Universität
Bremen

Center for Industrial
Mathematics (ZeTeM)

Faculty 03

Mathematics / Computer science

TorchPhysics

A Deep Learning Library for solving Differential Equations

Janek Gödeke, Tom Freudenberg
Bremen, 20.07.2023

Yesterday...

Manually

- implemented data-based & physics-informed learning of PDEs/ODEs
- sampled points within domains
- computed PDE-loss, e.g. parachute example

$$\sum_k \left\| \partial_t^2 u_\theta(t_k) - D(\partial_t u_\theta(t_k))^2 + g \right\|^2 + \dots$$

Today: TorchPhysics makes this easier!

Introduction of TorchPhysics

Initiation of TorchPhysics

- 2021: Robert Bosch GmbH got interested in PINNs
- Technical applications:
  Car parts, electronics, injection moulding, etc.

BOSCH

© factum

Initiation of TorchPhysics

- 2021: Robert Bosch GmbH got interested in PINNs
- Technical applications:
  Car parts, electronics, injection moulding, etc.

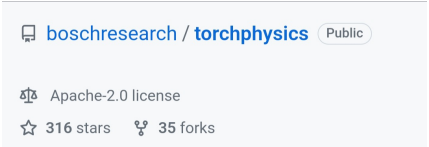
- Student project: Deep Learning library for PDEs
- Main Developers: Nick Heilenkötter & Tom Freudenberg
- TorchPhysics is under continuous extension

BOSCH

© factum

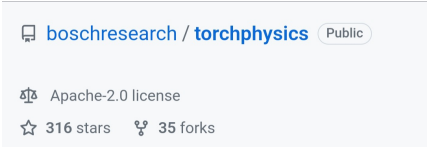
The Toolbox TORCHPHYSICS

- Library for solving PDE-related problems via (physics-informed) deep learning
- Open-Source on GitHub
- Build upon PYTORCH

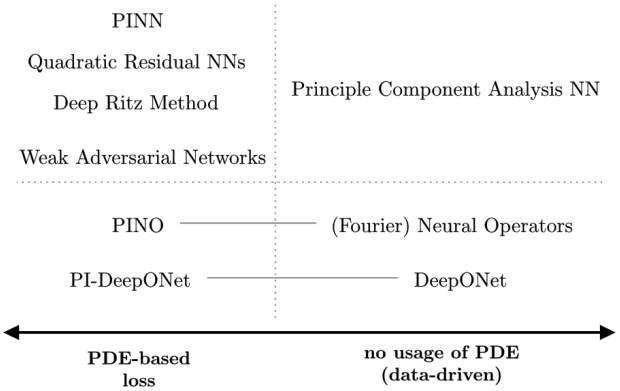


The Toolbox TORCHPHYSICS

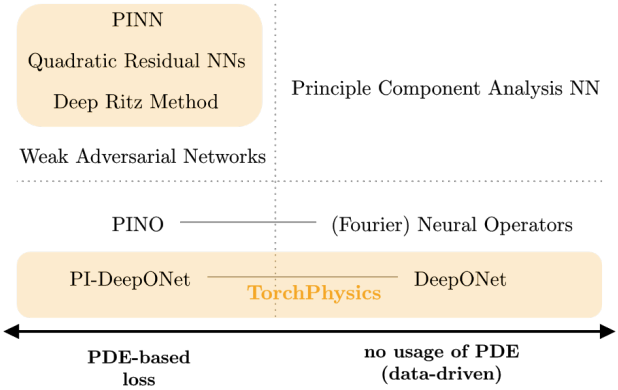
- Library for solving PDE-related problems via (physics-informed) deep learning
- Open-Source on GitHub
- Build upon PYTORCH
- TORCHPHYSICS provides:
  - Modular and extendable structure
  - Clean documentation, detailed tutorials
  - Intuitive transfer of maths into code



Overview of DL-Methods for Differential Equations



Overview of DL-Methods for Differential Equations



PINNs - Reminder of PDE-Based Loss

- Find solution $\mathbf{u} : \Omega \subset \mathbb{R}^n \to \mathbb{R}^m$ of

$$\begin{aligned} \mathcal{N}[\mathbf{u}](\mathbf{x}) &= 0, \quad \text{for } \mathbf{x} \in \Omega, \\ \mathcal{B}[\mathbf{u}](\mathbf{x}) &= 0, \quad \text{for } \mathbf{x} \in \partial\Omega. \end{aligned}$$

