Universität Bremen

Center for Industrial Mathematics (ZeTeM)

Faculty 03

Mathematics / Computer science

Introduction to TorchPhysics

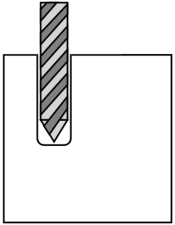
Getting started with a simple example

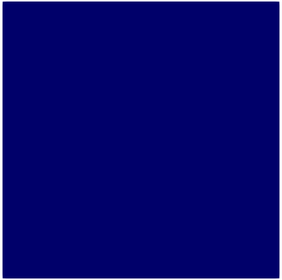
Tom Freudenberg, Nick Heilenkötter, Janek Gödeke

Heidelberg, 07.11.2023

Main Goal of Today

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

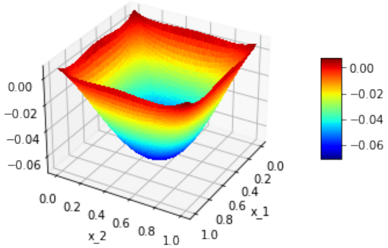




Starting with TorchPhysics

- We introduce the library with the Laplace equation:

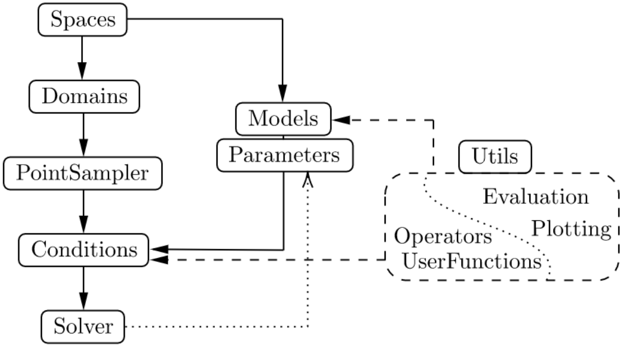
$$\begin{aligned} \Delta u &= 1 \quad \text{in } \Omega = [0,1] \times [0,1] \\ u &= 0 \quad \text{on } \partial\Omega \end{aligned}$$



But first...

Setting up Google Colab

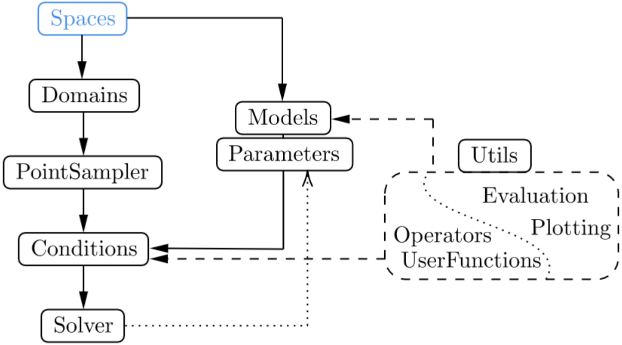
TorchPhysics
Structure



Tom Freudenberg, Nick Heilenkötter, Janek Gödeke
Faculty 03
Mathematics / Computer science

