Universität Bremen

Center for Industrial Mathematics (ZeTeM)

Faculty 03

Mathematics / Computer science

Introduction to TorchPhysics

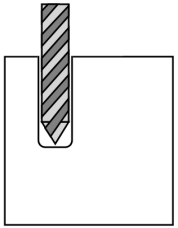
Getting started with a simple example

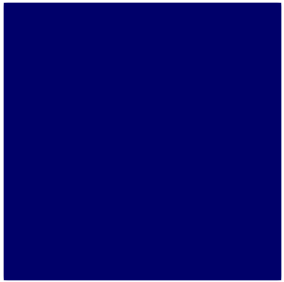
Tom Freudenberg, Nick Heilenkötter, Janek Gödeke

Berlin, 14.11.2024

Main Goal of Today

- Solve a simplified drilling problem
- Heat equation on a time dependent domain
- Use PINNs and TorchPhysics to solve this problem

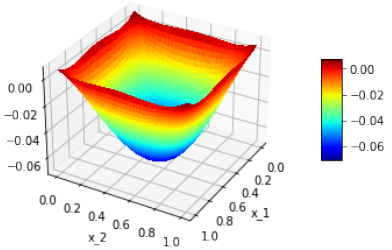




Starting with TorchPhysics

- We introduce the library with the Laplace equation:

$$\begin{aligned} \Delta u &= 1 \quad \text{in } \Omega = [0,1] \times [0,1] \\ u &= 0 \quad \text{on } \partial\Omega \end{aligned}$$

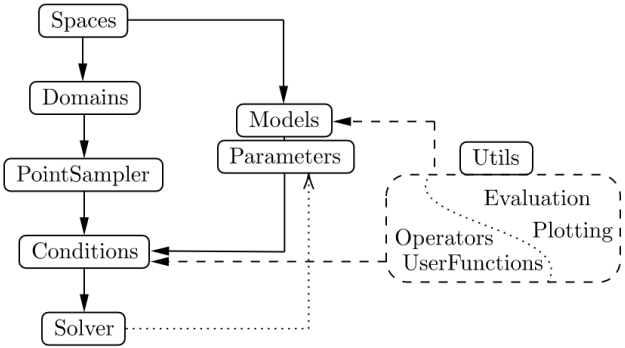


But first...

