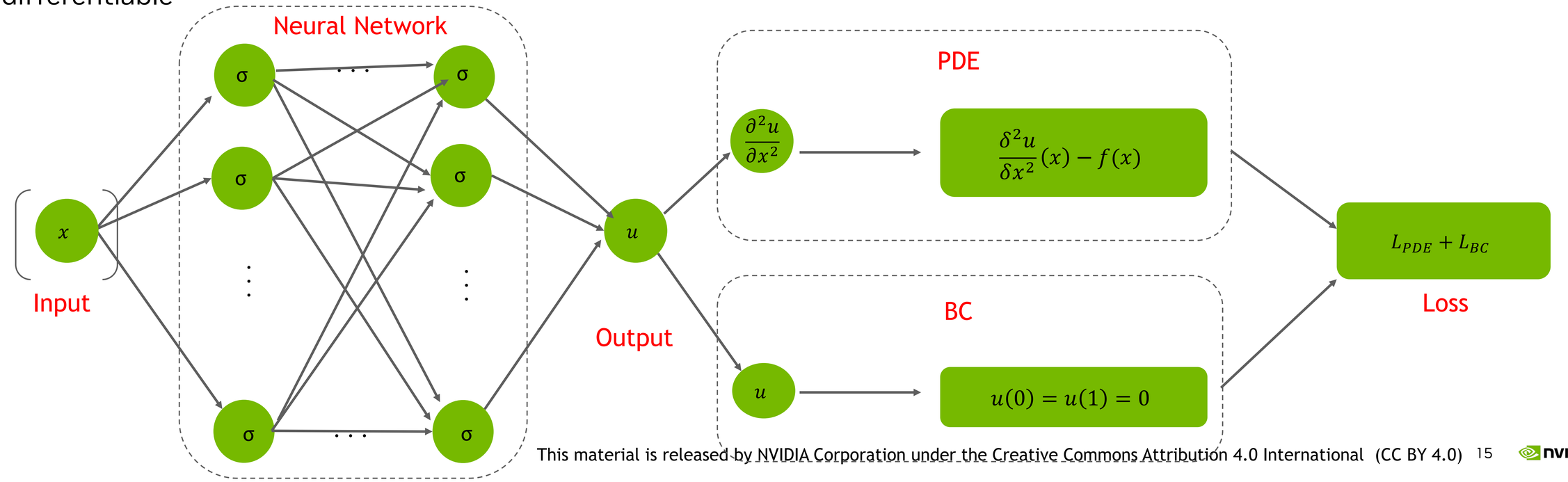
PINNs Theory: Neural Network Solver

- Goal: Train a neural network to satisfy the boundary conditions and differential equations by constructing an appropriate loss function
  - Consider an example problem:

$$\mathbf{P}: \begin{cases} \frac{\delta^2 u_{net}}{\delta x^2}(x) = f(x) \\ u_{net}(0) = u_{net}(1) = 0 \end{cases}$$

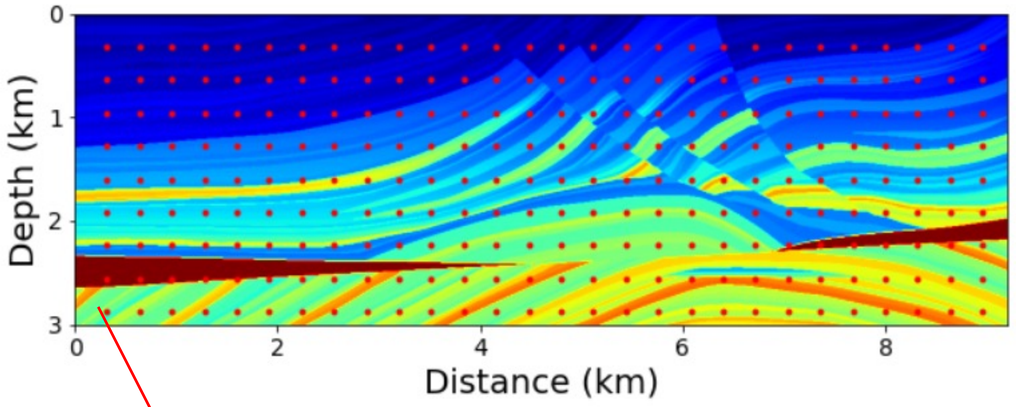
Forward Problem

- We construct a neural network $\boldsymbol{u_{net}(x)}$ which has a single value input $x \in \mathbb{R}$ and single value output $u_{net}(x) \in \mathbb{R}$.
- We assume the neural network is **infinitely differentiable** $u_{net} \in C^{\infty}$ - Use activation functions that are infinitely differentiable



AN EXAMPLE: SEISMIC PROBLEMS

Point cloud representation (not a mesh)



PHYSICS

Neural Network

Input

Output

PDE

$\frac{\partial^2 u}{\partial t^2} - c^2\left(\frac{\partial^2 u}{\partial x^2} + \frac{\partial^2 u}{\partial z^2}\right)$

IC

$u(x,z,t) - u_D$

BC

$\frac{\partial u(x,z,t)}{\partial n} - u_N$

$L_{PDE} + L_{IC} + L_{BC}$

Loss

NVIDIA

PINNs Theory: Neural Network Solver

Loss formulation

- Construct the loss function. We can compute the second order derivatives $\left(\frac{\delta^2 u_{net}}{\delta x^2}(x)\right)$ using Automatic differentiation