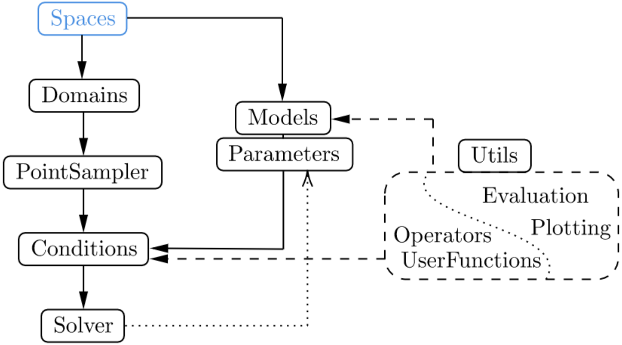
TorchPhysics

Spaces



TorchPhysics

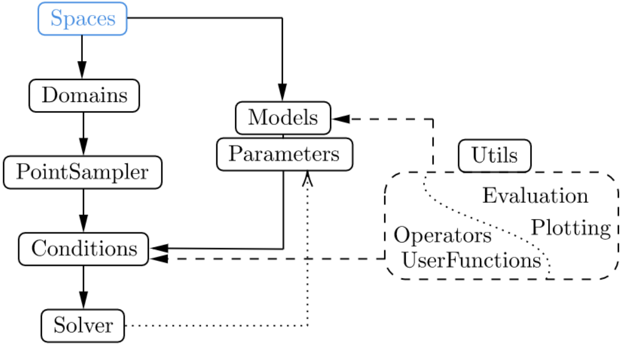
Spaces



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

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

TORCHPHYSICS
Spaces

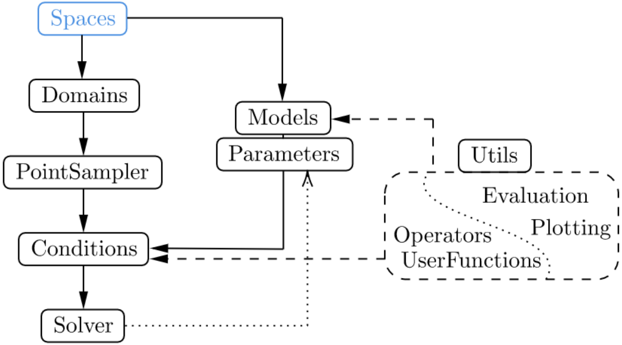


Example: $\Omega = [0,1] \times [0,1]$

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

TorchPhysics

Spaces

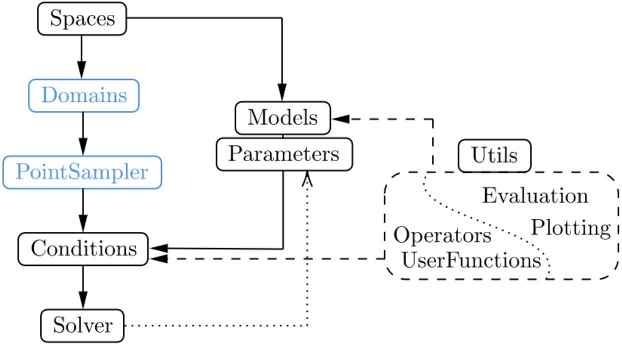


Example: $\Omega = [0,1] \times [0,1]$

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

```python
import torchphysics as tp
X = tp.spaces.R2('x')
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: $\Omega = [0,1] \times [0,1]$

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

```
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

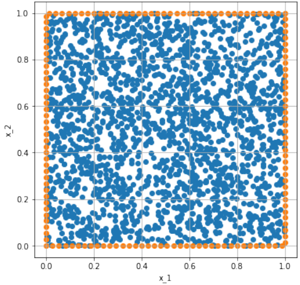
- Creation of training/validation points inside of the domains
- Different types of sampling:
  - `RandomUniformSampler`, `GridSampler`, `GaussianSampler`, `AdaptiveRejectionSampler`, ...

```
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)
```

PointSampler

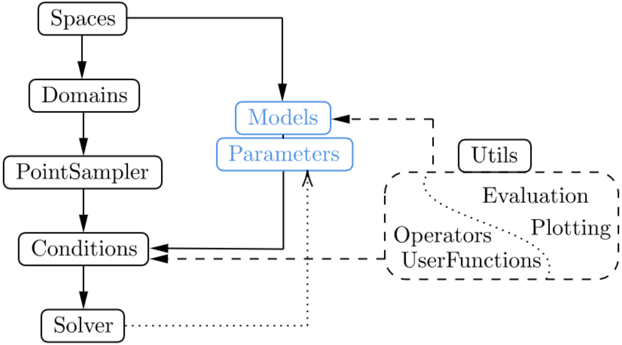
- Creation of training/validation points inside of the domains
- Different types of sampling:
  - `RandomUniformSampler`, `GridSampler`, `GaussianSampler`, `AdaptiveRejectionSampler`, ...

```
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)
```