Setting up coding environment

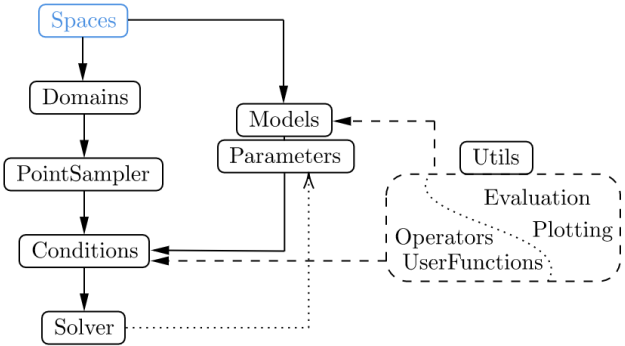
TorchPhysics

Structure



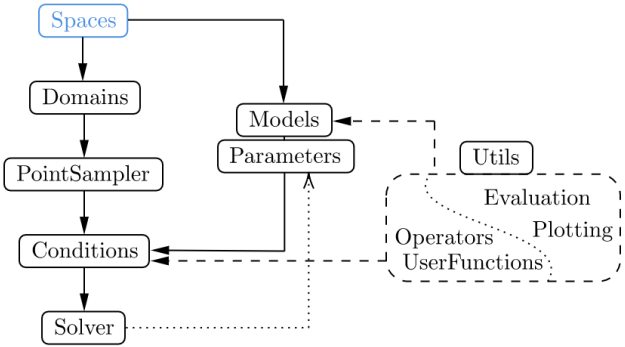
TorchPhysics

Spaces



TorchPhysics

Spaces

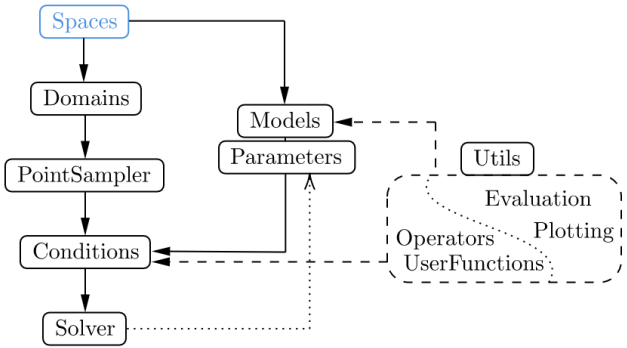


Example: $\Omega = [0,1] \times [0,1]$

$$\Delta u(x) = 1, \quad \text{for } x \in \Omega,$$
$$u(x) = 0, \quad \text{for } x \in \partial\Omega.$$

TorchPhysics

Spaces

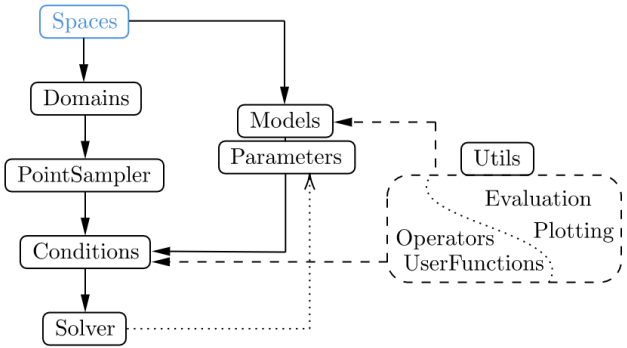


Example: $\Omega = [0,1] \times [0,1]$

$$\Delta u(x) = 1, \quad \text{for } x \in \Omega,$$
$$u(x) = 0, \quad \text{for } x \in \partial\Omega.$$

TorchPhysics

Spaces



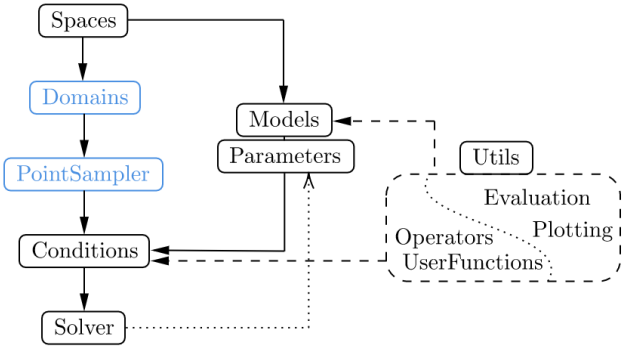
Example: $\Omega = [0,1] \times [0,1]$

$$\Delta u(x) = 1, \quad \text{for } x \in \Omega,$$
$$u(x) = 0, \quad \text{for } x \in \partial\Omega.$$

```
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, ...`

Domains

- Basic geometries implemented:
  - `Point, Interval, Parallelogram, Circle, ...`
- Here: `Parallelogram`

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

$$\begin{aligned} \Delta u(x) &= 1, \quad \text{for } x \in \Omega, \\ u(x) &= 0, \quad \text{for } x \in \partial\Omega. \end{aligned}$$

Domains

- Basic geometries implemented:
  - `Point, Interval, Parallelogram, Circle, ...`
- Here: `Parallelogram`

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

$$\begin{aligned} \Delta u(x) &= 1, \quad \text{for } x \in \Omega, \\ u(x) &= 0, \quad \text{for } x \in \partial\Omega. \end{aligned}$$

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

PointSampler

- Creation of training/validation points inside of the domains
- Different types of sampling:
  - `RandomUniformSampler`, `GridSampler`, `GaussianSampler`, `AdaptiveRejectionSampler`, `...`

PointSampler

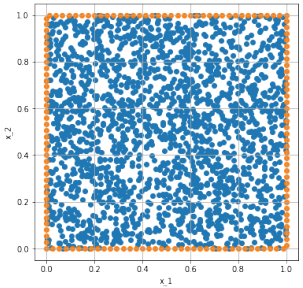
- Creation of training/validation points inside of the domains
- Different types of sampling:
  - `RandomUniformSampler, GridSampler, GaussianSampler, AdaptiveRejectionSampler, ...`

