Numerical Methods for Differential Equations
Exercise on the Finite Element Method

Numerical solution of the Poisson problem

Contents

1 Problem statement

Let us consider a domain $\Omega \subset \mathbb{R}^2$, with $\Gamma = \partial\Omega$ its boundary. This boundary is partitioned into two non-overlapping regions Γ_D and Γ_N where Dirichlet and Neumann boundary conditions, respectively, are applied. The strong form of the boundary value problem can be stated as: find u such that

$$\begin{aligned} -\frac{\partial^2 u}{\partial x^2} - \frac{\partial^2 u}{\partial y^2} &= f && \text{in } \Omega \\ u &= 0 && \text{on } \Gamma_D \end{aligned} \tag{1}$$

The domain Ω has a rectangular share defined as:

$$\Omega = \{(x, y) : -1 \leq x \leq 1, -1 \leq y \leq 1\}$$

In the problem (1), the source term f is:

$$f(x, y) = -4 + 2x^2 + 2y^2 \tag{2}$$

The boundary domain Γ coincides with Γ_D, i.e., there are no Neumann conditions. For this combination of problem and domain, the analytical solution is available and reads:

$$u\,(x, y) = x^2 + y^2 - x^2 y^2 - 1 \tag{3}$$

The purpose of this exercise is to solve the homogeneous boundary value problem (1) with the Finite Element Method (FEM) using triangular elements and linear basis functions.

Problem (1) governs, for instance, the equilibrium configuration of an elastic membrane with tension equal to one, fixed at the boundary, in a small displacement configuration and subject to an orthogonal force with intensity f. The function u describes the out-of-plane displacement of the membrane relative to the resting position $u = 0$ (see Fig. 1). In particular, we have:

- u [L]: out-of-plane displacement;
- $f = p/h$, where p [F/L^2] is the orthogonal distributed load, and h [F/L] is the radial tension applied on the boundaries. In our case, $h = 1$.

Dirichlet conditions determine how the membrane is fixed at the boundaries.

2 FEM solution

The boundary value problem (1) can be solved numerically using the FEM. The variational formulation leads to the Galerkin method.

The approximate solution u_h can be written as:

$$u_h\,(x, y) = \sum_{i=1}^{n} u_i \varphi_i\,(x, y) \tag{4}$$

where $\varphi_i(x, y)$, $i = 1, \ldots, n$ are suitable basis functions associated to the chosen discretization and u_i are the nodal values of u on n points in the domain Ω. These are the unknowns of our problem.

Substituting u_h in (1), we can write the residual

$$L\,(u_h) = -\frac{\partial^2 u_h}{\partial x^2} - \frac{\partial^2 u_h}{\partial y^2} - f \tag{5}$$

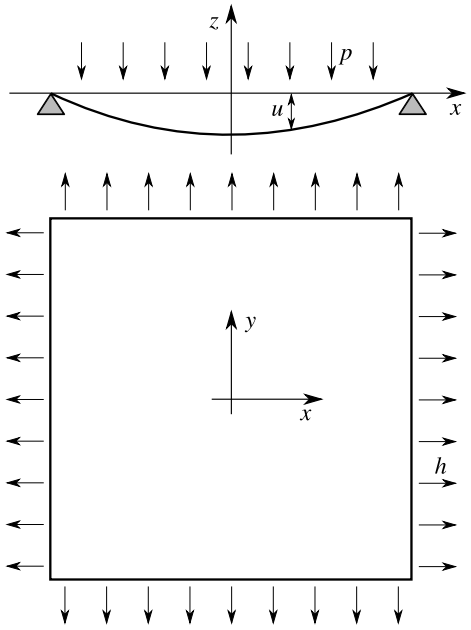

Figure 1: Elastic deformation of a square membrane.

and then impose the orthogonality to the n basis functions $\varphi_i(x, y)$, obtaining:

$$\int_\Omega L\left(u_h\right)\varphi_i(x,y)d\Omega = 0 \quad i = 1,\ldots,n \tag{6}$$

