NGCM6001 Simulation and Modelling Computer Lab 1

Aims

The aim of this computer laboratory is to gain a practical insight into the fundamental steps required to solve a partial differential equation (PDE) over a simple geometric domain using a finite element approach. In doing so, you will learn:

- How to recast the PDE problem into a variational form
- How to define simple domains
- How to deal with Dirichlet boundary conditions
- How to numerically solve the variational problem.
- How to compute derived quantities from the solution.
- How to visualise the mesh, the solution and derived quantities

The symbolic-numeric environment chosen for this laboratory is FEniCS (http://fenicsproject.org/) which is powerful and user-friendly tool for solving PDEs.

Basic problem: the Poisson equation

The Poisson equation is encountered in a wide range of domains in physics from heat conduction, electrostatics, diffusion and elasticity. The Poisson problem is defined as follows:

$$\begin{aligned} -\nabla^2 u(\boldsymbol{x}) &= f(\boldsymbol{x}) \quad \boldsymbol{x} \text{ in } \Omega \\ u(\boldsymbol{x}) &= u_0(\boldsymbol{x}) \quad \boldsymbol{x} \text{ on } \partial\Omega \end{aligned} \qquad (1)$$

Here, $u(\boldsymbol{x})$ is the unknown function, f and $\boldsymbol{u}_0$ are prescribed functions, $\nabla^2 = \Delta$ is the Laplace operator, Ω is the spatial domain and $\partial\Omega$ is its boundary.
A *time-independent* or *stationary* PDE like the Poisson equation combined with boundary conditions (1)constitute a **boundary-value problem**.

FEnics workflow

Solving a physical problem with FEnics involves the following workflow:

1. Identify the PDE and its boundary conditions
2. Recast the PDE problem into a variational problem
3. Write a Python programme encoding the :
    a. variational problem
    b. definition of input data: e.g. f and $\boldsymbol{u}_0$
    c. finite element mesh for the spatial domain Ω:
4. Add statements to the Python programme for:
    a. solving the variational problem
    b. computing derived quantities such as the gradient of the unknown function
    c. visualising the results

Variational formulation

The core of the recipe for turning a PDE into a variational problem is to multiply the PDE by a function v, integrate the resulting equation over Ω, and perform integration by parts of terms with second-order derivatives.

The function v which multiplies the PDE is in the mathematical finite element literature called a **test function**. The unknown function u to be approximated is referred to as a **trial function**.
The terms test and trial function are used in FEniCS programs too.

The test function v is required to vanish on the parts of the boundary where u is known

Suitable function spaces must be specified for the test and trial functions. For standard PDEs arising in physics and mechanics such spaces are well known.

$$-\int_{\Omega} \nabla^2 u(\boldsymbol{x}) v \,\mathrm{d}\boldsymbol{x} = \int_{\Omega} f(\boldsymbol{x}) v \,\mathrm{d}\boldsymbol{x} \tag{2}$$

Question 1:

Apply integration by parts to the PDE $-\nabla^2 u(\boldsymbol{x}) = f(\boldsymbol{x})$ with the divergence theorem (relation between the flow (flux) of a vector field though a surface and the behavior inside the surface) recast it into its variational form. You will also take into account the conditions about the test function v on the boundary where u is prescribed.

You have now derived the variational formulation of the Poisson problem. This variational form is what is commonly called a **weak form** of the boundary-value problem ("weak" because of the reduced continuity requirements placed on u). It contains the basic equation $-\nabla^2 u(\boldsymbol{x}) = f(\boldsymbol{x})$ and the condition $u = u_0 \text{ on } \partial\Omega$.

This continuous variational problem must be discretised to be solved using the finite element method. The continuous version of the unknown function u will be denoted by u_e (the index *e* stands for "exact") to make the distinction with the discretised solution of the problem that is typically denoted by u_h.
NB: In the context of FEnics, u will be meant to represent the discretised approximate solution of the weak form.

For a linear weak form like the one just established it proves convenient to introduce the following notation:

$$a(u,v) = \int_{\Omega} \nabla u . \nabla v \,\mathrm{d}\boldsymbol{x} \tag{3}$$

$$L(v) = \int_{\Omega} f v \,\mathrm{d}\boldsymbol{x} \tag{4}$$

Where a and L are respectively a **bilinear** and **linear operator**.