```
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  boundary_sampler = tp.samplers.GridSampler(omega.boundary,
10                                             n_points=5000)
```

PointSampler

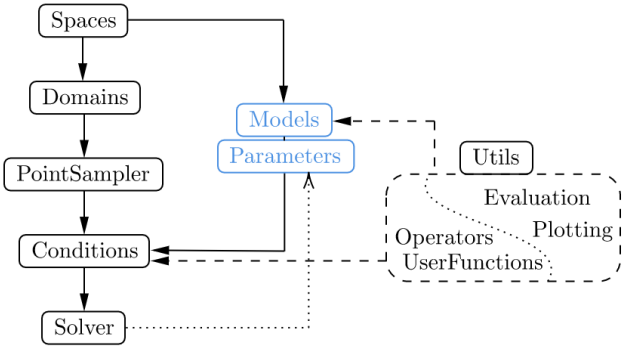
- Creation of training/validation points inside of the domains
- Different types of sampling:
  - `RandomUniformSampler, GridSampler, GaussianSampler, AdaptiveRejectionSampler, ...`

```
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  boundary_sampler = tp.samplers.GridSampler(omega.boundary,
10                                             n_points=5000)
```



TorchPhysics

Neural Networks

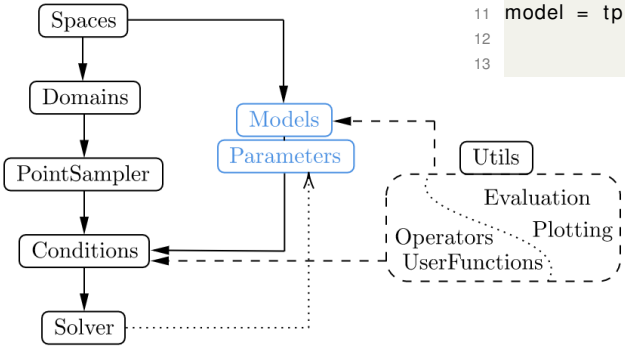


TorchPhysics

Neural Networks

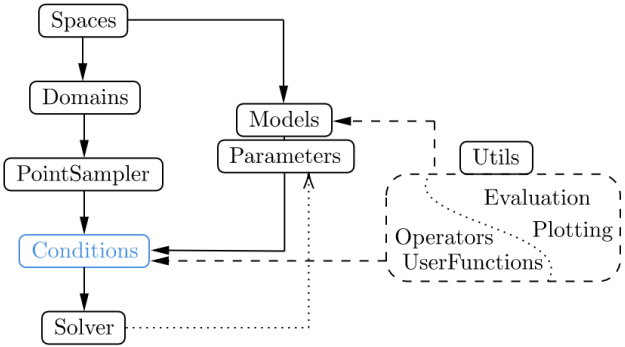
$$\Delta u(x) = 1$$

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



TorchPhysics

Conditions



Conditions

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

$$\Delta u(x) = 1 \text{ in } \Omega \qquad \text{or} \qquad u(x) = 0 \text{ at } \partial\Omega$$

- `DifferentialOperators` allow natural definition:

Conditions

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

$$\Delta u(x) = 1 \text{ in } \Omega \qquad \text{or} \qquad u(x) = 0 \text{ at } \partial\Omega$$

- `DifferentialOperators` allow natural definition:

```
14 def pde_residual(u, x):
15     return tp.utils.laplacian(u, x) - 1.0
16
17 pde_cond = PINNCondition(model,
18                            inner_sampler,
19                            pde_residual)
```

Conditions

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

$$\Delta u(x) = 1 \text{ in } \Omega \qquad \text{or} \qquad u(x) = 0 \text{ at } \partial\Omega$$

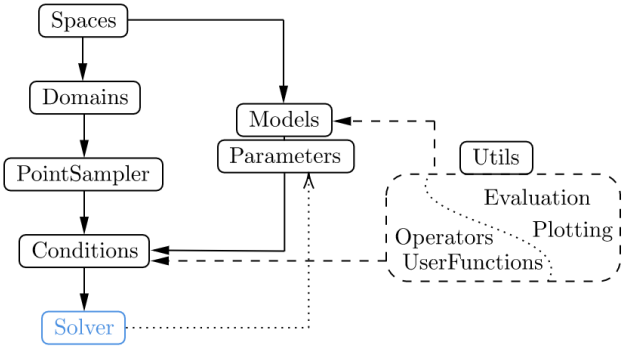
- `DifferentialOperators` allow natural definition:

```
14 def pde_residual(u, x):
15     return tp.utils.laplacian(u, x) - 1.0
16
17 pde_cond = PINNCondition(model,
18                            inner_sampler,
19                            pde_residual)
```

```
20 def boundary_residual(u):
21     return u - 0.0
22
23 boundary_cond = PINNCondition(model,
24                                boundary_sampler,
25                                boundary_residual)
```

TorchPhysics

Solver



Solver

- Collects all conditions → overall loss computation
- Flexible choice optimization algorithm

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

Solver

- Collects all conditions → overall loss computation
- Flexible choice optimization algorithm

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

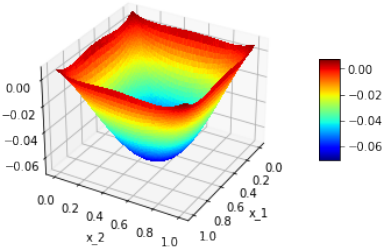
- Based upon  PyTorch Lightning

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

Utils - Plot Results

$$\Delta u(x) = 1.0 \quad \text{for } x \in \Omega$$
$$u(x) = 0.0 \quad \text{for } x \in \partial\Omega$$

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

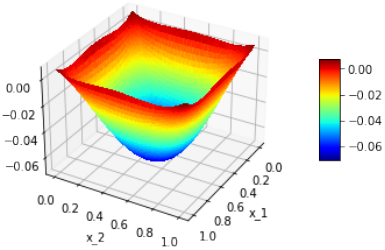


Learned solution of the PDE

Utils - Plot Results

$$\Delta u(x) = 1.0 \quad \text{for } x \in \Omega$$
$$u(x) = 0.0 \quad \text{for } x \in \partial\Omega$$

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



Learned solution of the PDE

Does your solution look like the one at the left?

First extension of the example

- Learning the time dependent Laplace equation (**heat equation**):

$$\begin{aligned} \partial_t u - 0.1\Delta u &= 1 && \text{in } \Omega \times [0,2] \\ u &= 0 && \text{on } \partial\Omega \times [0,2] \\ u(\cdot, 0) &= 0 && \text{in } \Omega \end{aligned}$$

First extension of the example

- Learning the time dependent Laplace equation (**heat equation**):

$$\begin{aligned} \partial_t u - 0.1\Delta u &= 1 \quad \text{in } \Omega \times [0,2] \\ u &= 0 \quad \text{on } \partial\Omega \times [0,2] \\ u(\cdot, 0) &= 0 \quad \text{in } \Omega \end{aligned}$$

- Aspects to adjust:
  - Adding a time variable `t`, time interval and sample time points
  - One more input to the neural network
  - Adapt PDE-condition and implement initial condition
- Template: `Exercise_2.ipynb`