- E.g. $\Omega = [0,1] \times [0,1], \mathbf{u} : \mathbb{R}^2 \to \mathbb{R}$

$$\begin{aligned} \mathcal{N}[\mathbf{u}](x) &= \Delta \mathbf{u}(x) - f(x), \quad \text{for } x \in \Omega, \\ \mathcal{B}[\mathbf{u}](x) &= \mathbf{u}(x) - u_0, \qquad \text{for } x \in \partial\Omega. \end{aligned}$$

PINNs - Reminder of PDE-Based Loss

- Find solution $\mathbf{u} : \Omega \subset \mathbb{R}^n \to \mathbb{R}^m$ of

$$\begin{aligned} \mathcal{N}[\mathbf{u}](\mathbf{x}) &= 0, \text{ for } \mathbf{x} \in \Omega, \\ \mathcal{B}[\mathbf{u}](\mathbf{x}) &= 0, \text{ for } \mathbf{x} \in \partial\Omega. \end{aligned}$$

- Sample points $x_i^{\mathcal{N}} \in \Omega$ and $x_j^{\mathcal{B}} \in \partial\Omega$
- Train network $\mathbf{u}_\theta$ minimizing the PDE-loss

$$\sum_i \left\| \mathcal{N}[\mathbf{u}_\theta](x_i^{\mathcal{N}}) \right\|^2 + \sum_j \left\| \mathcal{B}[\mathbf{u}_\theta](x_j^{\mathcal{B}}) \right\|^2$$

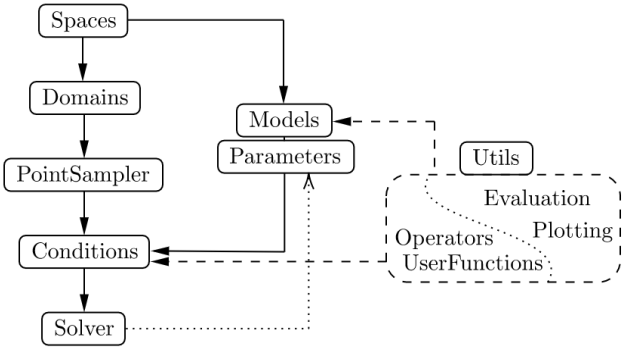
- E.g. $\Omega = [0,1] \times [0,1], \mathbf{u} : \mathbb{R}^2 \to \mathbb{R}$

$$\begin{aligned} \mathcal{N}[\mathbf{u}](x) &= \Delta \mathbf{u}(x) - f(x), \text{ for } x \in \Omega, \\ \mathcal{B}[\mathbf{u}](x) &= \mathbf{u}(x) - u_0, \qquad \text{for } x \in \partial\Omega. \end{aligned}$$

