Tutorial on Firedrake: solving 2D Poisson equation

Hanfeng Zhai
hzhai@stanford.edu

February 25, 2025

Introduction

This tutorial demonstrates how to solve the Poisson equation using the finite element method (FEM) with the Firedrake library. The Poisson equation is a widely used partial differential equation (PDE) that models physical phenomena such as heat conduction, electrostatics, and diffusion.

Mathematical Formulation

The Poisson equation in two dimensions is written as:

$$-\nabla \cdot (k\nabla u) = f \quad \text{in } \Omega, \tag{1}$$

where:

- u is the unknown scalar field (e.g., temperature).
- k is the thermal conductivity (assumed constant in this example).
- f is the source term (e.g., heat generation). $f = 1.0$ in this example.
- Ω is the computational domain ($\mathbb{R}^2$).

The boundary conditions are defined as:

$$u = g_D \quad \text{on } \Gamma_D, \tag{2}$$

$$-k\frac{\partial u}{\partial n} = g_N \quad \text{on } \Gamma_N, \tag{3}$$

where Γ_D and Γ_N are Dirichlet and Neumann boundaries, respectively, and n is the outward normal vector.

In this example, we solve Equation (1) with Dirichlet boundary conditions on all boundaries.

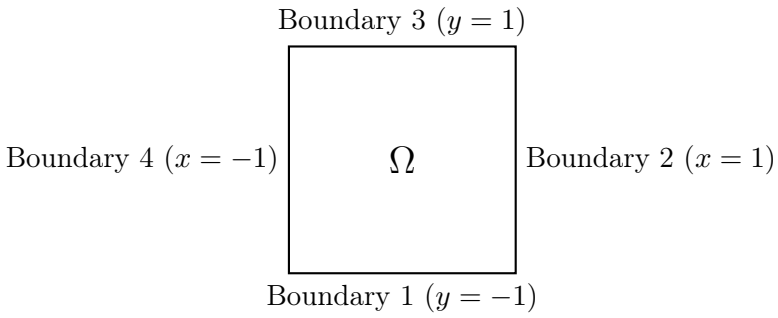
Implementation in Firedrake

The following Python code implements the solution of the Poisson equation using Firedrake. The computational domain is a square, and the Dirichlet boundary conditions set $u = 0$ on all edges of the domain. The source term is constant, $f = 1.0$, and $k = 1.0$. The solution is visualized as a heatmap and as a 3D surface plot.

Schematic of the Domain

The computational domain is a square defined as $[-1, 1] \times [-1, 1]$. The boundaries are labeled as follows:

- Boundary 1: $y = -1$,
- Boundary 2: $x = 1$,
- Boundary 3: $y = 1$,
- Boundary 4: $x = -1$.



Key Steps in the Code

1. **Mesh Generation:** The domain is discretized using a triangular mesh generated by Gmsh and converted for use in Firedrake.
2. **Boundary Conditions:** Dirichlet boundary conditions are applied on all edges of the square.
3. **Variational Formulation:** The weak form of the Poisson equation is derived as:

$$\int_\Omega k \nabla u \cdot \nabla v \, \mathrm{d}x = \int_\Omega f v \, \mathrm{d}x, \tag{4}$$

   where v is the test function.
4. **Solution Computation:** The linear system resulting from the discretization is solved, yielding the scalar field u.
5. **Visualization:** The solution is plotted as a 2D heatmap and a 3D surface.

[ ]:
```
from google.colab import drive
drive.mount('/content/drive', force_remount=True)

try:
    from firedrake import *
except ImportError:
    !wget "https://fem-on-colab.github.io/releases/firedrake-install-real.sh" -O␣
 ↪"/tmp/firedrake-install.sh"
    !bash "/tmp/firedrake-install.sh"
    from firedrake import *

!apt-get install gmsh
!pip install --upgrade gmsh
```

```
Mounted at /content/drive
Reading package lists... Done
Building dependency tree... Done
Reading state information... Done
gmsh is already the newest version (4.8.4+ds2-2build1).
0 upgraded, 0 newly installed, 0 to remove and 22 not upgraded.
Requirement already satisfied: gmsh in /usr/local/lib/python3.11/dist-packages
(4.13.1)
```