Using the divergence (Gauss) theorem, Eq. (6) becomes:

$$\int_\Omega \left(\frac{\partial u_h}{\partial x}\frac{\partial \varphi_i}{\partial x} + \frac{\partial u_h}{\partial y}\frac{\partial \varphi_i}{\partial y}\right) d\Omega - \int_\Gamma \left(\frac{\partial u_h}{\partial x}n_x + \frac{\partial u_h}{\partial y}n_y\right)\varphi_i d\gamma - \int_\Omega f\varphi_i d\Omega = 0 \tag{7}$$

Substituting the definition (4) into (7), we have:

$$\int_\Omega \left[\sum_{j=1}^{n}\left(\frac{\partial \varphi_j}{\partial x}\frac{\partial \varphi_i}{\partial x} + \frac{\partial \varphi_j}{\partial y}\frac{\partial \varphi_i}{\partial y}\right)u_j\right] d\Omega - \int_\Omega f\varphi_i d\Omega - \int_{\Gamma_q} q\varphi_i d\gamma = 0 \tag{8}$$

where q represent the known normal derivative $\partial u/\partial n$, i.e., the flux of u, on the Neumann boundary. In our problem, given that $\Gamma_N = \emptyset$, this contribution is lacking.

Eq. (8) are to be intended for any i in $1,\ldots,n$, thus they constitute a linear system of algebraic equations with n unknown values $u_1, u_2, \ldots, u_n$ that can be written as:

$$H\mathbf{u} - \mathbf{f} = \mathbf{0} \tag{9}$$

The chosen basis functions are piece-wise linear polynomials with local support. Thus, the generic coefficient h_{ij}, referring to the i-th equation, is the result of the assembly of local contributions $h_{ij}^{(e)}$ from all the finite elements sharing node i

$$h_{ij} = \sum_e h_{ij}^{(e)} = \sum_e \int_{\Omega^{(e)}} \left(\frac{\partial \varphi_j^{(e)}}{\partial x}\frac{\partial \varphi_i^{(e)}}{\partial x} + \frac{\partial \varphi_j^{(e)}}{\partial y}\frac{\partial \varphi_i^{(e)}}{\partial y}\right) d\Omega \tag{10}$$

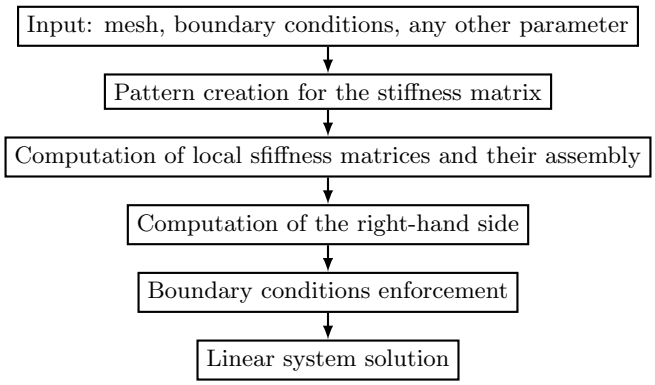

Figure 2: Flowchart of the code with the main steps.

while the right-hand side component f_i is:

$$f_i = \sum_e f_i^{(e)} = \sum_e \int_{\Omega^{(e)}} f\varphi_i^{(e)} d\Omega + \sum_e \int_{\Gamma_q^{(e)}} q\varphi_i^{(e)} \, d\gamma \tag{11}$$

where $\Gamma_q^{(e)}$ is provided for generality only, since it is null in the present study case.

From the definition of the single entry $h_{i,j}$ in Eq. (10), we can observe that H is a sparse symmetric positive definite matrix.

3 The FEM code

The boundary value problem (1) can be solved thanks to a code. Given the high level of complexity, this code has to exploit a well-defined structure, with different functions for different tasks, e.g., see the flowchart in Fig. 2.

3.1 Input data

Let's consider a non overlapping triangulation of Ω with triangular finite elements. This triangulation is defined with a table providing the topology for each triangle, i.e., the list of nodes in anticlockwise order. For instance:

```
 3   1   2
 3   4   1
20  21  13
 ⋮   ⋮   ⋮
```

means that the first element is identified by the vertices $(3, 1, 2)$, the second by $(3, 4, 1)$, the third by $(20, 21, 13)$ and so on. Finally, the position of each node is determined by two coordinates, e.g.:

```
      -1.0         -1.0
      -0.5         -1.0
-0.66122089  -0.65755367
         ⋮            ⋮
```

means that the first node has coordinates $(-1, -1)$, the second $(-0.5, -1)$, the third $(-0.66122089, -0.65755367)$, and so on.