$$L_{BC} = u_{net}(0)^2 + u_{net}(1)^2$$

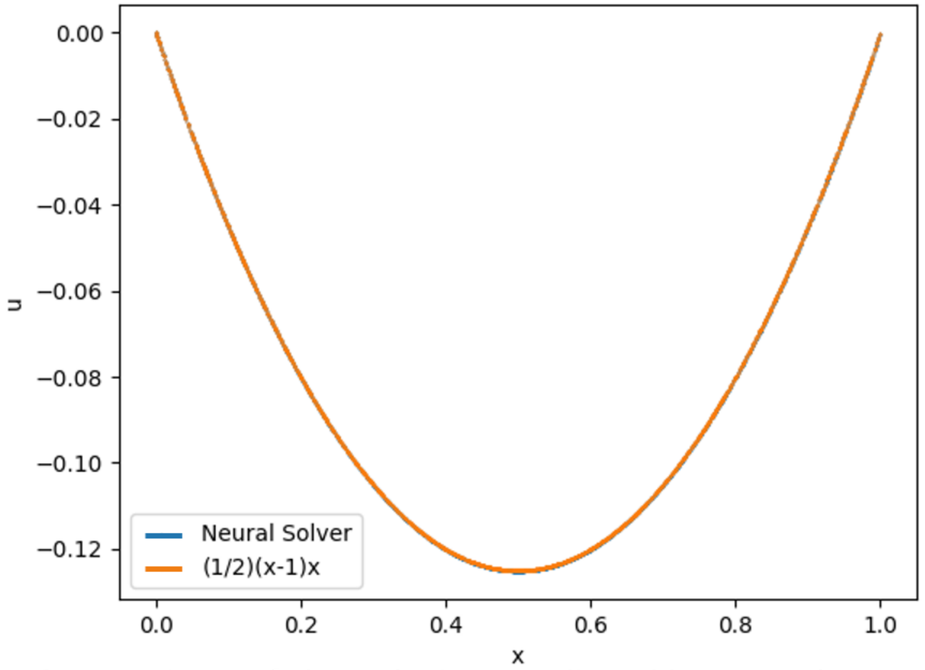
$$L_{Residual} = \int_0^1 \left(\frac{\delta^2 u_{net}}{\delta x^2}(x) - f(x)\right)^2 dx \approx \left(\int_0^1 dx\right)\frac{1}{N}\sum_{i=0}^{N}\left(\frac{\delta^2 u_{net}}{\delta x^2}(x_i) - f(x_i)\right)^2$$

- Where x_i are a batch of points in the interior $x_i \in (0, 1)$. Total loss becomes $L = L_{BC} + L_{Residual}$
- Minimize the loss using optimizers like Adam

PINNs Theory: Neural Network Solver

Results

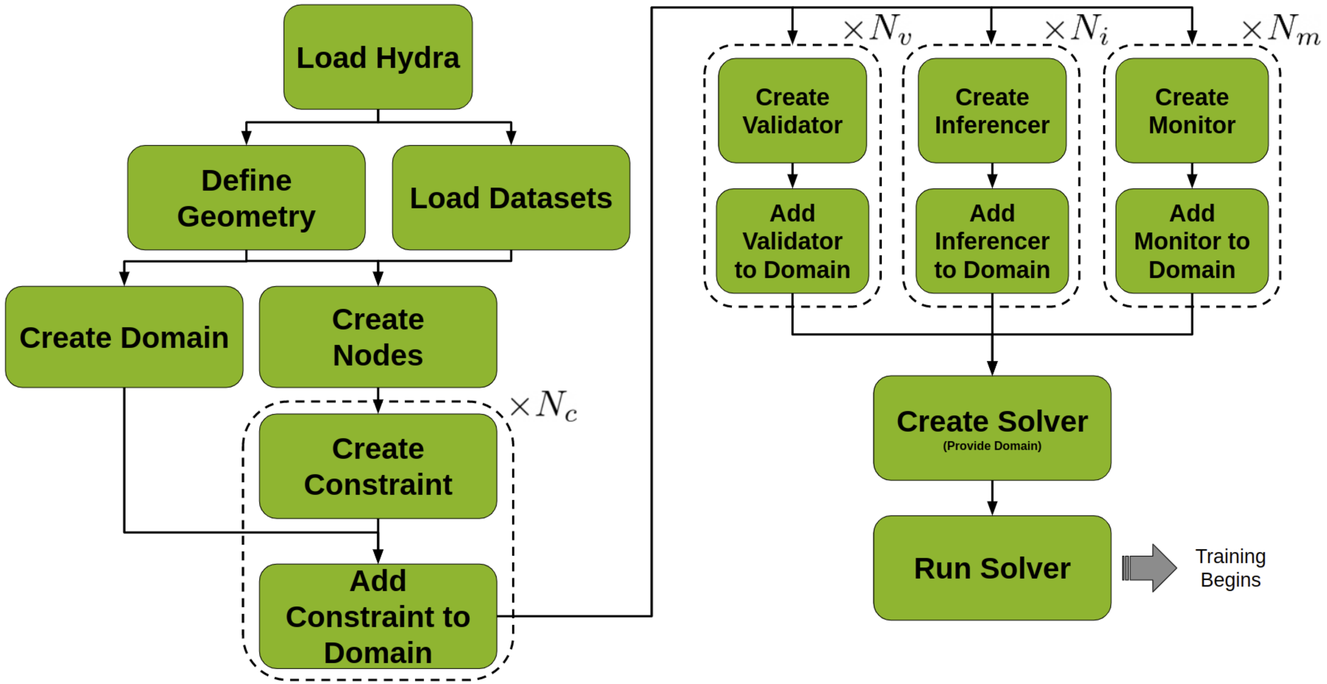
- For $f(x) = 1$, the true solution is $\frac{1}{2}(x-1)x$. After sufficient training we have,



Comparison of the solution predicted by Neural Network with the analytical solution