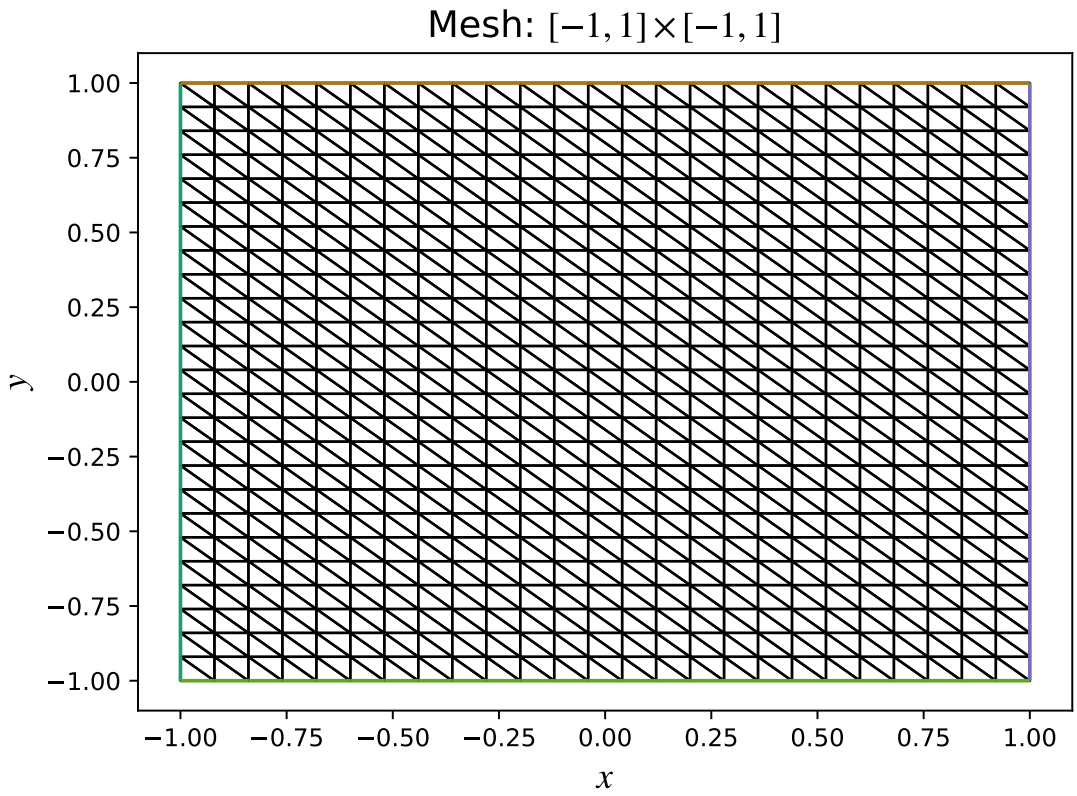
[ ]:
```
from firedrake import *
import matplotlib.pyplot as plt
from firedrake.pyplot import triplot

# Create a rectangular mesh on [0,2] x [0,2] (using 40x40 cells)
nx = 25
ny = 25
mesh = RectangleMesh(nx, ny, 2, 2)

# Shift the mesh coordinates from [0,2] to [-1,1] in both x and y directions
mesh.coordinates.dat.data[:] -= 1.0

# Plot the mesh
plt.figure(figsize=(5,5))
triplot(mesh)
plt.title(r"Mesh: $[-1,1] \times [-1,1]$", fontsize=15)
plt.xlabel(r"$x$", fontsize=15)
plt.ylabel(r"$y$", fontsize=15)
plt.tight_layout(); plt.savefig('mesh.pdf')
plt.show()
```

```
<Figure size 500x500 with 0 Axes>
```



[ ]:
```
# Define the function space (continuous Galerkin of order 1)
V = FunctionSpace(mesh, "CG", 1)

# Define trial and test functions
u = TrialFunction(V)
v = TestFunction(V)

# Define the source term: f = 1
f = Constant(1.0)

# Define the bilinear form and linear functional
# The weak form for Poisson is: int grad(u)·grad(v) dx = int f v dx
a = dot(grad(u), grad(v)) * dx
L = f * v * dx
```