TorchPhysics Structure

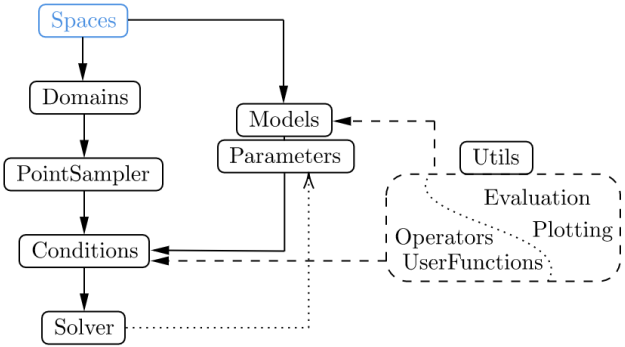
TorchPhysics

Structure



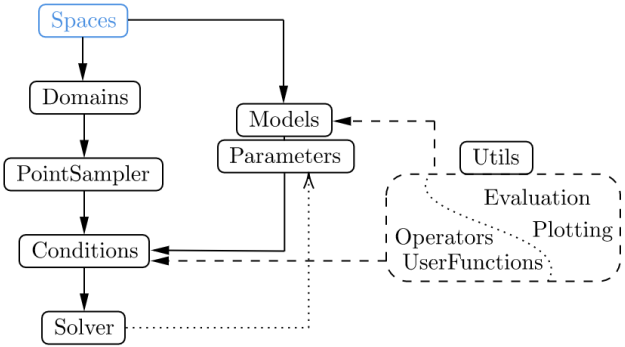
TorchPhysics

Spaces



TorchPhysics

Spaces

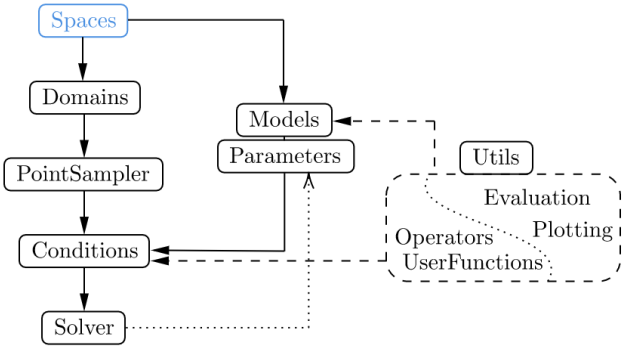


Example: $\Omega = [0,1] \times [0,1]$

$$\Delta u(x) = f(x), \quad \text{for } x \in \Omega,$$
$$u(x) = u_0, \quad \text{for } x \in \partial\Omega.$$

TorchPhysics

Spaces

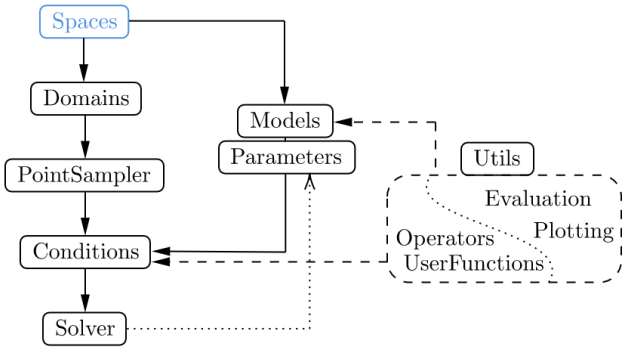


Example: $\Omega = [0,1] \times [0,1]$

$$\Delta u(x) = f(x), \quad \text{for } x \in \Omega,$$
$$u(x) = u_0, \quad \text{for } x \in \partial\Omega.$$

TorchPhysics

Spaces



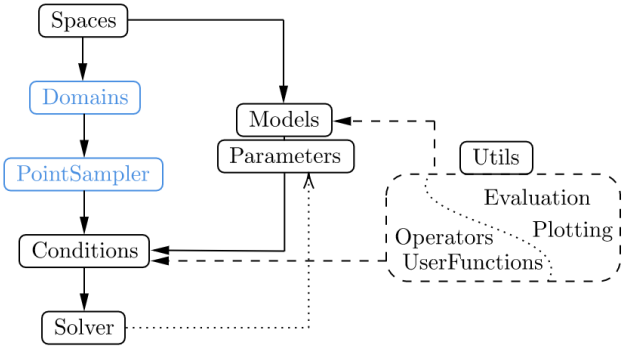
Example: $\Omega = [0,1] \times [0,1]$

$$\Delta u(x) = f(x), \quad \text{for } x \in \Omega,$$
$$u(x) = u_0, \quad \text{for } x \in \partial\Omega.$$

```
1 import torchphysics as tp
2 X = tp.spaces.R2('x')
3 U = tp.spaces.R1('u')
```

TorchPhysics

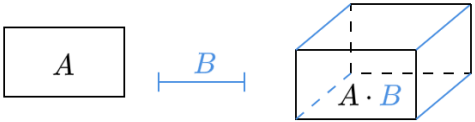
Domains



Domains

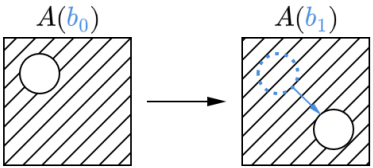
- Basic geometries implemented:
  - `Point, Interval, Parallelogram, Circle, ...`
- Complex domains via logical operators:

- Higher dimensional domains via Cartesian product:



Domains

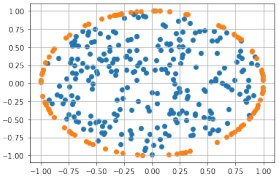
- Domain $A(b)$ may depend on point b from domain B (e.g. time-dependent)