Modulus: Anatomy of a project

Overview

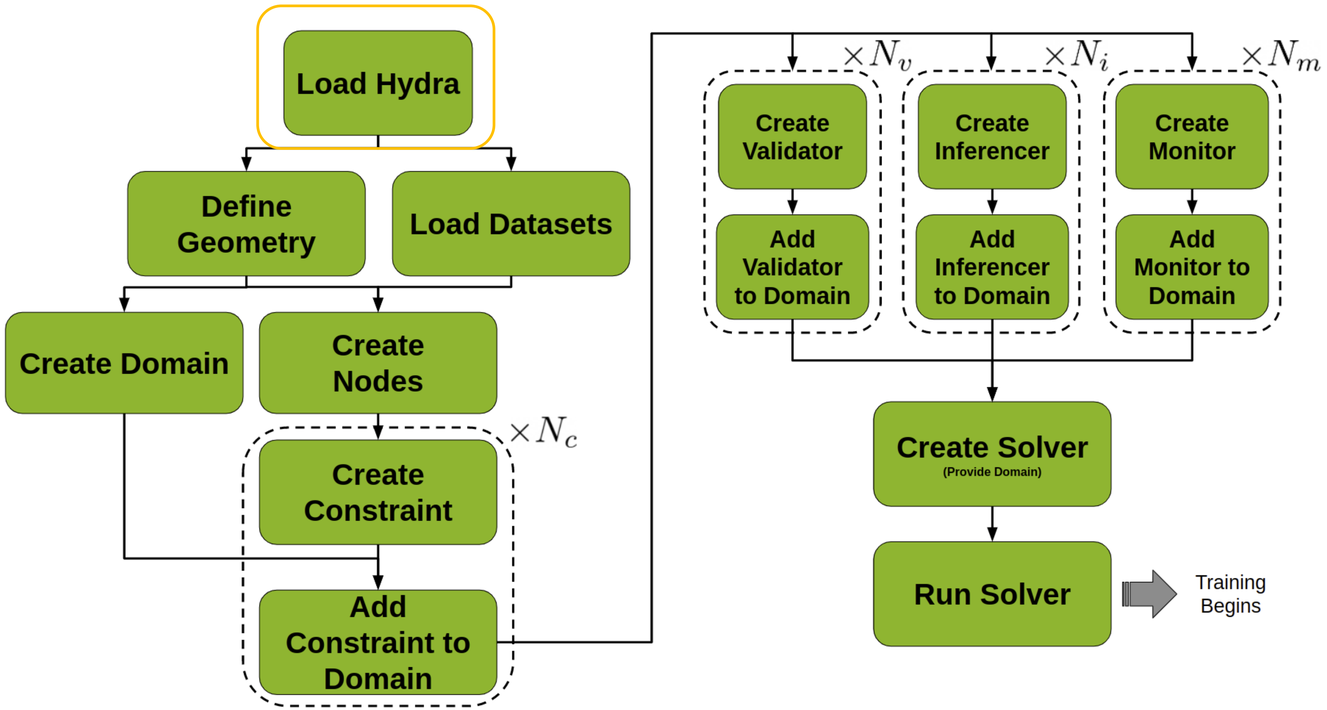


Modulus works by:

- Writing models which include at least one adaptable function (a NN)
- Writing objective functions as a combination of these models
- Describing the geometry/dataset where the models should be evaluated
- Minimizing the objective functions by using the provided data, by sampling the geometry, or both
- Running the models to obtain the desired effect

Modulus: Anatomy of a project

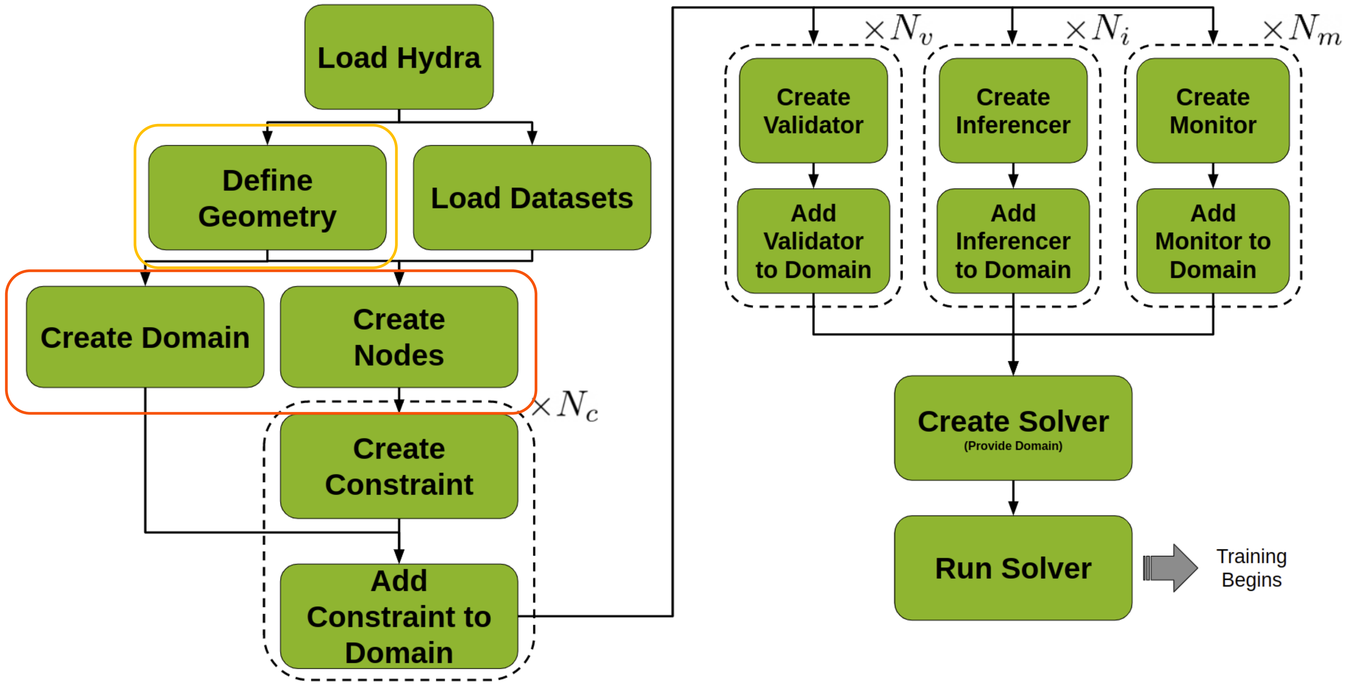
Load Hydra




```yaml
defaults :
  - modulus_default
  - scheduler: tf_exponential_lr
  - optimizer: adam
  - loss: sum
  - _self_

scheduler:
  decay_rate: 0.95
  decay_steps: 200

save_filetypes : "vtk,npz"

training:
  rec_results_freq : 1000
  rec_constraint_freq: 1000
  max_steps : 5000
```