The weak form of the Poisson problem (or any linear weak form) can therefore be expressed as:

$$a(u,v) = L(v) \tag{5}$$

For every linear problem to be solved:

- all the terms containing the unknown u must be collected in $a(u,v)$
- all terms with only known functions must be collected in $L(v)$

The explicit formulas for a and L are then coded in the programme.
Solving the discretised weak form consists in finding:

$$u \in V \text{ such that } a(u,v) = L(v) \quad \forall\, v \in \widehat{V} \tag{6}$$

ne

$V = \{v \in H^1(\Omega) : v = u_0 \text{ on } \partial\Omega\}$ is the space of **trial** functions

$\widehat{V} = \{v \in H^1(\Omega) : v = 0 \text{ on } \partial\Omega\}$ is the space of **test** functions

Specifying V and $\widehat{V}$ consists in choosing the mesh and the type of interpolation functions.

Practical implementation on a concrete example

Here we are going to particularise the Poisson problem (reproduced here for convenience):

$$\begin{aligned} -\nabla^2 u(\boldsymbol{x}) &= f(\boldsymbol{x}) \quad \boldsymbol{x} \text{ in } \Omega \\ u(\boldsymbol{x}) &= u_0(\boldsymbol{x}) \quad \boldsymbol{x} \text{ on } \partial\Omega \end{aligned} \tag{7}$$

This decision means that $\{u_0, f, \Omega\}$ needs to be defined.
We choose a simple 2D domain: the unit square $\Omega = [0,1]\times[0,1]$.

In order to compare the finite element solution to the exact solution we also *a priori* select a particular form for u. We choose:

$$u_e(x,y) = 1 + x^2 + 2y^2 \tag{8}$$

If we inject equation (8) into (7) we find that u_e is solution if:

$$f(x,y) = -6; \qquad u_0(x,y) = u_e(x,y) = 1 + x^2 + 2y^2 \tag{9}$$

Question 2:
Using FEnics write an algorithm to solve the boundary-value problem described above.

1. Create and visualise the mesh using first-order (linear) triangular elements. You will use (6,4) as arguments for the mesh function which means that the computational domain will be first divide into 24 rectangles which will be further split into triangles
2. Set up the boundary conditions and visualise them
3. Solve the problem
4. Plot the approximate solution over the domain using colour plots.
5. Plot the error between the exact and approximate solution over the domain using colour plots
6. Calculate the maximum error for the whole domain
7. Calculate and plot the gradient of u

Python programming of the example
<insert FEnics tutorial demo program>
Stationary/poisson/d1_p2D.py

A more concrete example: a pressurised elastic membrane

Here, we study the deflection $D(x,y)$ of an elastic circular membrane with radius R, subject to a localized perpendicular pressure force, modelled as a Gaussian function. The PDE representing this physical process is:

$$-T\nabla^2 D = p(x,y) \ \text{ in } \Omega = \{(\mathrm{x},\mathrm{y})\,|\,\mathrm{x}^2 + \mathrm{y}^2 \leq R\} \tag{10}$$

where p is the external pressure defined as:

$$p(x,y) = \frac{A}{2\pi\sigma}\exp\left[-\frac{1}{2}\left(\frac{x-x_0}{\sigma}\right)^2 - \frac{1}{2}\left(\frac{y-y_0}{\sigma}\right)^2\right] \tag{11}$$

T the constant tension in the membrane, A the amplitude of the pressure, (x_0, y_0) the localisation of the Gaussian pressure function and σ its width.

You will use the following values:

$T = 10; A = 1; R = 0.3; \theta = 0.2; x_0 = 0.6R\cos(\theta); y_0 = 0.6R\sin(\theta); \sigma = 0.025$

Question 3:

1. Determine an analytical solution for equation (10)
2. Using the previously written Python programme, make appropriate modifications to solve the membrane problem.
3. Solve the problem
4. Calculate the maximum deflection
5. Plot the approximate solution over the domain using colour plots.
6. Plot the error between the exact and approximate solution over the domain using colour plots
7. Calculate the maximum error for the whole domain
8. Calculate and plot the gradient of u