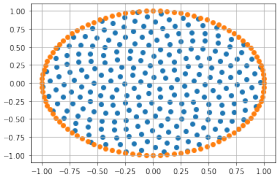


- Property `.boundary` returns the boundary as a new `Domain` object

PointSampler

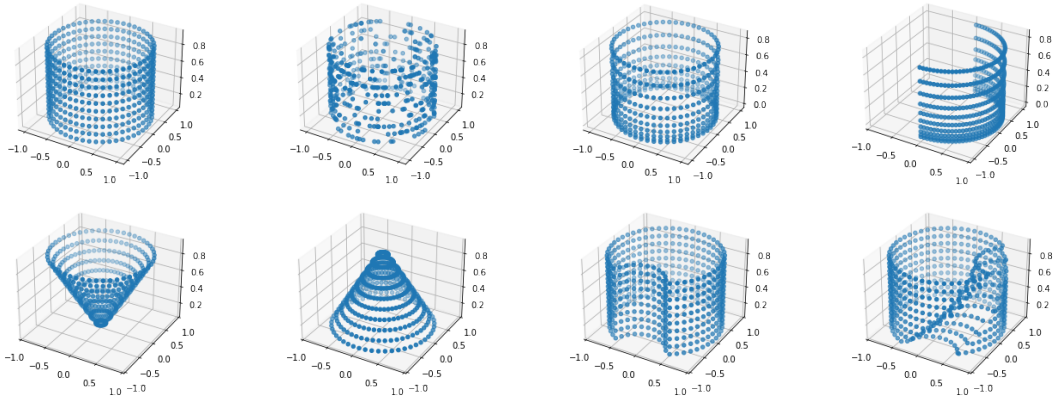
- Creation of training/validation points inside of the domains
- Different types of sampling:
  - `RandomUniformSampler, GridSampler, GaussianSampler, AdaptiveRejectionSampler, ...`
- Can be combined as desired
- Filtering of specific points possible





PointSampler

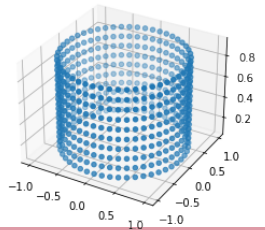
Flexible domain sampling



PointSampler - For Product Domains

- Cylinder in TorchPhysics: Product Domain

```
circle          = tp.domains.Circle(space=X, center=[0,0], radius=1)

interval        = tp.domains.Interval(space=Z, 0, 1)

cylinder_bound = circle.boundary * interval
```

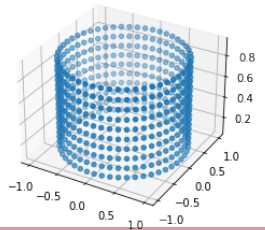


PointSampler - For Product Domains

- Cylinder in TorchPhysics: Product Domain

```
circle          = tp.domains.Circle(space=X, center=[0,0], radius=1)

interval        = tp.domains.Interval(space=Z, 0, 1)

cylinder_bound = circle.boundary * interval
```

- How to sample points at cylinder's boundary?



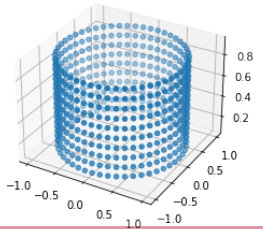
PointSampler - For Product Domains

Two options

1) Multiply samplers for circle.boundary and cylinder

```
sampler_circle_boundary = tp.samplers.GridSampler(circle.boundary, 70)

sampler_interval        = tp.samplers.GridSampler(interval, 10)

sampler_cylinder_bound  = sampler_circle_boundary * sampler_interval
```

- Works for all samplers!



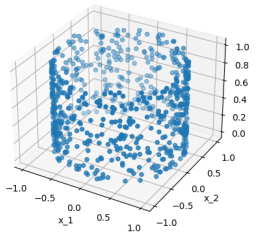
PointSampler - For Product Domains

Two options

2) Direct sampler for cylinder

```
1 sampler_cylinder_bound = tp.samplers.RandomUniformSampler(
2                                                 circle.boundary * interval, 700)
```

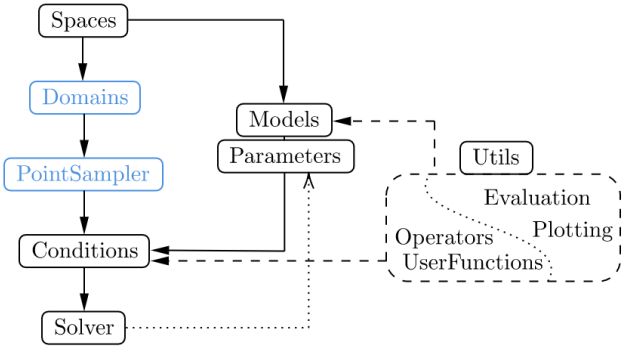
- Works for some samplers only!
  $\rightarrow$ E.g. not for GridSamplers



TorchPhysics

Domains - Code

```
4 omega = tp.domains.Parallelogram(X, [0,0], [1,0], [0,1])
```



