Lecture 02
Solving the Poisson equation

Anders Logg

May 18, 2018

What will you learn?

- *How to solve the Poisson equation*
- *How to formulate the boundary value problem*
- *How to derive the variational problem*
- *FEniCS programming*
  - Creating meshes
  - Defining function spaces
  - Defining boundaries and boundary conditions
  - Defining expressions
  - Defining variational problems
  - Solving linear variational problems
  - Posprocessing solutions
- *Exercise*

How to formulate the boundary value problem

Partial differential equation

The Poisson equation

$$-\Delta u = f \quad \text{in } \Omega$$

Partial differential equation

The Poisson equation

$$-\Delta u = f \quad \text{in } \Omega$$

u is the solution to be computed

f is a given source term

Ω is the computational domain

Δ is the *Laplacian* operator which in 2D has the form

$$\Delta u = \frac{\partial^2 u}{\partial x^2} + \frac{\partial^2 u}{\partial y^2}$$

Boundary conditions

Dirichlet boundary condition

$$u = u_{\mathrm{D}} \quad \text{on } \Gamma_{\mathrm{D}} \subseteq \partial\Omega$$

Boundary conditions

Dirichlet boundary condition

$$u = u_{\mathrm{D}} \quad \text{on } \Gamma_{\mathrm{D}} \subseteq \partial\Omega$$

Neumann boundary condition

$$-\partial_n u = g \quad \text{on } \Gamma_{\mathrm{N}} \subseteq \partial\Omega$$

Boundary conditions

Dirichlet boundary condition

$$u = u_{\mathrm{D}} \quad \text{on } \Gamma_{\mathrm{D}} \subseteq \partial\Omega$$

Neumann boundary condition

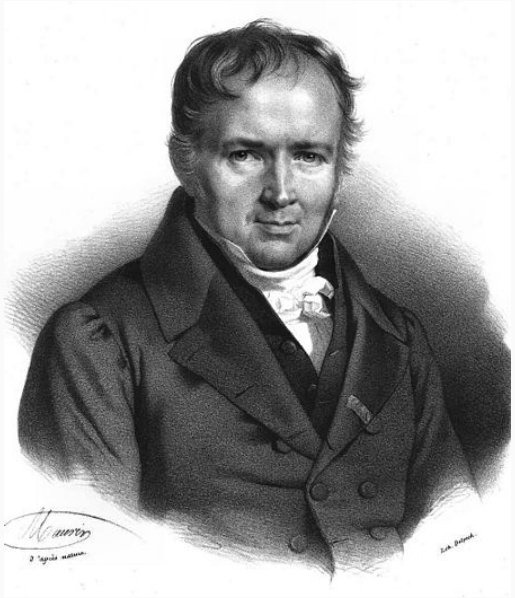
$$-\partial_n u = g \quad \text{on } \Gamma_{\mathrm{N}} \subseteq \partial\Omega$$

The Dirichlet condition $u = u_{\mathrm{D}}$ is also called a *strong* boundary condition

The Neumann condition $-\partial_n u = g$ is also called a *natural* boundary condition

Application examples

- Heat conduction, electrostatics, diffusion of substances, twisting of elastic rods, inviscid fluid flow, water waves, magnetostatics, ...
- As part of more complicated systems of PDEs, for example the incompressible Navier–Stokes equations or the Einstein field equations



Siméon Denis Poisson (1781–1840)
French mathematician, engineer and physicist

How to derive the variational problem

Recall from Lecture 1: The FEM cookbook

Partial differential equation (strong form)

$$\mathcal{A}u = f \tag{1}$$

Continuous variational problem (weak form)

Find $u \in V$ such that

$$a(u, v) = L(v) \quad \forall v \in V \tag{2}$$

Discrete variational problem (finite element method)

Find $u_h \in V_h$ such that

$$a(u_h, v) = L(v) \quad \forall v \in V_h \tag{3}$$

Discrete system of equations (linear system)

$$AU = b \tag{4}$$

From strong to weak form: (1) $\rightarrow$ (2)

Partial differential equation (strong form)

$$-\Delta u = f \quad \text{in } \Omega$$

From strong to weak form: (1) $\rightarrow$ (2)

Partial differential equation (strong form)

$$-\Delta u = f \quad \text{in } \Omega$$

Multiply by a test function v and integrate over the domain Ω:

$$\int_\Omega -\Delta u \, v \, \mathrm{d}x = \int_\Omega f v \, \mathrm{d}x$$

From strong to weak form: (1) $\rightarrow$ (2)

Partial differential equation (strong form)

$$-\Delta u = f \quad \text{in } \Omega$$

Multiply by a test function v and integrate over the domain Ω:

$$\int_\Omega -\Delta u \, v \, \mathrm{d}x = \int_\Omega f v \, \mathrm{d}x$$

Integrate by parts:

$$\int_\Omega \nabla u \cdot \nabla v \, \mathrm{d}x - \int_{\partial\Omega} \underbrace{\nabla u \cdot n}_{=\partial_n u} \, v \, \mathrm{d}s = \int_\Omega f v \, \mathrm{d}x$$

From strong to weak form: (1) $\rightarrow$ (2) / cont.