Modulus: Anatomy of a project

Create geometry, domain and nodes

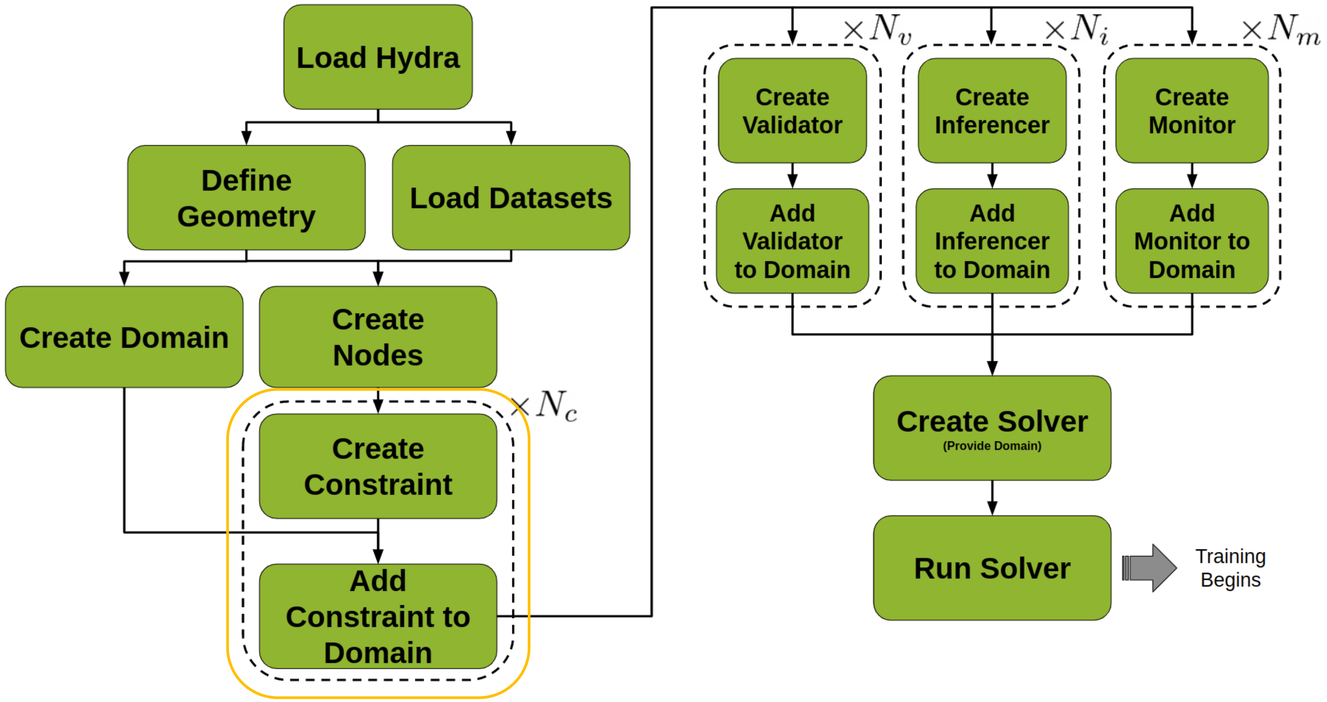



```python
@modulus.main(config_path="conf", config_name="config")
def run(cfg: ModulusConfig) -> None:

    # make geometry
    x = Symbol("x")
    geo = Line1D(0, 1)
```

```python
    # make list of nodes to unroll graph on
    eq = CustomPDE(f=1.0)
    u_net = FullyConnectedArch(
        input_keys=[Key("x")],
        output_keys=[Key("u")],
        nr_layers=3,
        layer_size=32
    )

    nodes = eq.make_nodes() + [u_net.make_node(name="u_network")]

    # make domain
    domain = Domain()
```

Modulus: Anatomy of a project

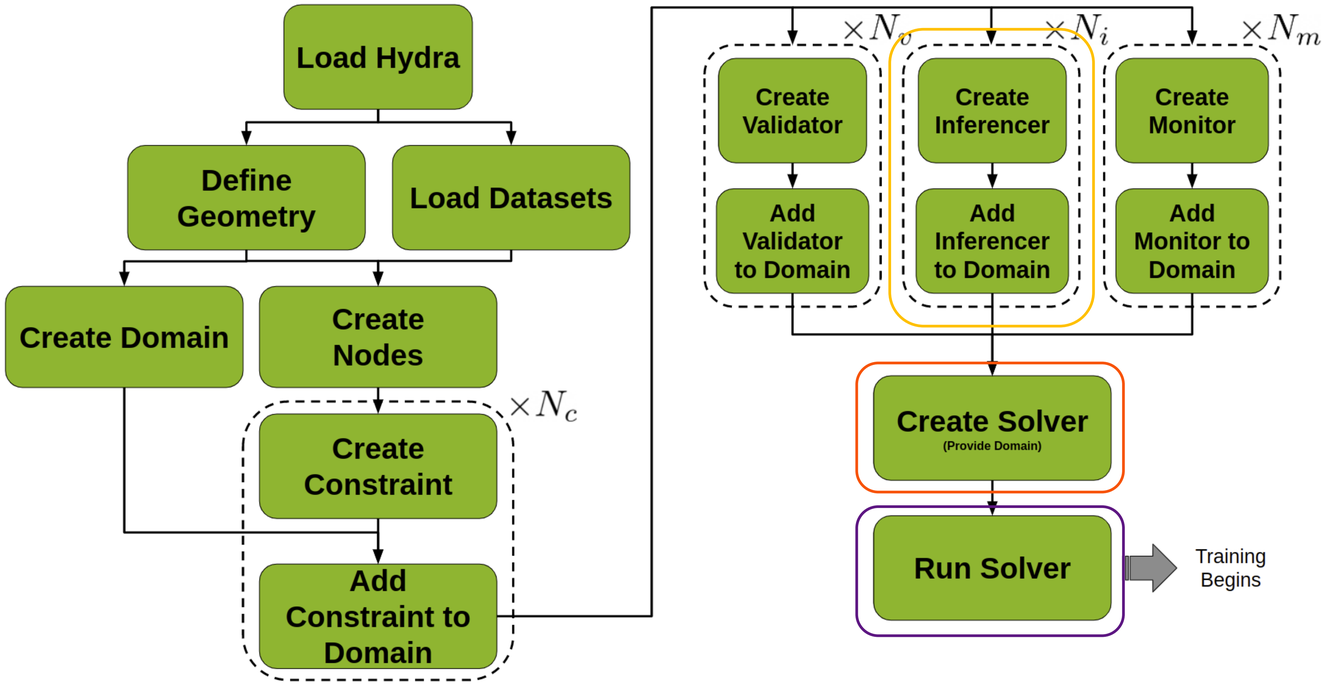
Add constraints




```python
# add constraints to solver

# bcs
bc = PointwiseBoundaryConstraint(
    nodes=nodes,
    geometry=geo,
    outvar={"u": 0},
    batch_size=2,
)
domain.add_constraint(bc, "bc")

# interior
interior = PointwiseInteriorConstraint(
    nodes=nodes,
    geometry=geo,
    outvar={"custom_pde": 0},
    batch_size=100,
    bounds={x: (0, 1)},
)
domain.add_constraint(interior, "interior")
```