TORCHPHYSICS

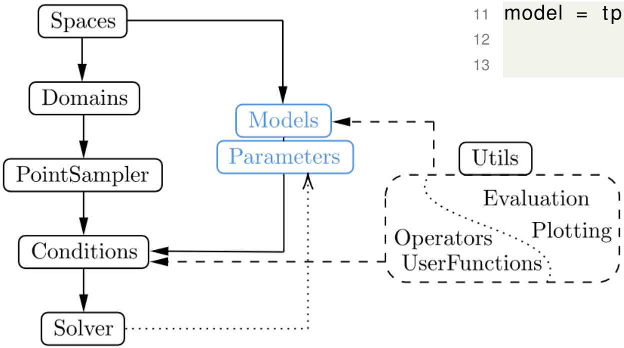
Neural Networks



TORCHPHYSICS

Neural Networks

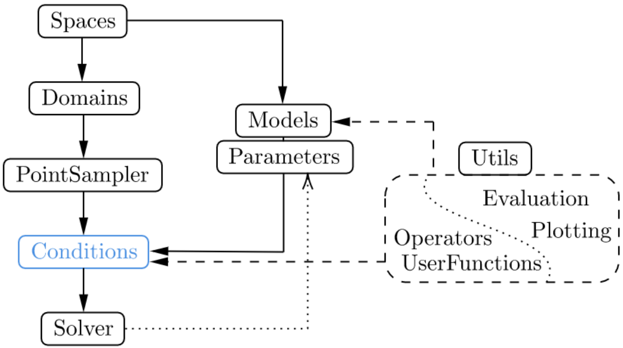
$$\Delta u(x) = 1$$




```
model = tp.models.FCN(input_space=X,
                      output_space=U,
                      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:

```
def pde_residual(u, x):
    return tp.utils.laplacian(u, x) - 1.0

pde_cond = PINNCondition(model,
                         inner_sampler,
                         pde_residual)
```

Conditions

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

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

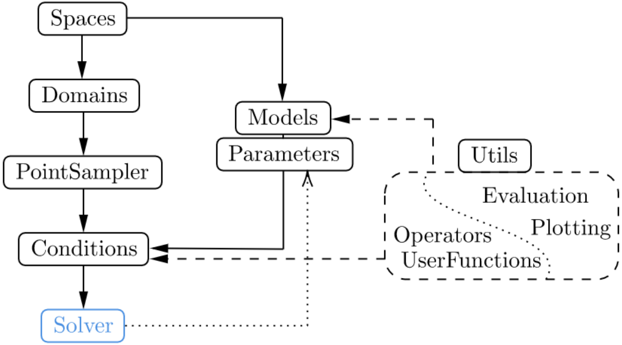
- `DifferentialOperators` allow natural definition:

```python
def pde_residual(u, x):
    return tp.utils.laplacian(u, x) - 1.0

pde_cond = PINNCondition(model,
                         inner_sampler,
                         pde_residual)
```

```python
def boundary_residual(u):
    return u - 0.0

boundary_cond = PINNCondition(model,
                              bound_sampler,
                              boundary_residual)
```

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))
```

Tom Freudenberg, Nick Heilenkötter, Janek Gödeke
Faculty 03
Mathematics / Computer science

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)
```

Utils - Plot Results

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

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

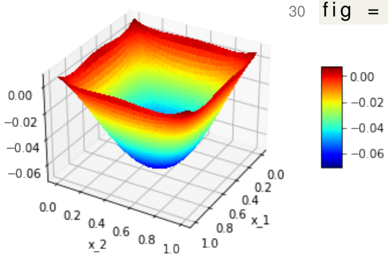


Figure: Solution of the PDE

Utils - Plot Results

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

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

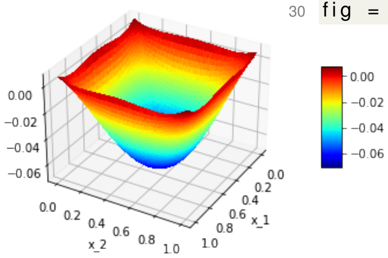


Figure: Solution of the PDE

Does your solution look like the one in the picture?

First extension of the example

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

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

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: `Example_2.ipynb`