2D Periodic boundary logic

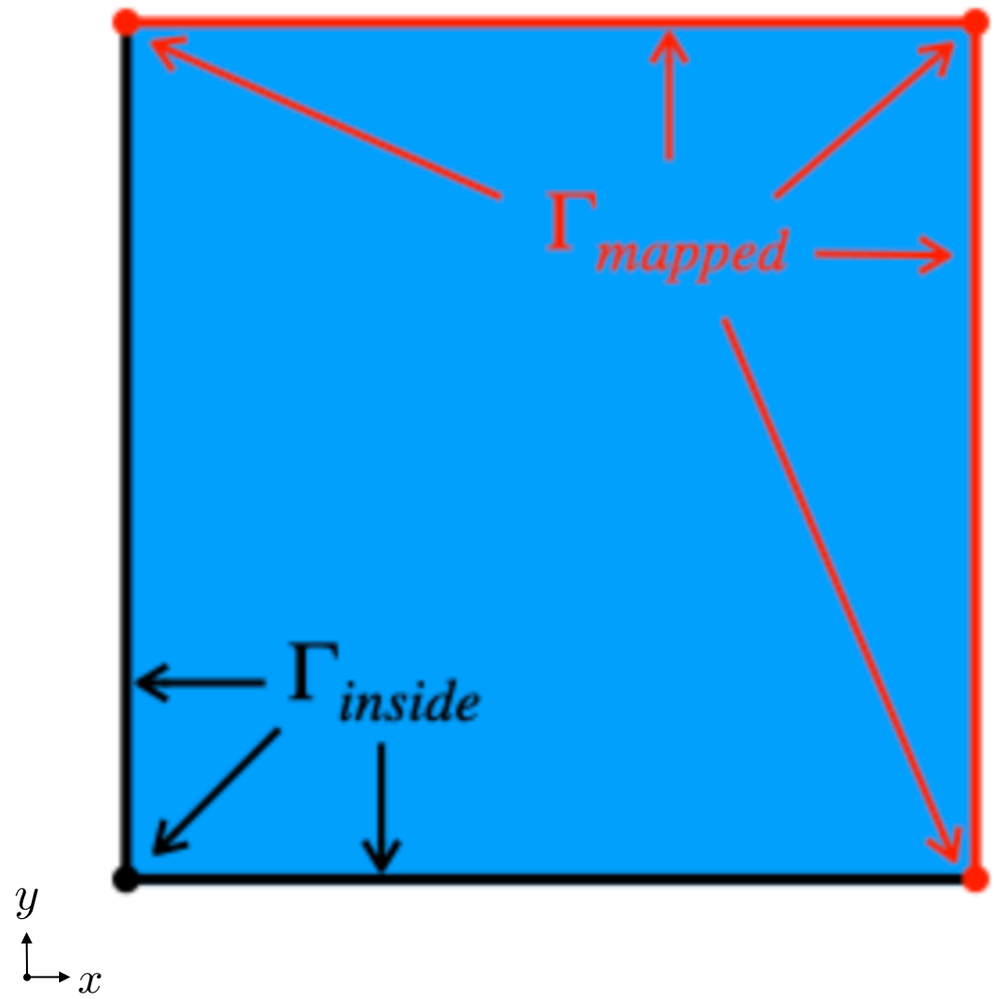
Source:

https://github.com/akosmrlj/FEniCS_tutorial/blob/master/CahnHilliard/CahnHilliard.ipynb

PeriodicBoundary() needs two functions:

1. inside() defines what points are on Γ_{inside}.
   (These points are ignored during mapping)

2. map() describes the mapping from Γ_{mapped} to Γ_{inside}.



2D Periodic boundary logic

Interface:

