



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
Berlin, 14.11.2024

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

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:

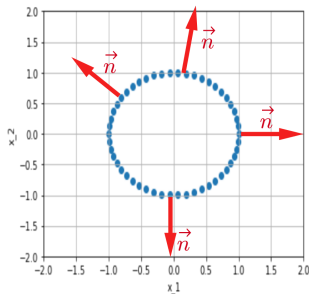
$$\nabla u \cdot \vec{n} = \alpha(u - f_R) \quad \text{on } \partial\Omega$$

Recall: Boundary conditions enforced in PINN via loss term $\frac{1}{N_B} \sum_{j=1}^{N_B} \|\mathcal{B}[\mathbf{u}_\theta](x_j^B)\|^2$

Boundary Conditions

Implementation in TorchPhysics

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



Boundary Conditions

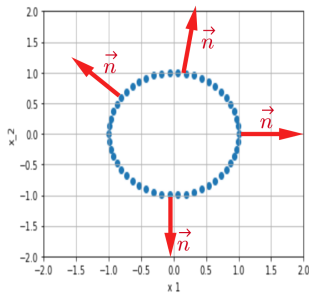
Implementation in TorchPhysics

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

```

1 def neumann_residual(u, x):
2     normals = omega.boundary.normal(x)
3     u_n      = tp.utils.normal_derivative(u, normals, x)
4
5     return u_n - f_N(x)
6
7 neumann_condition = PINNCondition(model,
8                                   bound_sampler,
9                                   neumann_residual)

```



Filter functions

- Usually distinct conditions on different boundary parts
→ Filters in TorchPhysics allow sampling on specific parts
- Checks if point on correct part
→ Returning True or False (vectorized)

Filter functions

- Usually distinct conditions on different boundary parts
→ Filters in TorchPhysics allow sampling on specific parts
- Checks if point on correct part
→ Returning True or False (vectorized)

```
1 def filter_positive(x):  
2     return x[:, :1] > 0  
3 sampler_1 = tp.samplers.RandomUniformSampler(omega.boundary,  
4                                             100, filter_fn=filter_positive)
```

Filter functions

- Usually distinct conditions on different boundary parts
→ Filters in TorchPhysics allow sampling on specific parts
- Checks if point on correct part
→ Returning True or False (vectorized)

```
1 def filter_positive(x):  
2     return x[:, :1] > 0  
3 sampler_1 = tp.samplers.RandomUniformSampler(omega.boundary,  
4                                             100, filter_fn=filter_positive)  
5  
6 def filter_negative(x):  
7     return x[:, :1] < 0  
8 sampler_2 = tp.samplers.RandomUniformSampler(omega.boundary,  
9                                             100, filter_fn=filter_negative)  
10  
11 fig = tp.utils.scatter(X, sampler_1, sampler_2)
```

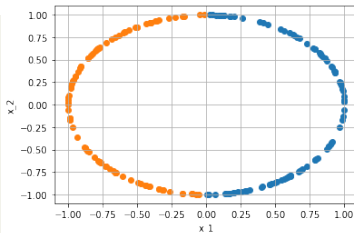

Filter functions

- Usually distinct conditions on different boundary parts
→ Filters in TorchPhysics allow sampling on specific parts
- Checks if point on correct part
→ Returning True or False (vectorized)

```

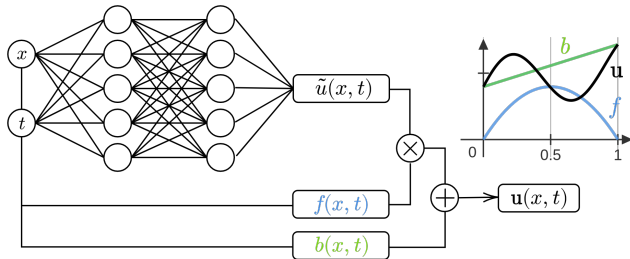
1 def filter_positive(x):
2     return x[:, :1] > 0
3 sampler_1 = tp.samplers.RandomUniformSampler(omega.boundary,
4                                                100, filter_fn=filter_positive)
5
6 def filter_negative(x):
7     return x[:, :1] < 0
8 sampler_2 = tp.samplers.RandomUniformSampler(omega.boundary,
9                                                100, filter_fn=filter_negative)
10
11 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

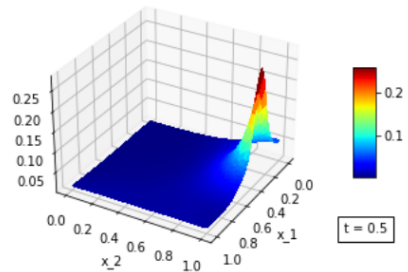
- Include boundary conditions in network architecture
- Dirichlet conditions: f is zero at boundary, b defines values on boundary

^aLu et al: *Physics-Informed Neural Networks with Hard Constraints for Inverse Design*, SIAM Journal on Scientific Computing, 2021

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

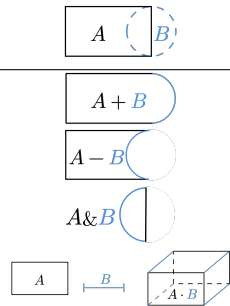


Domain creation

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

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

	$A \cup B$	$A \cap B$
union	$C = A + B$	$C = A \& B$
difference	$C = A - B$	$C = A * B$
intersection	$C = A \& B$	
cartesian product	$C = A * B$	



Time-dependent domains

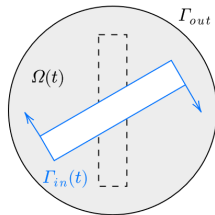
- A domain may depend on parameters from another domain
- Resolved when cartesian product is created

First option:

```

1 def corner1(t):
2     return rotation_matrix(t) * start_position_1
3
4 bar = tp.domains.Parallelogram(X, corner1, corner2, corner3)
5 circle = tp.domains.Circle(X, center, radius)
6
7 omega = circle - bar
8 t_int = tp.domains.Interval(T, 0, 1)
9
10 omega_t = omega * t_int
11 sampler = tp.samplers.RandomUniformSampler(omega_t, n_points=1000)

```



Time-dependent domains

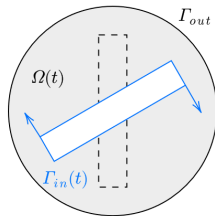
- A domain may depend on parameters from another domain
- Resolved when cartesian product is created

Second option:

```

1 def corner1(t):
2     return rotation_matrix(t) * start_position_1
3
4 bar = tp.domains.Parallelogram(X, corner1, corner2, corner3)
5 circle = tp.domains.Circle(X, center, radius)
6
7 omega = circle - bar
8 t_int = tp.domains.Interval(T, 0, 1)
9
10 sampler_omega = tp.samplers.GridSampler(omega, n_points=100)
11 sampler_t = tp.samplers.RandomUniformSampler(t_int, n_points=10)
12 sampler = sampler_omega * sampler_t

```



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 4: Change geometry
- Example 5: Time dependent geometry: drilling