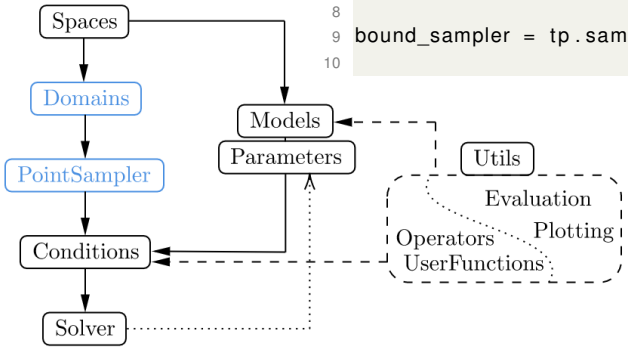
Example: $\Omega = [0,1] \times [0,1]$

$$\Delta u(x) = f(x), \quad \text{for } x \in \Omega,$$
$$u(x) = u_0, \quad \text{for } x \in \partial\Omega.$$

TorchPhysics

Domains - Code

```
4  omega = tp.domains.Parallelogram(X, [0,0], [1,0], [0,1])
5
6  inner_sampler = tp.samplers.RandomUniformSampler(omega,
7                                                   n_points=15000)
8
9  bound_sampler = tp.samplers.GridSampler(omega.boundary,
10                                                  n_points=5000)
```

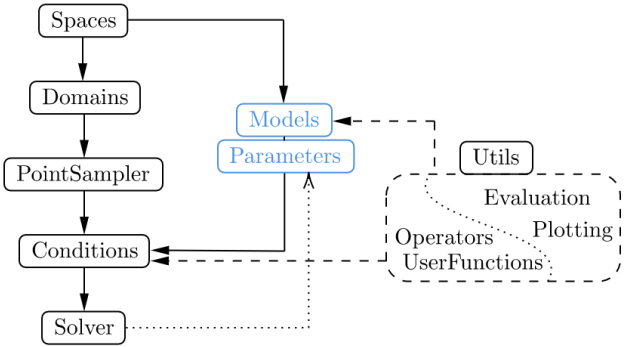


Example: $\Omega = [0,1] \times [0,1]$

$$\Delta u(x) = f(x), \quad \text{for } x \in \Omega,$$
$$u(x) = u_0, \quad \text{for } x \in \partial\Omega.$$

TorchPhysics

Neural Networks



Models

- Load any PyTorch model into TorchPhysics
- Several pre-implemented architectures:
  - FCN, ResNet, DeepOnet, . . .
  - Easy to customize

```
11 model = tp.models.FCN(input_space=X,
12                        output_space=U,
13                        hidden=(20,20,20))
```

Models

- Load any PyTorch model into TorchPhysics
- Several pre-implemented architectures:
  - FCN, ResNet, DeepOnet, ...
  - Easy to customize

```
11 model = tp.models.FCN(input_space=X,
12                        output_space=U,
13                        hidden=(20,20,20))
```

- Trainable parameters for parameter identification

```
1 D_space = tp.spaces.R1('D')
2 D       = tp.models.Parameter(init=1.0,
3                               space=D_space)
```

Models

- Load any PyTorch model into TorchPhysics
- Several pre-implemented architectures:
  - FCN, ResNet, DeepOnet, ...
  - Easy to customize

```
model = tp.models.FCN(input_space=X,
                      output_space=U,
                      hidden=(20,20,20))
```

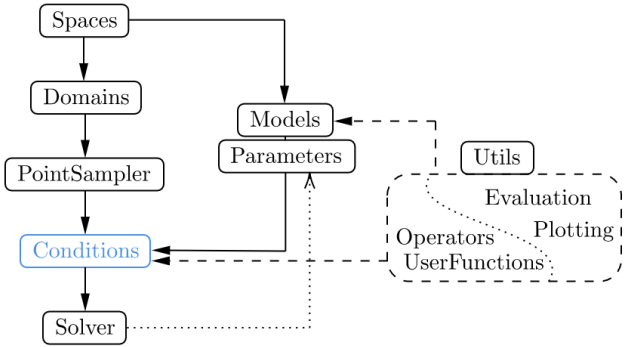
- Trainable parameters for parameter identification

```
D_space = tp.spaces.R1('D')
D       = tp.models.Parameter(init=1.0,
                              space=D_space)
```

- Further functionalities:
  - Input normalization
  - Parallel/sequential evaluation of multiple networks
  - ...