Modulus: Anatomy of a project

Create utils to visualize the results and run the solver




```
    # add inferencer
    inference = PointwiseInferencer(
        nodes=nodes,
        invar={"x": np.linspace(0, 1.0, 100).reshape(-1,1)},
        output_names=["u"],
    )
    domain.add_inferencer(inference, "inf_data")
```

```
    # make solver
    slv = Solver(cfg, domain)

    # start solver
    slv.solve()

if __name__ == "__main__":
    run()
```

```
python <script_name>.py
mpirun -np <#GPU> <script_name>.py
```

Solving Parameterized Problems

Problem definition

- Consider the parameterized version of the same problem as before. Suppose we want to determine how the solution changes as we move the position on the boundary condition $u(l) = 0$
- Parameterize the position by variable $l \in [1, 2]$ and the problem now becomes:

$$\mathbf{P}: \begin{cases} \dfrac{\delta^2 u}{\delta x^2}(x) = f(x) \\ u(0) = u(l) = 0 \end{cases}$$

- This time, we construct a neural network $u_{net}(x, l)$ which has x and l as input and single value output $u_{net}(x, l) \in \mathbb{R}$.
- The losses become

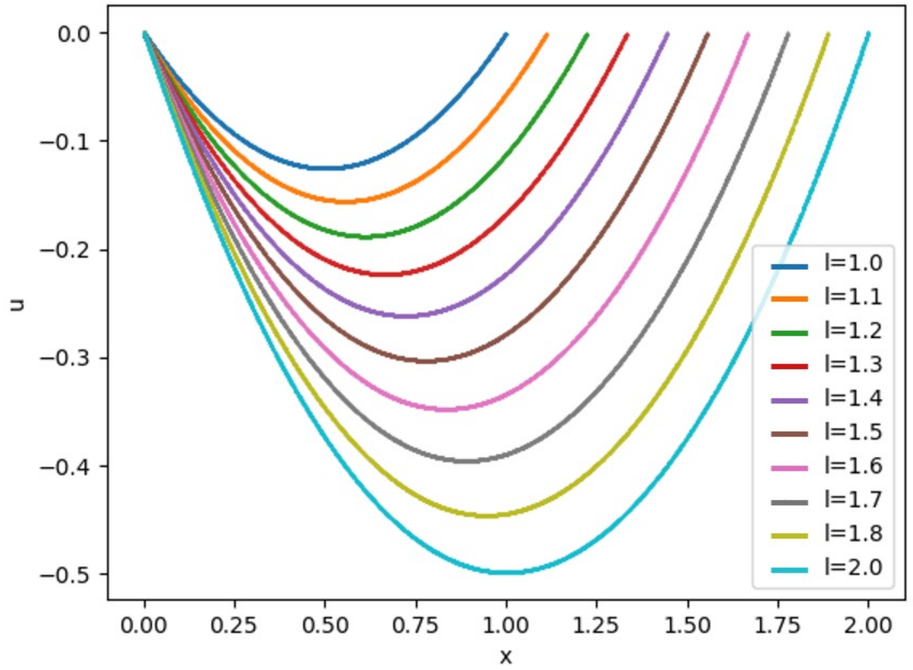
$$L_{Residual} = \int_1^2 \int_0^1 \left(\frac{\delta^2 u_{net}}{\delta x^2}(x) - f(x) \right)^2 dx dl \approx \left(\int_1^2 \int_0^1 dx\, dl \right) \frac{1}{N} \sum_{i=0}^{N} \left(\frac{\delta^2 u_{net}}{\delta x^2}(x_i, l_i) - f(x_i) \right)^2$$

$$L_{BC} = \int_1^2 \big(u_{net}(0, l)\big)^2 + \big(u_{net}(l, l)\big)^2 dl \approx \left(\int_1^2 dl \right) \frac{1}{N} \sum_{i=0}^{N} (u_{net}(0, l_i))^2 + (u_{net}(l_i, l_i))^2$$

Solving Parameterized Problems

Results

- For $f(x) = 1$, for different values of l we have different solutions



Solution to the parametric problem

Solving Inverse Problems

Problem definition

- For inverse problems, we start with a set of observations and then calculate the causal factors that produced them
- For example, suppose we are given the solution $u_{true}(x)$ at 100 random points between 0 and 1 and we want to determine the $f(x)$ that is causing it
- Train two networks $u_{net}(x)$ and $f_{net}(x)$ to approximate $u(x)$ and $f(x)$