Note that $\partial\Omega = \Gamma_{\mathrm{D}} \cup \Gamma_{\mathrm{N}}$:

$$-\int_{\partial\Omega} \partial_n u \, v \, \mathrm{d}s = -\int_{\Gamma_{\mathrm{D}}} \partial_n u \underbrace{v}_{=0} \, \mathrm{d}s + \int_{\Gamma_{\mathrm{N}}} \underbrace{-\partial_n u}_{=g} \, v \, \mathrm{d}s$$

$$= \int_{\Gamma_{\mathrm{N}}} g v \, \mathrm{d}s$$

From strong to weak form: (1) $\rightarrow$ (2) / cont.

Note that $\partial\Omega = \Gamma_\mathrm{D} \cup \Gamma_\mathrm{N}$:

$$-\int_{\partial\Omega} \partial_n u \, v \, \mathrm{d}s = -\int_{\Gamma_\mathrm{D}} \partial_n u \underbrace{v}_{=0} \mathrm{d}s + \int_{\Gamma_\mathrm{N}} \underbrace{-\partial_n u}_{=g} v \, \mathrm{d}s$$

$$= \int_{\Gamma_\mathrm{N}} g v \, \mathrm{d}s$$

$\Rightarrow$ Continuous variational problem (weak form)

Find $u \in V^\mathrm{D}$ such that

$$\underbrace{\int_\Omega \nabla u \cdot \nabla v \, \mathrm{d}x}_{=a(u,v)} = \underbrace{\int_\Omega f v \, \mathrm{d}x - \int_{\Gamma_\mathrm{N}} g v \, \mathrm{d}s}_{=L(v)}$$

for all $v \in V^0$

Trial and test spaces

The trial space

$$V^{\mathrm{D}} = \{v \in H^1(\Omega) \mid v = u_{\mathrm{D}} \text{ on } \Gamma_{\mathrm{D}}\}$$

$u \in V^{\mathrm{D}}$ is called the *trial function*

Trial and test spaces

The trial space

$$V^{\mathrm{D}} = \{v \in H^1(\Omega) \mid v = u_{\mathrm{D}} \text{ on } \Gamma_{\mathrm{D}}\}$$

$u \in V^{\mathrm{D}}$ is called the *trial function*

The test space

$$V^0 = \{v \in H^1(\Omega) \mid v = 0 \text{ on } \Gamma_{\mathrm{D}}\}$$

$v \in V^0$ is called the *test function*

From weak form to finite element method: (2) $\rightarrow$ (3)

Let V_h^{D} be a discrete finite element subspace of V^{D}

Let V_h^0 be a discrete finite element subspace of V^0

From weak form to finite element method: (2) $\rightarrow$ (3)

Let V_h^{D} be a discrete finite element subspace of V^{D}

Let V_h^0 be a discrete finite element subspace of V^0

$\Rightarrow$ Discrete variational problem (finite element method)

Find $u_h \in V_h^{\mathrm{D}}$ such that

$$\underbrace{\int_\Omega \nabla u_h \cdot \nabla v \,\mathrm{d}x}_{=a(u_h,v)} = \underbrace{\int_\Omega f v \,\mathrm{d}x - \int_{\Gamma_{\mathrm{N}}} g v \,\mathrm{d}s}_{=L(v)}$$

for all $v \in V_h^0$

From finite element method to linear system: (3) $\rightarrow$ (4)

From Lecture 1, we know that

$$A_{ij} = a(\phi_j, \phi_i)$$
$$b_i = L(\phi_i)$$

From finite element method to linear system: (3) $\rightarrow$ (4)

From Lecture 1, we know that

$$A_{ij} = a(\phi_j, \phi_i)$$
$$b_i = L(\phi_i)$$

It follows that the stiffness matrix A and load vector b for Poisson's equation are given by

$$A_{ij} = \int_\Omega \nabla\phi_j \cdot \nabla\phi_i \,\mathrm{d}x$$
$$b_i = \int_\Omega f\phi_i \,\mathrm{d}x - \int_{\Gamma_\mathrm{N}} g\phi_i \,\mathrm{d}s$$