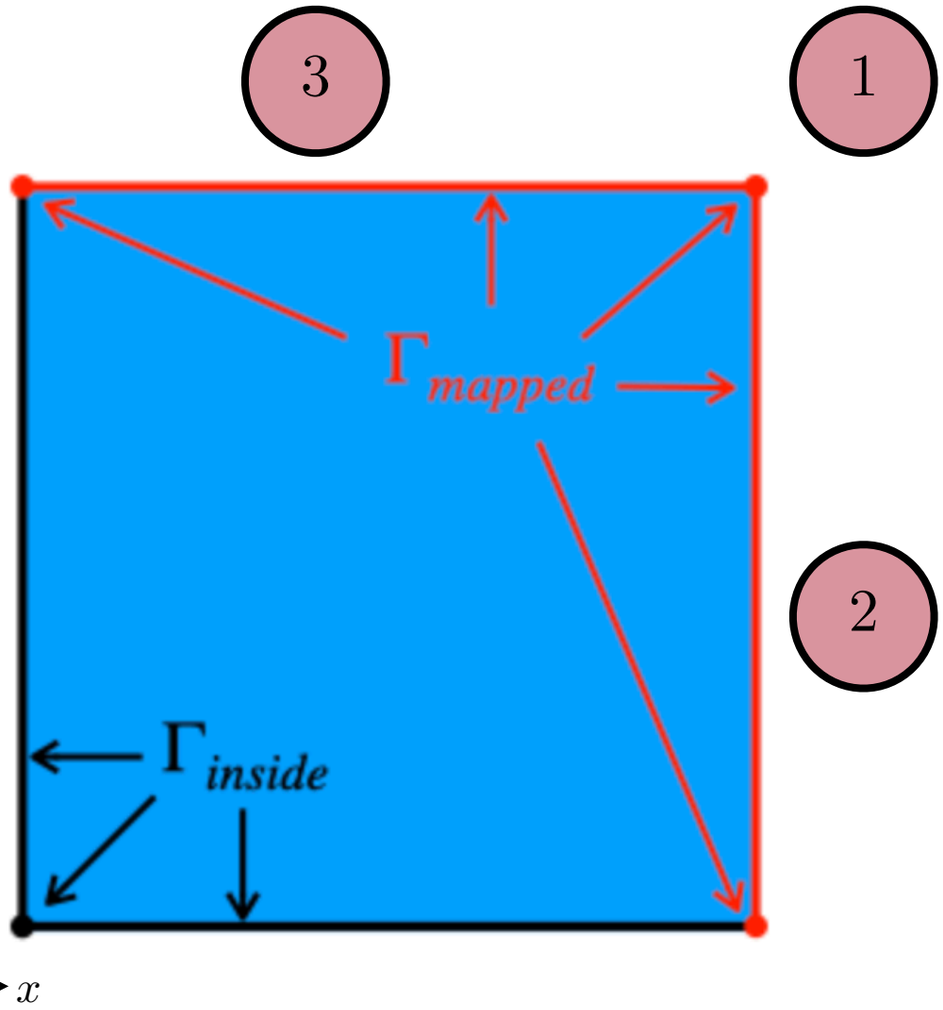
The interface between Γ_{inside} and Γ_{mapped} should be excluded from Γ_{inside}.

In 2-D, this interface consists of the two points (0,1) and (1,0).

Mapping:

The three regions of Γ_{mapped} are then mapped according to:

1. Point (1,1) is mapped to (0,0)
2. Edge (x=1) is mapped to edge (x=0)
3. Edge (y=1) is mapped to edge (y=0)



1

2

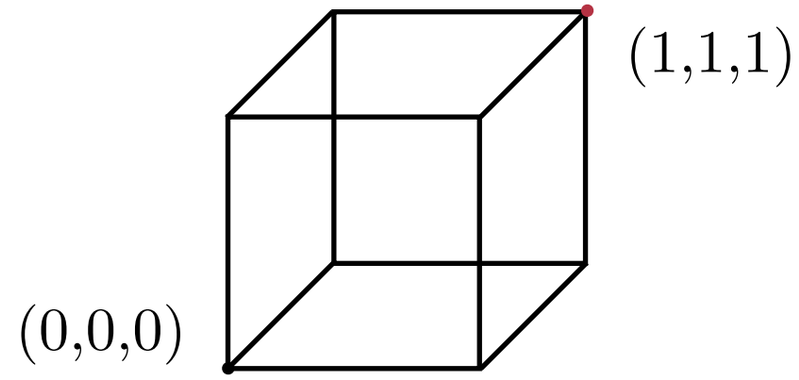
3

```
def map(self, x, y):
    if near(x[0], 1) and near(x[1], 1):
        y[0] = x[0] - 1.
        y[1] = x[1] - 1.
    elif near(x[0], 1):
        y[0] = x[0] - 1.
        y[1] = x[1]
    else:   # near(x[1], 1)
        y[0] = x[0]
        y[1] = x[1] - 1.
```

3D Periodic boundary logic

Similar logic to 2D case applies in 3D, except that:

1. The interface between Γ_{inside} and Γ_{mapped} consists of edges, not points.
2. The mapping from Γ_{mapped} to Γ_{inside} is of course different.

