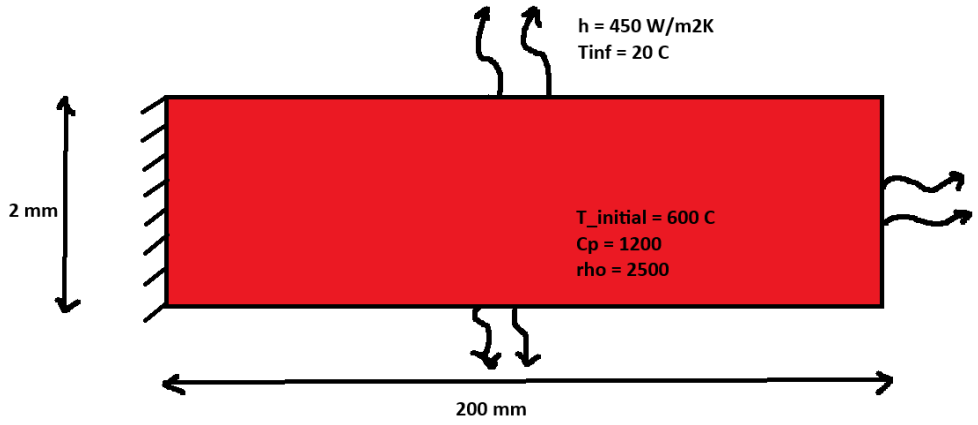
Adding Convective Boundary Conditions for a 2D Heat Diffusion Problem

Boundary conditions are fundamental to solving partial differential equations (PDEs), but not all are equally simple to implement. While Dirichlet boundary conditions are straightforward in NVIDIA Modulus, adding Neumann or convective boundary conditions can be a bit more challenging. During my internship at AnK, I explored and implemented two methods to incorporate convective boundary conditions into a 2D heat diffusion problem. Here's a walkthrough of my approach and insights.

Problem Definition

The task involved solving a 2D heat diffusion problem on a glass slab with convective boundary conditions applied to three walls (top, right, and bottom). The fourth wall (left) was treated with a zero-flux (Neumann) boundary condition.



This problem posed an excellent opportunity to dive deep into NVIDIA Modulus and understand its flexibility for modeling complex boundary conditions.

Two Methods for Adding Convective Boundary Conditions

1. Defining a Custom Class

For the left wall's zero-flux boundary condition, NVIDIA Modulus provides a built-in class, GradNormal, which calculates the gradient normal to the wall. This class can be used directly:

```
left_wall = PointwiseBoundaryConstraint(
    nodes=nodes,
    geometry=glass_slab,
    outvar={"normal_gradient_T": 0},
    batch_size=cfg.batch_size.LeftWall,
    criteria=Eq(x, -length / 2),
    parameterization=time_range,
)
heat_domain.add_constraint(left_wall, "left_wall")
```

For convective boundary conditions, however, I needed to customize the class to include the convection term. I extended GradNormal to implement the boundary condition equation:

$$-k\frac{\partial T}{\partial \vec{n}} = h(T - T_\infty)$$

Here's the critical addition:

```
self.equations["BC_variable_" + self.T] = (
    self.equations["normal_gradient_" + self.T] + ((self.h * (T - self.T_inf)) / self.K)
)
```

This allowed me to apply the boundary condition as:

```
top_wall = PointwiseBoundaryConstraint(
    nodes=nodes,
    geometry=glass_slab,
    outvar={"BC_variable_T": 0},
    batch_size=cfg.batch_size.TopWall,
    criteria=Eq(y, height / 2),
    parameterization=time_range,
)
heat_domain.add_constraint(top_wall, "top_wall")
```

By defining my custom class, I achieved full control over the boundary condition implementation.

---

2. Modifying the Diffusion Interface Class

The second approach involved tweaking the existing DiffusionInterface class in NVIDIA Modulus. Originally designed to handle boundary conditions at interfaces between two media, this class can be repurposed for our convective boundary conditions.

Key modifications included:

1. Simplifying the variables for a single medium.
2. Adapting the flux equations to include convection:

$$-k\frac{\partial T}{\partial \vec{n}} = h(T - T_\infty)$$

Updated code snippet:

```
# variables to match the boundary conditions (example Temperature)
        T_1 = Function(T_1)(*input_variables)
        # T_2 = Function(T_2)(*input_variables)
------------------------------------------------------------


        # set equations
        self.equations = {}
        self.equations["diffusion_interface_dirichlet_" + self.T_1 + "_" + "ambient"] = (
            T_1 - 20
        )
        flux_1 = self.k * (
            normal_x * T_1.diff(x) + normal_y * T_1.diff(y) + normal_z * T_1.diff(z)
        )
        flux_2 = self.h * (
            20 - T_1
        )
        self.equations["diffusion_interface_neumann_" + self.T_1 + "_" + "ambient"] = (
            flux_1 - flux_2
        )
```

With these adjustments, I used the DiffusionInterface class to apply convective boundary conditions seamlessly.

Lessons Learned

While both methods worked, each had its trade-offs:

- **Custom Class:** Offers flexibility and control but requires more coding.
- **Modified Interface:** Easier to implement but may not be as intuitive for highly customized conditions.

By exploring both, I gained a deeper understanding of how NVIDIA Modulus handles PDEs and boundary conditions. These methods provide a robust framework for solving real-world heat transfer problems, such as those encountered in glass forming or thermal stress analysis.

Takeaways

Adding convective boundary conditions in NVIDIA Modulus is an excellent exercise in balancing theoretical understanding with practical implementation. Whether you prefer customizing from scratch or adapting existing frameworks, Modulus offers the tools to tackle complex simulations efficiently.

Feel free to reach out if you have questions or insights to share—let's keep learning!