TorchPhysics

Conditions



Conditions

- Different types, e.g. PINNCondition
- Represents one mathematical condition, e.g.

$$\Delta u(x) = f(x) \text{ in } \Omega \qquad \text{or} \qquad u(x) = u_0 \text{ at } \partial\Omega$$

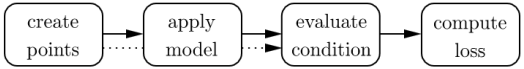
- `DifferentialOperators` allow natural definition:

```
14 def pde_residual(u, x):
15     return tp.utils.laplacian(u, x) - f(x)
16
17 pde_cond = PINNCondition(model,
18                           inner_sampler,
19                           pde_residual)
```

```
20 def boundary_residual(u, x):
21     return u - u_0
22
23 boundary_cond = PINNCondition(model,
24                               bound_sampler,
25                               boundary_residual)
```

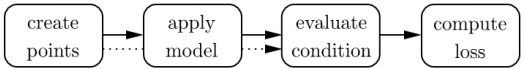
Conditions

- Common scheme of most conditions:

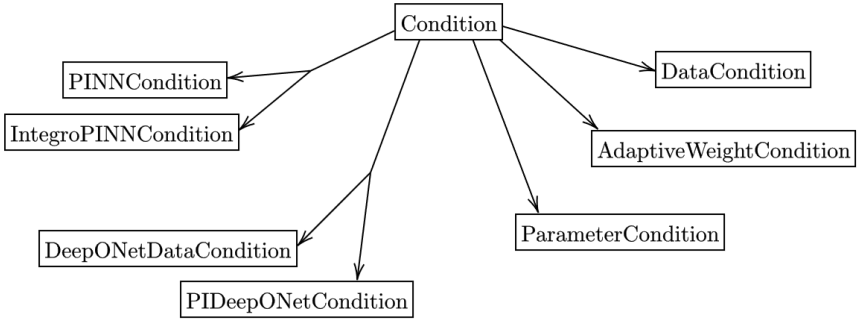


Conditions

- Common scheme of most conditions:

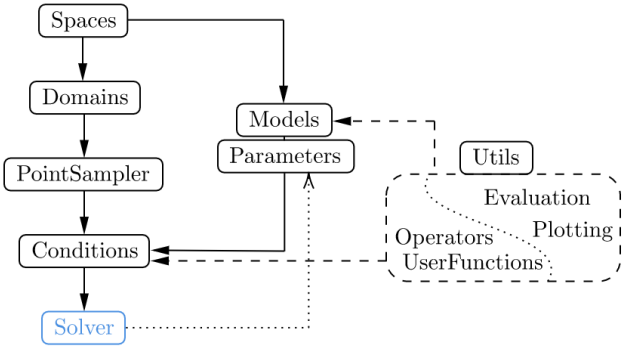


- Some other conditions:



TorchPhysics

Solver



Solver

- Collects all conditions
  $\rightarrow$ overall loss computable
- Flexible choice optimization algorithm

```
optim = tp.OptimizerSetting(torch.optim.Adam, lr=0.001)
solver = tp.solver.Solver([boundary_cond, pde_cond],
                          optimizer_setting=optim))
```

Solver

- Collects all conditions
  $\rightarrow$ overall loss computable
- Flexible choice optimization algorithm

```
optim = tp.OptimizerSetting(torch.optim.Adam, lr=0.001)
solver = tp.solver.Solver([boundary_cond, pde_cond],
                          optimizer_setting=optim))
```

- Based upon  PyTorch Lightning

```
import pytorch_lightning as pl
trainer = pl.Trainer(gpus=1, # use one GPU
                     max_steps=3000) # iteration number
trainer.fit(solver)
```

Solver

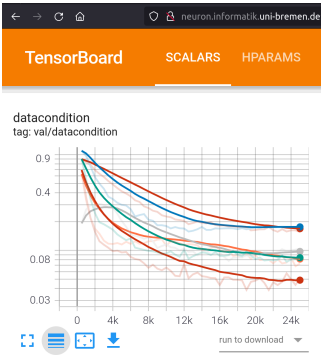
- Collects all conditions
  $\rightarrow$ overall loss computable
- Flexible choice optimization algorithm

```
optim = tp.OptimizerSetting(torch.optim.Adam, lr=0.001)
solver = tp.solver.Solver([boundary_cond, pde_cond],
                          optimizer_setting=optim))
```