From finite element method to linear system: (3) → (4)

From Lecture 1, we know that

$$A_{ij} = a(\phi_j, \phi_i)$$
$$b_i = L(\phi_i)$$

It follows that the stiffness matrix A and load vector b for Poisson's equation are given by

$$A_{ij} = \int_\Omega \nabla\phi_j \cdot \nabla\phi_i \,\mathrm{d}x$$
$$b_i = \int_\Omega f\phi_i \,\mathrm{d}x - \int_{\Gamma_\mathrm{N}} g\phi_i \,\mathrm{d}s$$

⇒ Discrete system of equations (linear system)

$$AU = b$$

Summary

Variational problem: $a(u, v) = L(v)$

$$a(u, v) = \int_\Omega \nabla u \cdot \nabla v \, \mathrm{d}x$$

$$L(v) = \int_\Omega f v \, \mathrm{d}x - \int_{\Gamma_\mathrm{N}} g v \, \mathrm{d}s$$

Linear system: $AU = b$

$$A_{ij} = \int_\Omega \nabla \phi_j \cdot \nabla \phi_i \, \mathrm{d}x$$

$$b_i = \int_\Omega f \phi_i \, \mathrm{d}x - \int_{\Gamma_\mathrm{N}} g \phi_i \, \mathrm{d}s$$

FEniCS programming

First steps: Running Python

Editor + terminal

- Edit `foo.py` in editor
- Run command `python foo.py` in terminal

Python terminal

- Run command `python`
- Execute Python commands in Python terminal

Jupyter notebook

- Start Jupyter notebook
- Edit, run, document, share code and results in notebook

First steps: Importing FEniCS

Import everything from FEniCS:

```
from fenics import *
```

Import select functionality from FEniCS:

```
from fenics import FunctionSpace
V = FunctionSpace(...)
```

Import FEniCS and use `fenics` prefix:

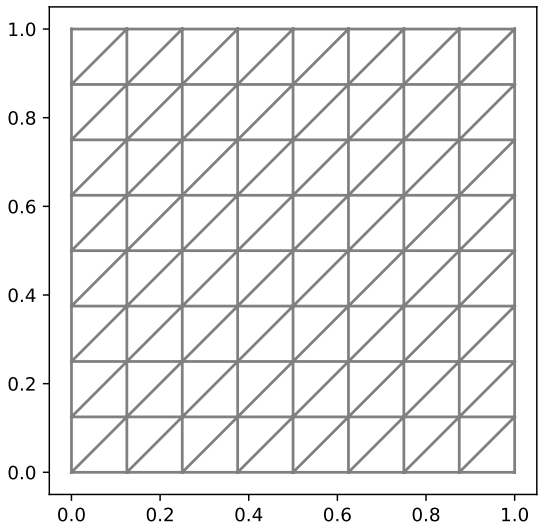
```
import fenics
V = fenics.FunctionSpace(...)
```

Creating simple meshes

Simple meshes can be created using built-in mesh generators in FEniCS:

```
mesh = UnitSquareMesh(8, 8)
```

This creates a mesh of the unit square with $8 \times 8 \times 2 = 128$ triangles



Built-in meshes

The following mesh generators are availabe in FEniCS:

- `UnitIntervalMesh`
- `UnitSquareMesh`
- `UnitCubeMesh`
- `IntervalMesh`
- `RectangleMesh`
- `BoxMesh`