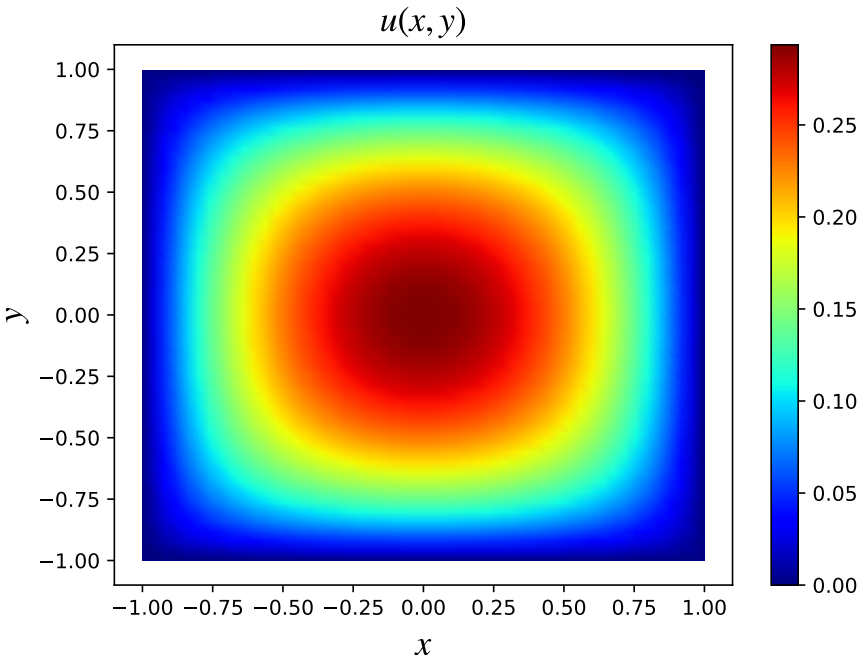
[ ]:
```
# Impose homogeneous Dirichlet boundary conditions (u = 0 on the entire boundary)
bc = DirichletBC(V, 0.0, "on_boundary")
```

[ ]:
```
# Compute the solution u_sol
u_sol = Function(V)
```

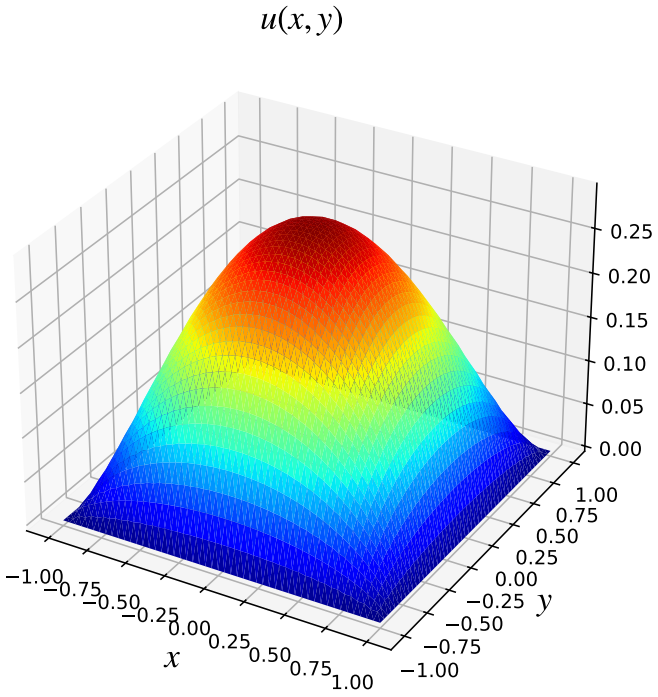
```
solve(a == L, u_sol, bcs=bc)
```

[ ]:
```
import matplotlib.pyplot as plt
from firedrake.pyplot import tripcolor, trisurf
plt.rcParams.update({"text.usetex": False, "mathtext.fontset": "stix"})
plt.figure(figsize=(5,5))
mappable = tripcolor(u_sol, cmap='jet')
plt.colorbar(mappable)
plt.title(r"$u(x,y)$", fontsize=17.5)
plt.xlabel(r"$x$", fontsize=17.5)
plt.ylabel(r"$y$", fontsize=17.5)
plt.tight_layout(); plt.savefig('u_sol_firedrake.pdf')
plt.show()
```

```
<Figure size 500x500 with 0 Axes>
```



[ ]:
```
fig = plt.figure(figsize=(8,6))
ax = fig.add_subplot(111, projection='3d')
trisurf(u_sol, axes=ax, cmap='jet')
ax.set_title(r"$u(x,y)$", fontsize=17.5)
ax.set_xlabel(r"$x$", fontsize=17.5)
ax.set_ylabel(r"$y$", fontsize=17.5)
plt.savefig('u_sol_firedrake_3D.pdf')
plt.show()
```



The code can be accessed via Google Colab.

Summary

This coding procedure outlined a systematic approach to simulating and visualizing heat conduction in a square domain using Python. The key steps included:

- Defining the computational domain with clear specifications for the grid and material properties.
- Applying boundary conditions to model the physical constraints accurately.
- Solving the governing equations using numerical methods for heat transfer.
- Visualizing the results to gain insights into the temperature distribution across the domain.

Students are encouraged to experiment with the provided framework by modifying the boundary conditions, such as changing the fixed temperatures or implementing insulated boundaries. Observing the resulting changes in temperature distribution provides a deeper understanding of how boundary conditions influence the system's behavior. This iterative process fosters critical thinking and reinforces concepts of heat transfer and numerical modeling.