Universität
Bremen

Center for Industrial
Mathematics (ZeTeM)
Faculty 03
Mathematics / Computer science

Introduction to TorchPhysics

Boundary conditions and geometry operations

Nick Heilenkötter, Janek Gödeke, Tom
Freudenberg
Heidelberg, 07.11.2023

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

Boundary Conditions

Common types of boundary conditions

- Dirichlet condition:

$$u = f_D \quad \text{on } \partial\Omega$$

- Neumann condition:

$$\nabla u \cdot \vec{n} = f_N \quad \text{on } \partial\Omega$$

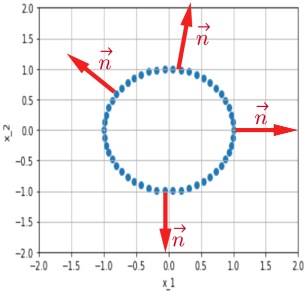
- Robin condition:

$$\nabla u \cdot \vec{n} = \alpha(u - f_R) \quad \text{on } \partial\Omega$$

Boundary Conditions

Implementation in TorchPhysics

- Need normal vector $\vec{n}$ for Neumann and Robin conditions
  $\rightarrow$ Domains in `TorchPhysics` can compute their normal vectors



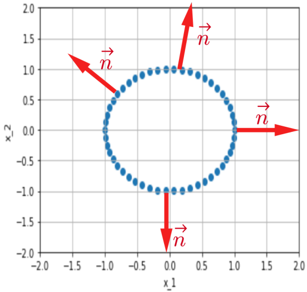
Nick Heilenkötter, Janek Gödeke, Tom Freudenberg
Faculty 03
Mathematics / Computer science

Boundary Conditions

Implementation in TorchPhysics

- Need normal vector $\vec{n}$ for Neumann and Robin conditions
  $\rightarrow$ Domains in `TorchPhysics` can compute their normal vectors

```python
def neumann_residual(u, x):
    normals = omega.boundary.normal(x)
    u_n     = tp.utils.normal_derivative(u, normals, x)

    return u_n - f_N(x)

neumann_condition = PINNCondition(model,
                                  bound_sampler,
                                  neumann_residual)
```



Filter functions

- Usually distinct conditions on different boundary parts
  $\rightarrow$ Filters in `TorchPhysics` allow sampling on specific parts
- Checks if point on correct part
  $\rightarrow$ Returning `True` or `False`

Filter functions

- Usually distinct conditions on different boundary parts
  $\rightarrow$ Filters in `TorchPhysics` allow sampling on specific parts
- Checks if point on correct part
  $\rightarrow$ Returning `True` or `False`

```python
def filter_positive(x):
    return x[:, :1] > 0
sampler_1 = tp.samplers.RandomUniformSampler(omega.boundary,
                                             100, filter_fn=filter_positive)
```

Filter functions

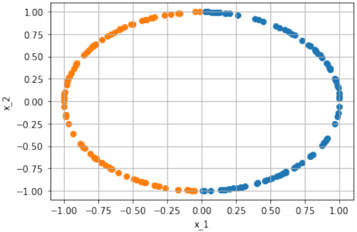
- Usually distinct conditions on different boundary parts
  $\rightarrow$ Filters in `TorchPhysics` allow sampling on specific parts
- Checks if point on correct part
  $\rightarrow$ Returning `True` or `False`

```
def filter_positive(x):
    return x[:, :1] > 0
sampler_1 = tp.samplers.RandomUniformSampler(omega.boundary,
                                             100, filter_fn=filter_positive)

def filter_negative(x):
    return x[:, :1] < 0
sampler_2 = tp.samplers.RandomUniformSampler(omega.boundary,
                                             100, filter_fn=filter_negative)

fig = tp.utils.scatter(X, sampler_1, sampler_2)
```

Filter functions

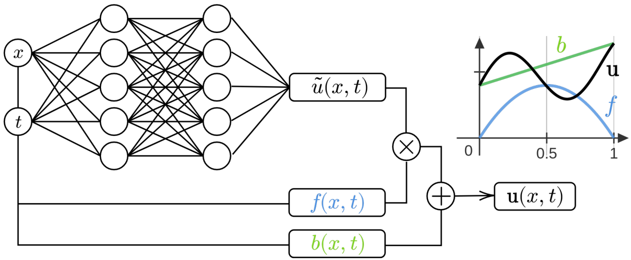
- Usually distinct conditions on different boundary parts
  $\rightarrow$ Filters in `TorchPhysics` allow sampling on specific parts
- Checks if point on correct part
  $\rightarrow$ Returning `True` or `False`

```python
def filter_positive(x):
    return x[:, :1] > 0
sampler_1 = tp.samplers.RandomUniformSampler(omega.boundary,
                                   100, filter_fn=filter_positive)

def filter_negative(x):
    return x[:, :1] < 0
sampler_2 = tp.samplers.RandomUniformSampler(omega.boundary,
                                   100, filter_fn=filter_negative)

fig = tp.utils.scatter(X, sampler_1, sampler_2)
```



PINN Extension: Hard Constraints

- Solution of PDE can be highly dependent on BC
- PINN: BC are enforced softly



Idea:[a]

- Include boundary conditions in network architecture
- Dirichlet conditions: *f* is zero at boundary, *b* defines values on boundary

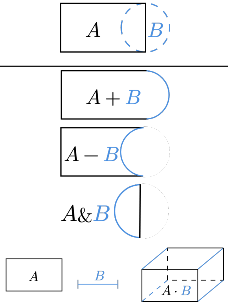
[a] Lu et al: *Physics-Informed Neural Networks with Hard Constraints for Inverse Design*, SIAM Journal on Scientific Computing, 2021

Domain creation

PointSampler samples points from a Domain, which can be created by:

- Loading an `.stl`-file
- Combining simple pre-implemented domains by:

	`A =, B =`
union	`C = A + B`
difference	`C = A - B`
intersection	`C = A & B`
cartesian product	`C = A * B`

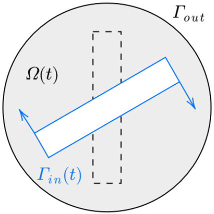


Time-dependent domains

- A domain may depend on parameters from another domain
- Resolved when cartesian product is created

First option:

```
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
t_int = tp.domains.Interval(T, 0, 1)

omega_t = omega * t_int
sampler = tp.samplers.RandomUniformSampler(omega_t, n_points=1000)
```

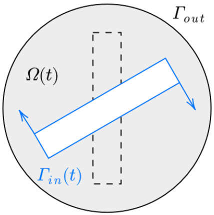


Time-dependent domains

- A domain may depend on parameters from another domain
- Resolved when cartesian product is created

Second option:

```
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
t_int = tp.domains.Interval(T, 0, 1)

sampler_omega = tp.samplers.GridSampler(omega, n_points=100)
sampler_t = tp.samplers.RandomUniformSampler(t_int, n_points=10)
sampler = sampler_omega * sampler_t
```



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

Exercises

$$
\begin{aligned}
\partial_t u - \kappa \Delta u &= 0.0 && \text{in } [0,2] \times \Omega \\
u(0,\cdot) &= 0 && \text{in } \Omega \\
u(t,x) &= 0, && \text{for } x_2 = 0, \\
\kappa \nabla u(t,x) \cdot n &= 0, && \text{on } \Gamma_N, \\
\kappa \nabla u(t,x) \cdot n &= f, && \text{on } \Gamma_H
\end{aligned}
$$

- Example 3: More complex boundary conditions
- Example 4: Change geometry
- Example 5: Time dependent geometry: drilling