- Based upon PyTorch Lightning

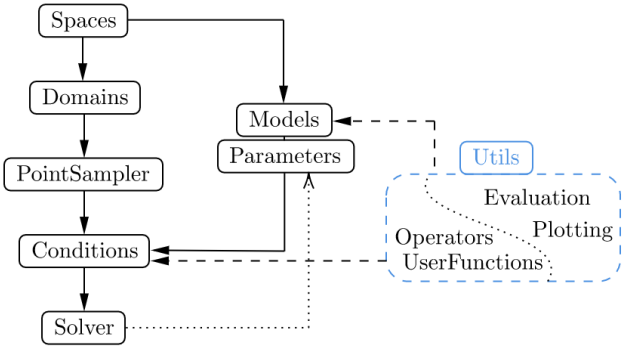
```
import pytorch_lightning as pl
trainer = pl.Trainer(gpus=1, # use one GPU
                     max_steps=3000) # iteration number
trainer.fit(solver)
```

- Training on multiple GPUs
- Detailed monitoring of all conditions during training



TorchPhysics

Utils



Utils - Plot Results

$$\Delta u(x) = f(x), \qquad \text{for } x \in \Omega$$
$$u(x) = u_0, \qquad \text{for } x \in \partial\Omega$$

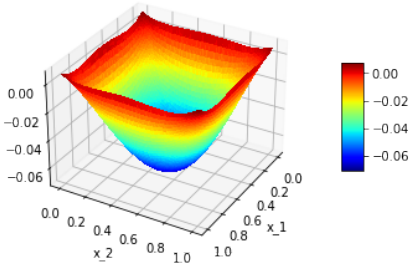


Figure: Solution of the PDE, for $u_0 = 0, f = 1$

```python
import torchphysics as tp
X = tp.spaces.R2('x')
U = tp.spaces.R1('u')

omega = tp.domains.Parallelogram(X, [0,0], [1,0], [0,1])

inner_sampler = tp.samplers.RandomUniformSampler(omega,
                                                 n_points=15000)

bound_sampler = tp.samplers.GridSampler(omega.boundary,
                                        n_points=5000)

model = tp.models.FCN(input_space=X,
                      output_space=U,
                      hidden=(20,20,20))

def pde_residual(u, x):
    return tp.utils.laplacian(u, x) - f(x)

def boundary_residual(u, x):
    return u - u_0

boundary_cond = tp.conditions.PINNCondition(model,
                                            bound_sampler,
                                            boundary_residual)

pde_cond = tp.conditions.PINNCondition(model,
                                       inner_sampler,
                                       pde_residual)


optim = tp.OptimizerSetting(torch.optim.Adam, lr=0.001)
solver = tp.solver.Solver([boundary_cond, pde_cond],
                          optimizer_setting=optim))
import pytorch_lightning as pl
trainer = pl.Trainer(gpus=1, # use one GPU
                     max_steps=3000) # iteration number
trainer.fit(solver)

plot_sampler = tp.samplers.PlotSampler(plot_domain=omega,
                                       n_points=2000)
fig = tp.utils.plot(model, lambda u : u, plot_sampler)
```

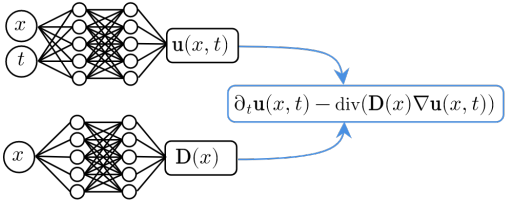
Examples and Applications with PINNs

Inverse Problem

Heat Equation

- Given measurements $\{\mathbf{u}_i, x_i, t_i\}_{i=1}^N$
- Find diffusion $\mathbf{D} : \Omega \to \mathbb{R}$ such that

$$\partial_t \mathbf{u} = \operatorname{div}(D \nabla \mathbf{u}), \qquad \text{in } \Omega \times [0, T]$$



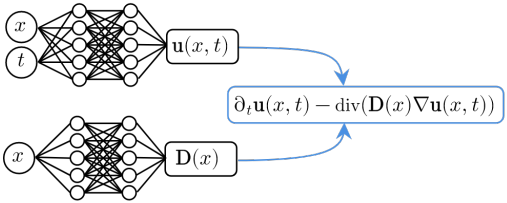
Inverse Problem

Heat Equation

- Given measurements $\{\mathbf{u}_i, x_i, t_i\}_{i=1}^N$
- Find diffusion $\mathbf{D} : \Omega \to \mathbb{R}$ such that

$$\partial_t \mathbf{u} = \operatorname{div}(D \nabla \mathbf{u}), \qquad \text{in } \Omega \times [0, T]$$

- **Idea**: Use two NNs, one for **u** and one for **D**
- Both trained on PDE-condition