Another required piece of information are the boundary conditions. Generally speaking, both the set of nodes located on the Dirichlet and Neumann boundaries have to be provided. In the specific application, we already know that $\Gamma_N = \emptyset$, thus only the list of Dirichlet nodes is given. For instance

```
4
8
13
⋮
```

means that 4, 8, 13, etc., are the nodes where the value of the solution, i.e., the Dirichlet boundary condition, have to be enforced.

Three ASCII files are used to represent a computational grid, characterized by different extensions:

1. `topol`: the file containing the triangle topology;
2. `coord`: the file containing the node coordinates;
3. `bound`: the file containing the set of Dirichlet nodes.

3.2 Pattern creation for the stiffness matrix

The pattern of a sparse matrix is the set of its nonzero entries. For the matrix H, this is determined by the nodal contacts in the computational grid. In other words, h_{ij} is different from zero if and only if one finite element shares both node i and j. Since H is stored in the CSC format (compressed sparse column), it is necessary to known the pattern in advance. The CSC format consists of three arrays:

- the row indices of the nonzeros;
- the column indices of the nonzeros;
- the pointer to first element of each column in the column indices array.

The pattern for H can be easily built from the topology input, i.e., the matrix `topol`. From `topol`, we have A, the adjacency matrix with `ne` rows and `nn` columns, where `ne` is the total number of elements and `nn` is the total number of nodes. $A(i, j)$ will be one only if the node j is part of the element i, otherwise, it is set to 0.

For instance, referring to the first rows of the aforementioned topology matrix, the data structure can be written as:

MATLAB	`A = sparse(row,col,1)`
python	`data = numpy.ones(numpy.size(row))` `A = scipy.sparse.csc_matrix((data,(row, col)))`

where `row` is the vector with the row indices:

$$\texttt{row} = [1, 1, 1, 2, 2, 2, 3, 3, 3, \dots]^T$$

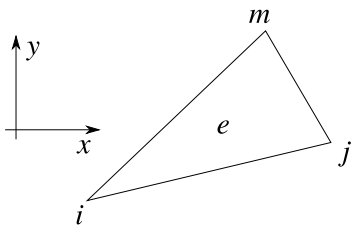

Figure 3: Triangular element with nodes i, j, m.

and `col` contains all the rows of `topol`:

$$\texttt{col} = [3, 1, 2, 3, 4, 1, 20, 21, 13, \dots]^T$$

Finally, the pattern for H is given by $A^T A$, i.e.,:

```
H = A' * A
```

MATLAB	`H = A' * A`
python	`H = A.transpose() * A`

provides the data structure to store H, with the proper sparsity.

3.3 Computation of local stiffness matrices and their assembly

The domain Ω is subdivided into triangular elements e. Let us name $u_h^{(e)}$ the solution restricted to the element e:

$$u_h^{(e)}(x, y) = \sum_{k=1}^{3} u_k(t)\varphi_k^{(e)}(x, y) \tag{12}$$

Referring to the generic finite element of Fig. 3, we have that the solution is linear with respect to the nodal unknowns u_i, u_j, and u_m. Those values are the solution at the corresponding nodes, since $\varphi_k(x_k, y_k) = 1$. To respect this constraint and be null on the other nodes, the basis functions are computed as:

$$\begin{aligned}
\varphi_i^{(e)}(x, y) &= \frac{a_i + b_i x + c_i y}{2\Delta} \\
\varphi_j^{(e)}(x, y) &= \frac{a_j + b_j x + c_j y}{2\Delta} \\
\varphi_m^{(e)}(x, y) &= \frac{a_m + b_m x + c_m y}{2\Delta}
\end{aligned} \tag{13}$$

where Δ is the element surface measure:

$$\Delta = \frac{1}{2} \begin{vmatrix} 1 & x_i & y_i \\ 1 & x_j & y_j \\ 1 & x_m & y_m \end{vmatrix},$$

