Chapter 1

FEniCS

1.1 Poisson example

For a simple example to see how FEniCS works consider the Poisson equation. Defining $\Omega \subset \mathbb{R}^n$ to be the domain and the boundary as $\partial\Omega = \Gamma_D \cup \Gamma_N$ where Γ_D and Γ_N correspond to the Dirichlet and Neumann boundaries respectively. Thus, the Poisson equation with corresponding boundary conditions reads as:

$$\begin{aligned} -\Delta u &= f \quad \text{in } \Omega \\ u &= 0 \quad \text{in } \Gamma_D \\ \nabla u \cdot n &= g \quad \text{in } \Gamma_N \end{aligned} \tag{1.1.1}$$

For this example we will be considering the following function, domains and boundaries to be:

- $f = (x - 1/2)^2 - 4(y - 1/2)^4$
- $g = 5x$
- $\Omega = [0, 1] \times [0, 1]$
- $\Gamma_D = \{(0, y) \cup (1, y) \quad | \quad 0 \leq y \leq 1\}$
- $\Gamma_N = \{(x, 0) \cup (x, 1) \quad | \quad 0 \leq x \leq 1\}$

Given an appropriate function space $\mathcal{V}$, then the weak formulation of (1.1.1) is given by

$$a(u,v) = L(v) \quad \text{for } u, v \in \mathcal{V}, \tag{1.1.2}$$

where

$$a(u,v) = \int_\Omega \nabla u \cdot \nabla v \, dx \quad \text{and} \quad L(v) = \int_\Omega f v dx + \int_{\Gamma_N} g v \, ds. \tag{1.1.3}$$

Defining mesh and function space

For this example we want to consider a uniform unit square mesh. This is created by using the following python code

```
1 mesh = UnitSquareMesh(32,32)
```

Once the mesh is create we no turn to the function space. For this model problem we want to consider the function space $\mathcal{V}$ to be:

$$\mathcal{V} = \{v \in H_1(\Omega) : \vec{v} = \vec{v}_0 \text{ on } \partial\Omega\},$$

this is done using

```
1 V = FunctionSpace(mesh,'CG',1)
```

Defining subdomains

Separating different parts of the boundary is easily done within FEniCS. For this our simple Possion model the boundary is split up into two parts. The top and bottom defines the Dirichlet boundary condition whilst the left and right portions of the boundary define the Neumann conditions. Before partitioning the domain into the Dirichlet and Neumann boundaries we need to define the different sections of the boundary. This is done using classes as follows:

```
# Defining boundary classes
class Left(SubDomain):
    def inside(self, x, on_boundary):
        return near(x[0], 0.0)

class Right(SubDomain):
    def inside(self, x, on_boundary):
        return near(x[0], 1.0)

class Bottom(SubDomain):
    def inside(self, x, on_boundary):
        return near(x[1], 0.0)

class Top(SubDomain):
    def inside(self, x, on_boundary):
        return near(x[1], 1.0)

# Initialize sub-domain instances
left = Left()
top = Top()
right = Right()
bottom = Bottom()
```

Using the classes above we group the Dirichlet and Neumann boundaries together. These can then be used to set up the boundary conditions for our problem.

```
boundaries = FacetFunction("size_t", mesh)
boundaries.set_all(0)
left.mark(boundaries, 1)
right.mark(boundaries, 1)
top.mark(boundaries, 2)
bottom.mark(boundaries, 2)
```

Boundary Conditions and weak formulation

The boundaries have been split into their separate domains so now we can think about defining the boundary conditions, bilinear and linear (1.1.2) forms.

First consider the homogeneous Dirichlet boundary:

$$u = 0 \quad \text{in } \Gamma_D.$$

The Dirichlet boundary condition is created by using the DirichletBC class. When defining Dirichlet boundary conditions on separate subdomains we need to use four imports to the DirichletBC class. The first one is the function space that the boundary condition applies to, the value on the boundary and then the last two arguments define which part of the boundary the boundary condition is defined on.

```
u0 = Constant(0.0)
bc = DirichletBC(V, u0, boundaries,1)
```

To define the variational form of the problem we first need to determine the trial function u and test function v belonging to the function space $\mathcal{V}$. Furthermore, the source term f and Neumann condition g are required in the linear form $L(v)$. Both are defined using the Expression class. Together with the trial and test functions the variational form can be created using the following code snippet:

```
# Define variational problem
u = TrialFunction(V)
v = TestFunction(V)
f = Expression("(pow(x[0]-0.5,2)-4*pow(x[1]-0.5,4))")
g = Expression("(5*x[0])")
a = inner(grad(u), grad(v))*dx
L = f*v*dx + g*v*ds(2)
```

Assembly and solving

Now the variational (or weak form) have been stated the next step is to solve the problem. This can be done in two was within FEniCS

1. Assemble system and solve
2. Solve without assembly.

```
u = Function(V)
A, b = assemble_system(a,L,bc)
solve(A,u.vector(),b)
```

```
u = Function(V)
solve(a == L, u, bc)
```

Visualisation

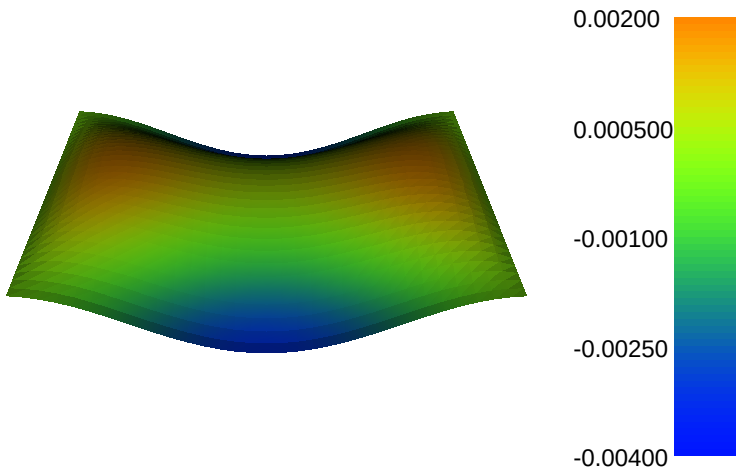

Figure 1.1: Numerical solution to (1.1.1)