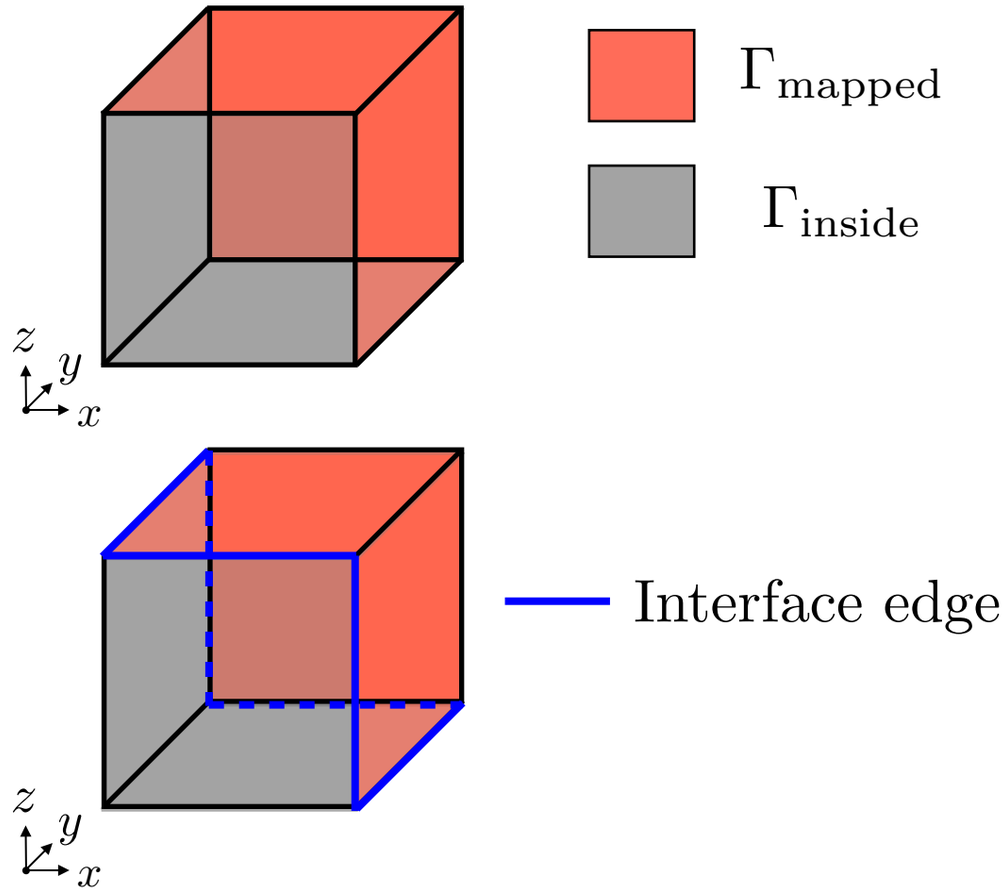


Interface

The interface between Γ_{inside} and Γ_{mapped}, marked in blue, consists of 6 edges:

1. x=1, z=0
2. y=1, z=0
3. y=1, x=0
4. z=1, x=0
5. z=1, y=0
6. x=1, y=0

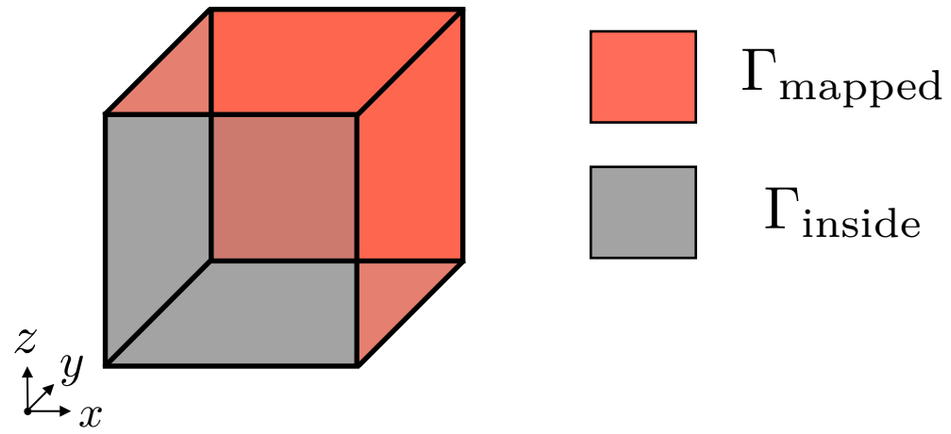
As in 2D, these edges are excluded from Γ_{inside}.



3D Periodic boundary logic

Mapping

The mapping logic from Γ_{mapped} to Γ_{inside} is then straightforward, accounting for each possible location in Γ_{mapped}.




```
def map(self, x, y):
    if near(x[0], 1) and near(x[1], 1) and near(x[2], 1):
        y[0] = x[0] - 1.
        y[1] = x[1] - 1.
        y[2] = x[2] - 1.
    elif near(x[0], 1) and near(x[1], 1):
        y[0] = x[0] - 1.
        y[1] = x[1] - 1.
        y[2] = x[2]
    elif near(x[0], 1) and near(x[2], 1):
        y[0] = x[0] - 1.
        y[1] = x[1]
        y[2] = x[2] - 1.
    elif near(x[1], 1) and near(x[2], 1):
        y[0] = x[0]
        y[1] = x[1] - 1.
        y[2] = x[2] - 1.
    elif near(x[0], 1):
        y[0] = x[0] - 1.
        y[1] = x[1]
        y[2] = x[2]
    elif near(x[1], 1):
        y[0] = x[0]
        y[1] = x[1] - 1.
        y[2] = x[2]
    else:   # near(x[2], 1)
        y[0] = x[0]
        y[1] = x[1]
        y[2] = x[2] - 1.
```

The point (1,1,1)

The edge x=1, y=1

The edge x=1, z=1

The edge y=1, z=1

The face x=1

The face y=1

The face z=1