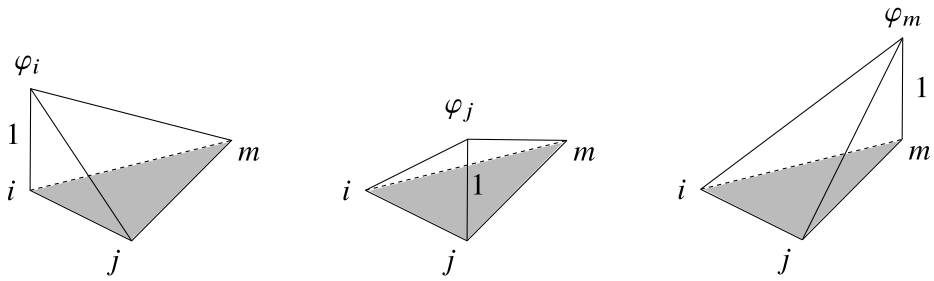

Figure 4: Linear basis functions for a generic triangular element.

and the coefficients a_i, b_i e c_i are given by:

$$\begin{aligned} a_i &= x_j y_m - x_m y_j \\ b_i &= y_j - y_m \\ c_i &= x_m - x_j \end{aligned}$$

The others are obtained thanks to an anticlockwise indices permutation:

$$a_j = x_m y_i - x_i y_m \qquad b_j = y_m - y_i \qquad c_j = x_i - x_m$$

$$a_m = x_i y_j - x_j y_i \qquad b_m = y_i - y_j \qquad c_m = x_j - x_i$$

The behavior of the basis functions $\varphi_k^{(e)}$ is shown in Fig. 4.

From Eq. (10), we have the expression of any entry of the stiffness matrix H:

$$h_{ij}^{(e)} = \int_{\Omega^{(e)}} \left[\frac{\partial \varphi_j^{(e)}}{\partial x} \frac{\partial \varphi_i^{(e)}}{\partial x} + \frac{\partial \varphi_j^{(e)}}{\partial y} \frac{\partial \varphi_i^{(e)}}{\partial y} \right] d\Omega = \frac{1}{4\Delta} (b_i b_j + c_i c_j) \tag{14}$$

Thus, the local stiffness matrix $H^{(e)}$ for a triangular element is:

$$H^{(e)} = \frac{1}{4\Delta} \left\{ \begin{bmatrix} b_i b_i & b_i b_j & b_i b_m \\ b_j b_i & b_j b_j & b_j b_m \\ b_m b_i & b_m b_j & b_m b_m \end{bmatrix} + \begin{bmatrix} c_i c_i & c_i c_j & c_i c_m \\ c_j c_i & c_j c_j & c_j c_m \\ c_m c_i & c_m c_j & c_m c_m \end{bmatrix} \right\} \tag{15}$$

Once the local contributions $H^{(e)}$ are available, they have to be assembled in the global stiffness matrix H, i.e., they have to summed to the existing values in the *right* position. The right position is given by the global numbering of the nodal indices i, j and m.

To simplify the computation, it is worth to create a dedicated function for the computation of the stiffness matrix. Given the topology and the coordinates, the function will loop over all elements, compute the local contributions and assemble them in the corresponding position. The output will be the assembled stiffness matrix H. Once `Hloc` is formed for the element k, the pseudo-code of Alg. 3.3 assembles it.

3.4 Computation of the right-hand side

Since there are only Dirichlet boundary conditions, the i-th component of the right-hand side f is:

$$f_i = \int_\Omega f(x,y) \varphi_i d\Omega \tag{16}$$

```
for i = 1, 3 do
    row = elem(k,i)
    for j = 1, 3 do
        col = elem(k,j)
        H(row,col) = H(row,col) + Hloc(i,j)
    end for
end for
```

where we remind that $f(x, y) = -4 + 2x^2 + 2y^2$. Thanks to the local nature of φ_i, the integral in (16) can be restricted from Ω to the set of triangles sharing node i. Since i is a sort of centroid of this element patch, we can assume the function f to be constant over this subdomain and equal to the value in the node i. Thus, $f(x_i, y_i)$ can be extracted from the integral (16), reading:

$$f_i \simeq f(x_i, y_i) \int_\Omega \varphi_i d\Omega = \left(-4 + 2x_i^2 + 2y_i^2\right) \frac{\sum_e \Delta_e}{3} \tag{17}$$

where the summation is extended to all the triangles sharing node i. The term $\sum_e \Delta_e/3$ is the surface measure relative to this node, i.e., one third of the area of the triangle patch surrounding node i. To compute this contribution, an array storing the surface measure for each element can be saved during the assembly loop, where Δ has to be determined, then, thanks to the connection list of the topology, these areas can be redistributed across all the nodes.