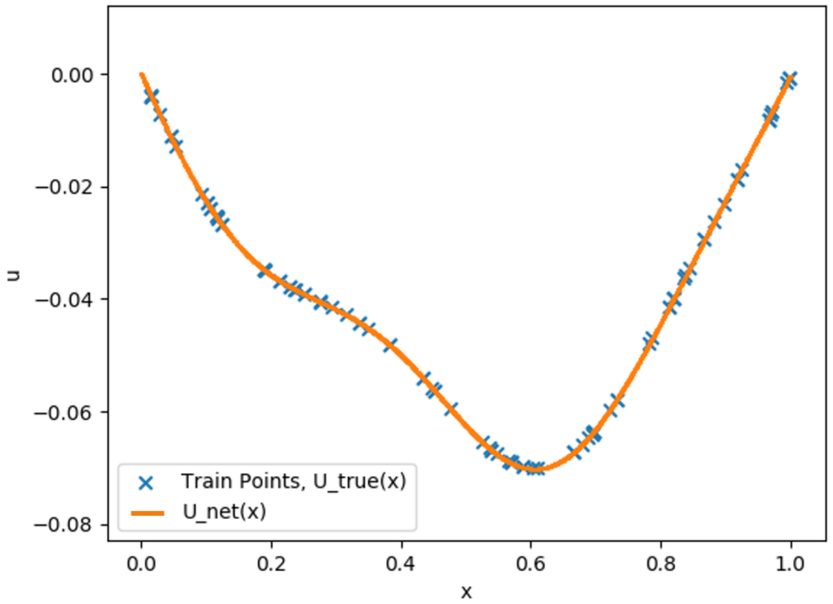
$$L_{Residual} \approx \left(\int_0^1 dx\right)\frac{1}{N}\sum_{i=0}^{N}\left(\frac{\delta^2 u_{net}}{\delta x^2}(x_i) - f(x_i)\right)^2$$

$$L_{Data} = \frac{1}{100}\sum_{i=0}^{100}\left(u_{net}(x_i) - u_{true}(x_i)\right)^2$$

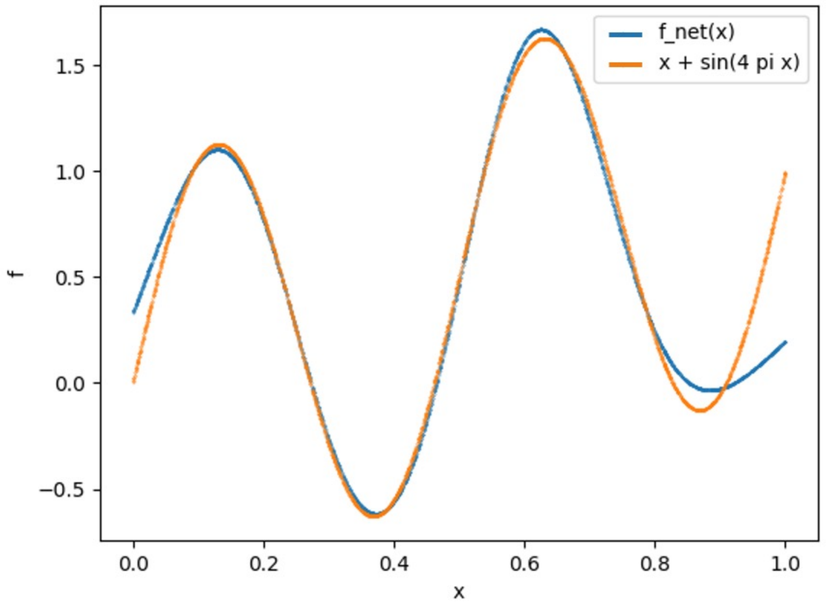
Solving Inverse Problems

Results

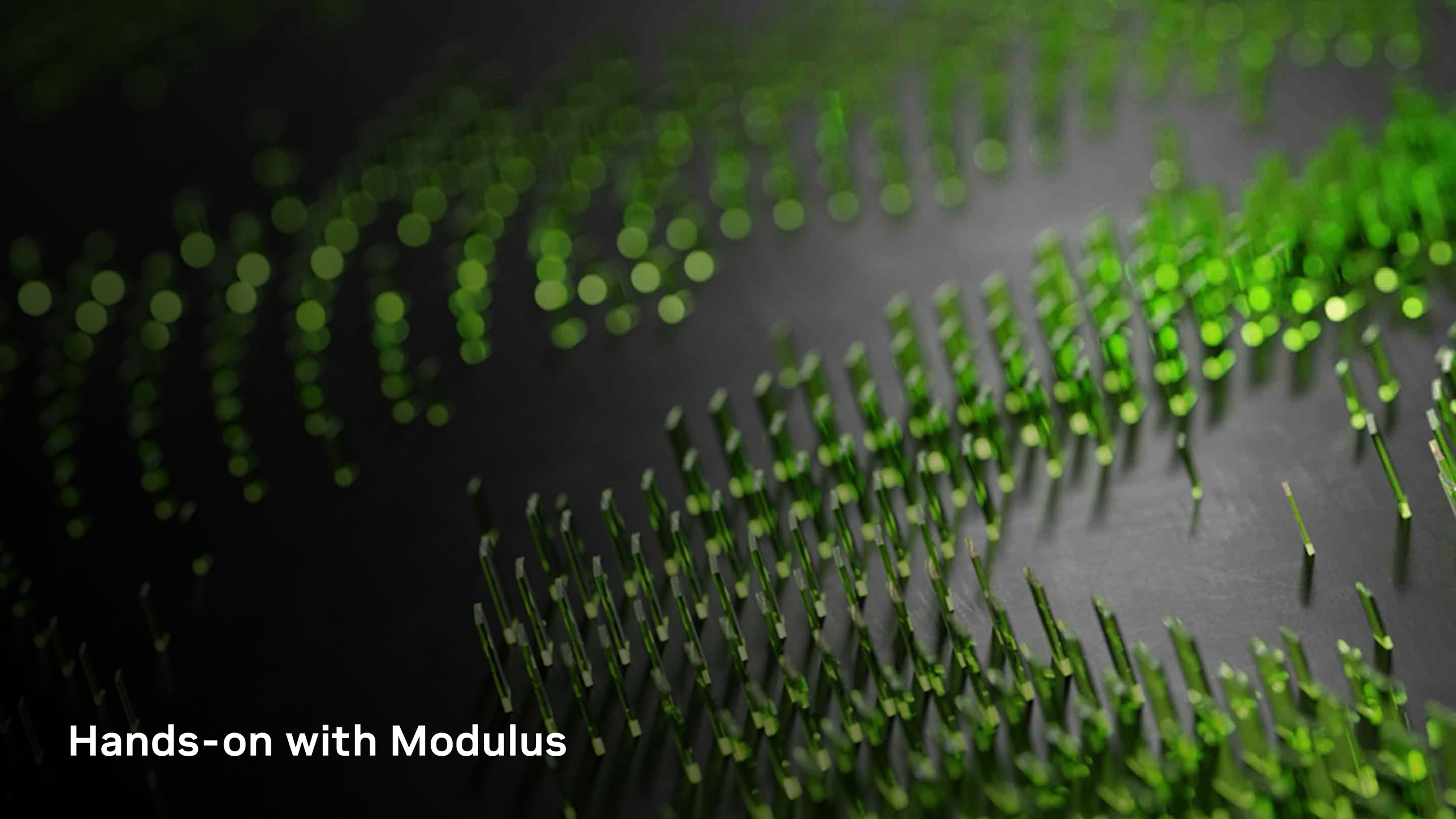
- For $u_{true}(x) = \frac{1}{48}\left(8x(-1+x^2) - \frac{3\sin(4\pi x)}{\pi^2}\right)$ the solution for $f(x)$ is $x + \sin(4\pi x)$