Inverse Problem

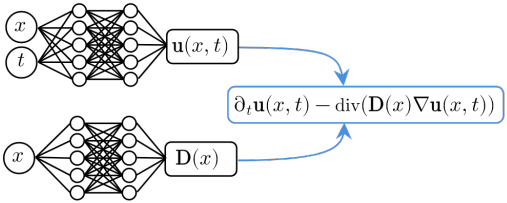
Heat Equation

- Given measurements $\{\mathbf{u}_i, x_i, t_i\}_{i=1}^N$
- Find diffusion $\mathbf{D} : \Omega \to \mathbb{R}$ such that

$$\partial_t \mathbf{u} = \operatorname{div}(D \nabla \mathbf{u}), \qquad \text{in } \Omega \times [0, T]$$

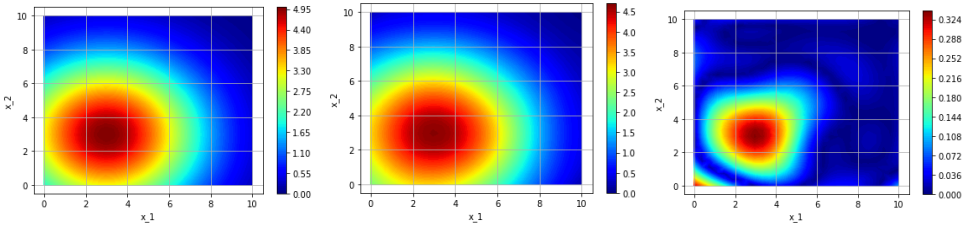
- **Idea**: Use two NNs, one for **u** and one for **D**
- Both trained on PDE-condition
- u_θ also on data condition

$$u_\theta(x_i, t_i) = \mathbf{u_i}$$



Inverse Problem - PINN Result

Heat Equation



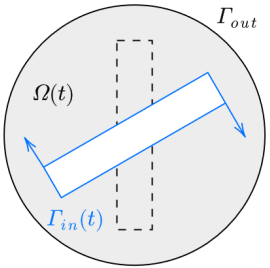
(a) True D (b) D obtained via PINN (c) Absolute error

Time-Dependent Domain

PINNs for Navier-Stokes Example

Bar heats up and rotates within some fluid:

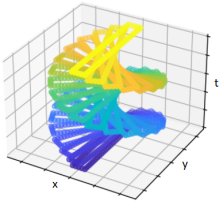
$$
\begin{aligned}
\partial_t u + (u \cdot \nabla) u &= \nu \Delta u - \nabla p, && \text{in } \Omega(t),\\
\nabla \cdot u &= 0, && \text{in } \Omega(t),\\
\partial_t T + u \cdot \nabla T &= \lambda \Delta T, && \text{in } \Omega(t),\\
u(0,\cdot), p(0,\cdot), T(0,\cdot) &= 0, 0, 270 && \text{in } \Omega(0),\\
u, T &= (0,0), 270, && \text{on } \Gamma_{\text{out}},\\
u, T &= u_{\text{in}}(t), T_{\text{in}}(t), && \text{on } \Gamma_{\text{in}}.
\end{aligned}
$$

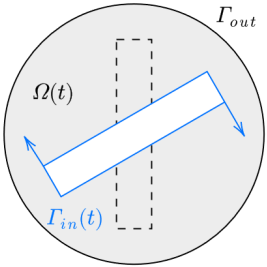


Time-Dependent Domain

Implementation inside TorchPhysics

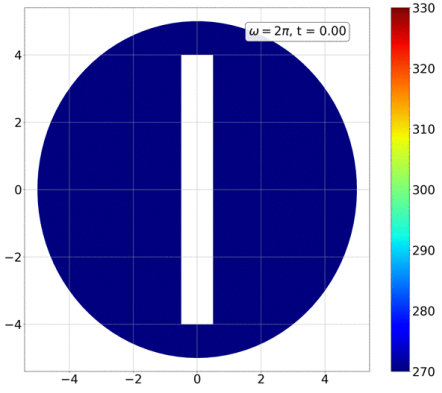
```
def corner1(t):
    return rotation_matrix(t) * start_position_1

bar    = tp.domains.Parallelogram(X, corner1, corner2, corner3)
circle = tp.domains.Circle(X, center, radius)

omega  = circle - bar
```





Time-Dependent Domain

Learned Temperature Field



Summary and Outlook

TORCHPHYSICS **is**

- easy to get started with
- providing flexible domain creation and sampling
- built of decoupled modules $\rightarrow$ easily extendable

TORCHPHYSICS **will**

- provide further methods, e.g. FNOs, Hidden Physics
- contain more examples for every implemented method
- always be open for contributions!