Creating complex meshes

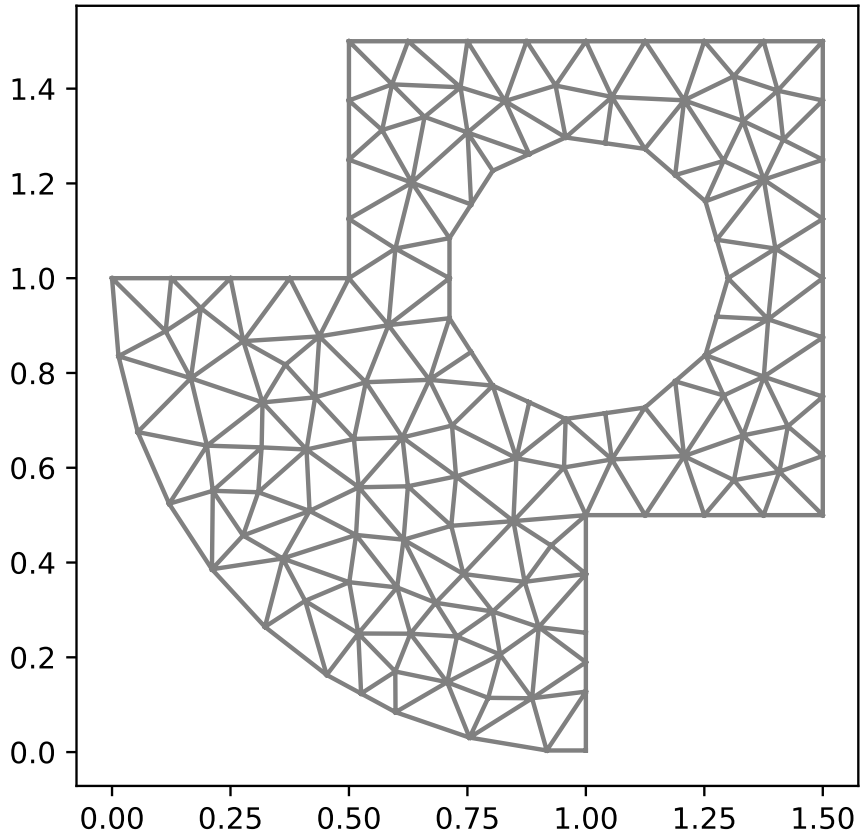
More complex meshes can be generated using Constructive Solid Geometry (CSG) through the FEniCS component **mshr**:

```
from fenics import *
from mshr import *
r0 = Rectangle(Point(0, 0), Point(1, 1))
r1 = Rectangle(Point(0.5, 0.5), Point(1.5, 1.5))
c0 = Circle(Point(1, 1), 0.3)
c1 = Circle(Point(1, 1), 1)
g = ((r0 + r1) - c0)*c1
mesh = generate_mesh(g, 10)
```

Creating complex meshes / contd.

This corresponds to the following mathematical set operations:

$$\Omega = ((R_0 \cup R_1) \setminus C_0) \cap C_1$$



Defining function spaces

Function spaces are defined using the `FunctionSpace` class:

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

Other examples:

```
V = FunctionSpace(mesh, 'P', 2)
V = FunctionSpace(mesh, 'P', 3)

V = VectorFunctionSpace(mesh, 'P', 1)

V = FunctionSpace(mesh, 'DG', 0)

V = FunctionSpace(mesh,'BDM',2)
V = FunctionSpace(mesh,'Nedelec 1st kind H(curl)',3)
```

Defining boundaries

Boundaries can be defined using Python functions:

```
def boundary(x, on_boundary):
    return x[0] == 1.0
```

Other examples:

```
def boundary(x):
    return x[0] < DOLFIN_EPS or \
           x[1] > 1.0 - DOLFIN_EPS

def boundary(x):
    return near(x[0], 0.0) or near(x[1], 1.0)

def boundary(x, on_boundary):
    return on_boundary and x[0] > DOLFIN_EPS
```

Defining boundary conditions

Dirichlet boundary conditions are defined using the `DirichletBC` class:

```
bc = DirichletBC(V, uD, boundary)
```

This states that a function in the function space defined by `V` should be equal to `uD` on the boundary defined by `boundary`

Note that the above line does not yet apply the boundary condition to all functions in the function space

Defining expressions

Expressions are useful for defining coefficients, right-hand sides or boundary values

Expressions are defined using the `Expression` class:

```
f = Expression('exp(-sin(pi*x[0])*cos(pi*x[1]))',
               degree=2)
```

Note that the formula must be written in C++ syntax and the polynomial degree must be specified

For more information:

```
from fenics import *
help(Expression)
```

Defining variational problems

Recall the variational problem for Poisson's equation:

$$a(u, v) = \int_\Omega \nabla u \cdot \nabla v \, \mathrm{d}x$$

$$L(v) = \int_\Omega f v \, \mathrm{d}x - \int_{\Gamma_\mathrm{N}} g v \, \mathrm{d}s$$

FEniCS implementation:

```
u = TrialFunction(V)
v = TestFunction(V)
f = Expression(...)
g = Expression(...)
a = dot(grad(u), grad(v))*dx
L = f*v*dx - g*v*ds
```

Solving linear variational problems

Variational problems are solved by calling the `solve()` function:

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

Note the reuse of the variable name `u` as both a `TrialFunction` in the variational problem and a `Function` to store the solution

Postprocessing solutions

Add the following incantation on top (after importing FEniCS)

```
%matplotlib inline
```

Plot the solution and the mesh:

```
plot(u)
plot(mesh)
```

For postprocessing in ParaView, store the solution in VTK format:

```
file = File('solution.pvd)
file << u
```