Comparison of $u\_net\ (x)$ and train points from u_true



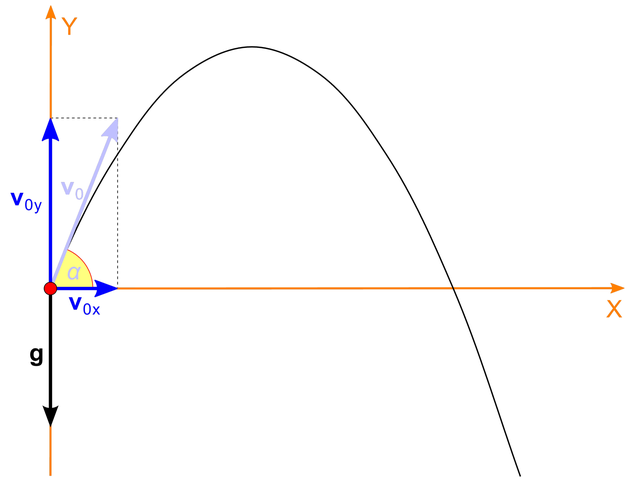
Comparison of the true solution for $f(x)$ and the $f_{net}(x)$ inverted out



Hands-on with Modulus

Projectile motion

Problem description



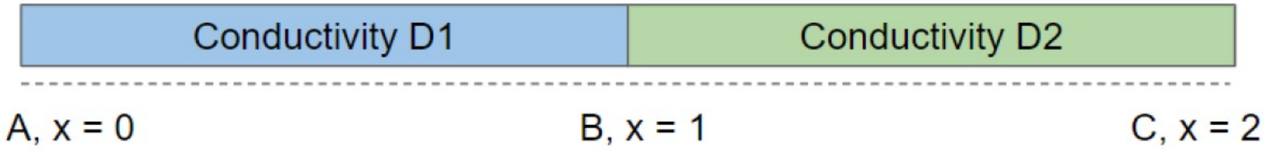
- Solving projectile motion in the case of absence of air-resistance on the surface on Earth.
- Equations: Equations of motions are as follows.

$$\frac{d^2}{dt^2}S_x = 0 \qquad \frac{d^2}{dt^2}S_y = -g$$

- Let us solve these two equations with the ball starting at origin (0,0)

1D diffusion

Problem description



- Composite bar with material of conductivity $D_1 = 10$ for $x \in (0,1)$ and $D_2 = 0.1$ for $x \in (1,2)$. Point A and C are maintained at temperatures of 0 and 100 respectively
- Equations: Diffusion equation in $1D$

$$\frac{d}{dx}\left(D_1 \frac{dU_1}{dx}\right) = 0 \qquad \text{When } 0 < x < 1$$

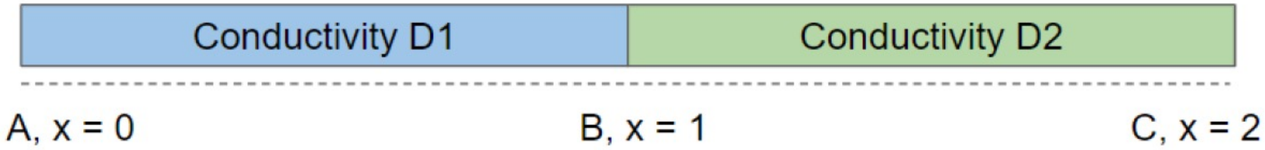
$$\frac{d}{dx}\left(D_2 \frac{dU_2}{dx}\right) = 0 \qquad \text{When } 1 < x < 2$$

- Flux and field continuity at interface (x=1)

$$\left(D_1 \frac{dU_1}{dx}\right) = \left(D_2 \frac{dU_2}{dx}\right)$$
$$U_1 = U_2$$

Parameterized 1D diffusion

Problem description



- Composite bar with material of conductivity D_1 for $x \in (0,1)$ and $D_2 = 0.1$ for $x \in (1,2)$.
- Solve the problem for multiple values of D_1 in the range $(5, 25)$ in a single training
- Same boundary and interface conditions as before
- Equations: Diffusion equation in $1D$

$$\frac{d}{dx}\left(D_1 \frac{dU_1}{dx}\right) = 0 \qquad \text{When } 0 < x < 1$$