3.5 Boundary conditions enforcement

From a numerical viewpoint, to impose a Dirichlet boundary condition on a node means we have to set the function value on that location. To do so, we need to modify the equation relative to this node and to impose the known value. The new equation reads:

$$u_i = u_{D,i} \tag{18}$$

i.e., the i-th row of the stiffness matrix is composed by one value only, a diagonal 1, and the right-hand side is the fixed value $u_{D,i}$ for that node. This operation clearly destroys the symmetry of H. To restore this beneficial property, we can impose this condition in an approximate way, thanks to the *penalty* method. The diagonal term of the i-th row is substituted by a large value, i.e., $R_{max} = 10^{15}$, and the corresponding right-hand side component becomes $R_{max} u_{D,i}$. In our case, since $u_{D,i} = 0$ for any i in Γ_D, to change the diagonal term is enough. In this way, the Dirichlet condition would be exactly satisfied only when $R_{max} \to \infty$, but for practical applications, a value large enough with respect to the order of magnitude of the average coefficients works fine.

If there were Neumann boundary conditions, they would have contributed to the right-hand side.

3.6 Linear system solution

Since the stiffness matrix H is large, sparse and symmetric positive definite (SPD), a Krylov subspace iterative method can be efficiently used to solve the associated linear system. In particular, the Preconditioned Conjugate Method (PCG) is the reference method to solve the linear system (9). Among the simplest preconditioners, we remind Jacobi, i.e., the inverse of the diagonal elements, and the incomplete Cholesky factorization. The exit criterion is met when the residual norm is less then 10^{-8} times the right-hand side norm. The null vector is taken as initial solution.

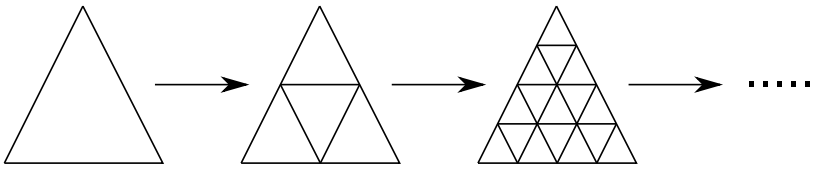

Figure 5: Refinement scheme for the computational grid.

3.7 Error computation

Once the FEM code is ready, the numerical solution can be validated through the analytical one, in particular, the FEM convergence can be studied. Refining the grid, as shown in Fig. 5, it is expected an error reduction. For this kind of refinement, where a node node is added for each edge midpoint and triangle centroid, the total number of triangles increases of a factor 4 and the characteristic length decreases by a factor 2 at each level. The Euclidean norm of the error decreases as ℓ^2, where ℓ represents the average mesh size, i.e., the characteristic length of the edges. The error norm reads:

$$\varepsilon = \sqrt{\int_\Omega \left(u_h - u\right)^2 d\Omega} \tag{19}$$

where u_h is the numeric solution and u is the analytical solution (3). The integral (19) can be computed thanks to the midpoint rule, i.e.:

$$\varepsilon \simeq \left\{ \sum_{i=1}^{n} \left[\left(u_i - u(x_i, y_i)\right)^2 \frac{\sum_e \Delta_e}{3} \right] \right\}^{1/2} \tag{20}$$

where the summation is meant over all the elements e sharing node i.

4 Requirements

Four subsequent refinement levels of the original computational grid are provided, for a total of five meshes. The candidate has to describe their development in a report, with this minimum set of requirements:

- semi-logarithmic convergence plots for the relative residual norm during the PCG iterations. Two plots with the convergence histories for all the refinement levels are required: one using Jacobi as preconditioner, the other with the incomplete Cholesky factorization.
- summary table for the FEM convergence, with the error value ε and the ratio between the error at the previous refinement level for each level.
- the source code, with appropriate comments for each function.