Exercise

Exercise 2: Solving the Poisson equation

In this exercise, we will solve the Poisson equation with FEniCS:

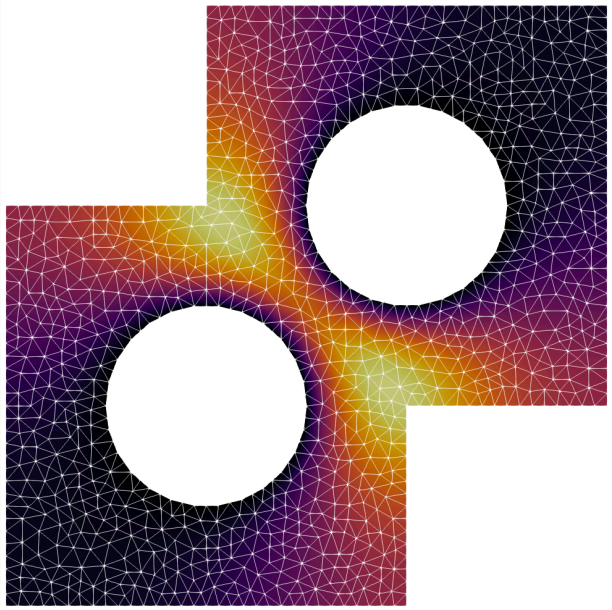
$$
\begin{aligned}
-\Delta u &= f && \text{in } \Omega \\
u &= u_{\mathrm{D}} && \text{on } \Gamma_{\mathrm{D}} \\
-\partial_n u &= g && \text{on } \Gamma_{\mathrm{N}}
\end{aligned}
$$

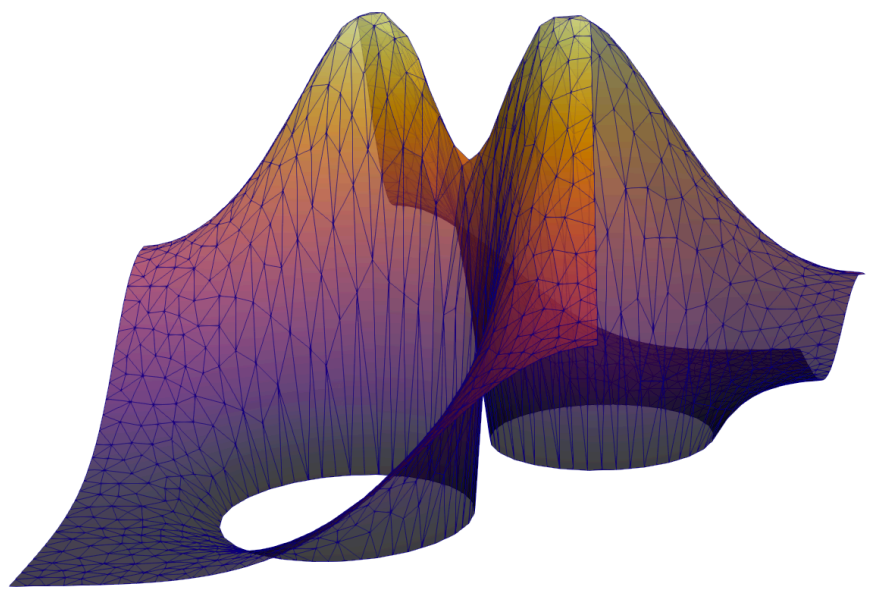
Write a FEniCS program to compute and plot the solution, and save the solution to file for visualization in Paraview!

Exercise 2: Problem data

- $\Omega = (S_0 \cup S_1) \setminus (C_0 \cup C_1)$
- S_0 is the unit square centered at $(0.5, 0.5)$
- S_1 is the unit square centered at $(1, 1)$
- C_0 is a circle with radius 0.25 centered at $(0.5, 0.5)$
- C_1 is a circle with radius 0.25 centered at $(1, 1)$
- $\Gamma_{\mathrm{D}} =$ the boundaries of the two circles
- $\Gamma_{\mathrm{N}} = \partial\Omega \setminus \Gamma_{\mathrm{D}}$
- $f(x, y) = \exp(-10((x - 0.75)^2 + (y - 0.75)^2))$
- $u_{\mathrm{D}} = 0$
- $g = 0$

Exercise 2: Solution





Left: *Solution to Exercise 2 plotted in Paraview using the "Inferno" colormap*

Right: *Solution to Exercise 2 plotted in Paraview using the "Warp By Scalar" filter, the "Inferno' colormap and 75% opacity*