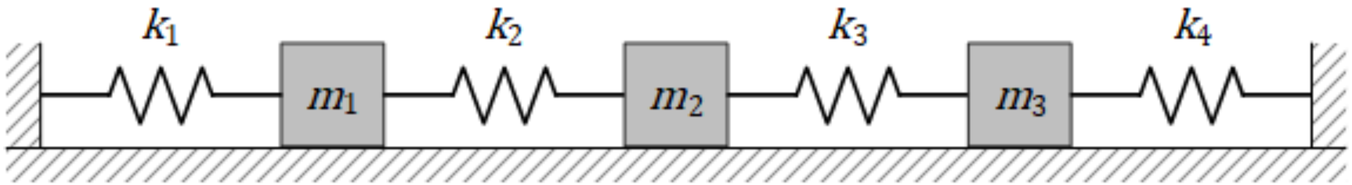
$$\frac{d}{dx}\left(D_2 \frac{dU_2}{dx}\right) = 0 \qquad \text{When } 1 < x < 2$$

- Flux and field continuity at interface (x=1)

$$\left(D_1 \frac{dU_1}{dx}\right) = \left(D_2 \frac{dU_2}{dx}\right)$$
$$U_1 = U_2$$

Optional - Coupled Spring Mass System

Problem description



- Three masses connected by four springs
- System's equations (ordinary differential equations):

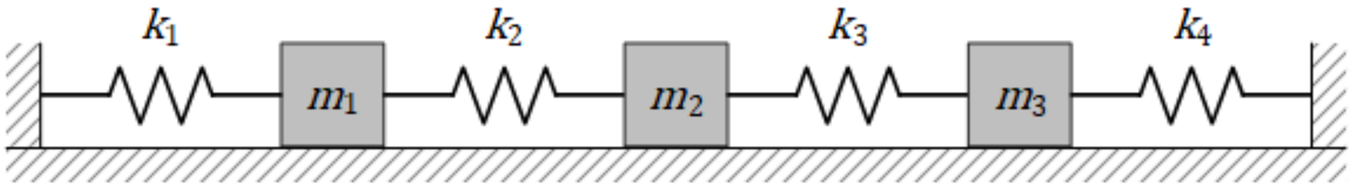
$$m_1 x_1''(t) = -k_1 x_1(t) + k_2\big(x_2(t) - x_1(t)\big)$$
$$m_2 x_2''(t) = -k_2\big(x_2(t) - x_1(t)\big) + k_3\big(x_3(t) - x_2(t)\big)$$
$$m_3 x_3''(t) = -k_3\big(x_3(t) - x_2(t)\big) - k_4 x_3(t)$$

- For given values masses, spring constants and boundary conditions

$$[m_1, m_2, m_3] = [1, 1, 1]$$
$$[k_1, k_2, k_3, k_4] = [2, 1, 1, 2]$$
$$[x_1(0), x_2(0), x_3(0)] = [1, 0, 0]$$
$$[x_1'(0), x_2'(0), x_3'(0)] = [0, 0, 0]$$

Optional - Inverse Problem – Coupled Spring Mass System

Problem description



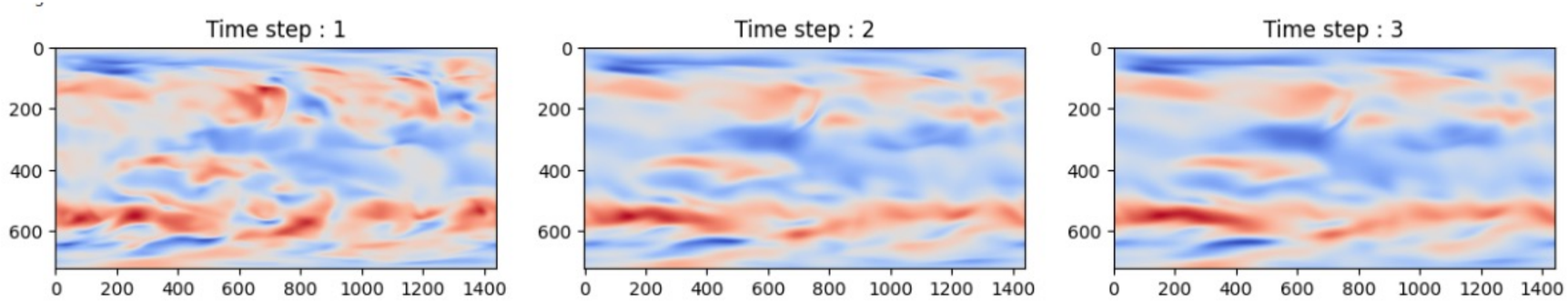
- For the same system, assume we know the analytical solution which is given by:

$$x_1(t) = \frac{1}{6}\cos(t) + \frac{1}{2}\cos\left(\sqrt{3}t\right) + \frac{1}{3}\cos(2t)\,;$$
$$x_2(t) = \frac{2}{6}\cos(t) - \frac{1}{3}\cos(2t)\;;$$
$$x_3(t) = \frac{1}{6}\cos(t) - \frac{1}{2}\cos\left(\sqrt{3}t\right) + \frac{1}{3}\cos(2t)$$

- With the above data and the values for m_2, m_3, k_1, k_2, k_3 same as before, use the neural network to find the values of m_1 and k_4

Weather prediction using Navier-Stokes

Problem description



- We aim to predict the velocities for 6-hour timesteps using the Navier-Stokes equation.
- The Navier–Stokes equations mathematically express the conservation of momentum and conservation of mass for Newtonian fluids.
- We will take a 2d projected input from the ERA5 Reanalysis dataset to be used as initial conditions for our input. The process of taking the 3D sphere and projecting it onto a 2D mesh is shown in the diagram below, the 2D mesh is of the size (1440,720)