Exercise # 3. Numerical Solution of the Poisson Problem.

Alexandre Rodrigues (2039952)

January 24, 2022

1 Problem statement

The aim of this exercise is to solve the following homogeneous boundary value problem. Find u such that

$$-\frac{\partial^2 u}{\partial x^2} - \frac{\partial^2 u}{\partial y^2} = f \qquad \text{in } \Omega \tag{1}$$

$$u = 0 \qquad \text{on } \Gamma \tag{2}$$

valid in the domain $\Omega = \{(x, y) : -1 \leq x \leq, -1 \leq y \leq 1\}$, $\Gamma = \partial\Omega$ being its boundary.

The term f is:

$$f(x, y) = -4 + 2x^2 + 2y^2 \tag{3}$$

Only Dirichlet conditions apply so the analytical solution will be:

$$u(x, y) = x^2 + y^2 - x^2y^2 - 1 \tag{4}$$

This problem will be solved with the Finite Element Method using triangular elements and linear basis functions. It can simulate an elastic membrane fixed at the boundary, with small displacement and an orthogonal force with intensity f. The Dirichlet conditions define how the membrane is fixed and u is the out-of-plane displacement of the membrane.

2 Step-by-step Implementation

We can reduce the problem to a linear system.

$$H\mathbf{u} - \mathbf{f} = 0, \tag{5}$$

with n unknowns (vector u), n being the number of FEM points.

Based on the homework text I implemented each of these steps in Matlab.

1. **Input files from specified mesh:**

   All 3 files are loaded - `topol`, `bound` and `coord` - for the current mesh refinement level.

2. **Create pattern for the stiffness matrix:**

   1. Create a range vector 1 to `Ne`, then place it 3 times as a column in a matrix and reshape the matrix to obtain the column vector $row = [1, 1, 1, 2, 2, 2, 3, \ldots]^T$.

   2. The `col` vector is simply obtained by reshaping the `topol` matrix as a column vector.

   3. Compute the adjacency matrix as `A = sparse(row,col,1)` and the pattern for the stiffness matrix as `H = A' * A`, finally cleaning the matrix coefficients as we only want the sparse pattern, `H = H * 0`.

3. **Compute the stiffness matrix H:**

   I created the function `[H, delta] = computeStiff(H, topol, coord)` to encapsulate all the following computations. It has the topology and coordinates matrices as inputs, as well as the H matrix with its pattern already defined before. It returns the final H matrix and the `delta` vector with the surface measures of each element. Using a for loop for each element:

   1. Get coordinates of the 3 nodes that define the element, compute the surface measure for that element and save it in the `delta` vector.

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

   2. Compute the b and c coefficients of the basis functions (a is not needed):

$$b_i = y_j - y_m \qquad c_i = x_m - x_j,$$

   others are obtained using anticlockwise indices permutation.

   3. Compute `Hloc` for an element as

$$H_{loc} = \frac{1}{4\Delta}\{b^T b + c^T c\}$$

   where $b = [b_i, b_j, b_m]$ and $c = [c_i, c_j, c_m]$

   4. Assemble the stiffness matrix H from the local matrices using algorithm 3.3 of the exercise text.

4. **Compute the right hand size f :**

   Using a for loop to iterate in the list of nodes:

   1. Get the node coordinates;

   2. Find elements that have this node as a vertex and get their surface measures from the `delta` vector;

   3. Compute the right hand size vector as $f_i \approxeq (-4 + 2x_i^2 + 2y_i^2)\frac{\sum_e \Delta_e}{3}$

5. **Enforce the boundary conditions:**

   For this problem we can simply substitute the diagonal value of H at the node `i` of the boundary with a large enough constant, `H(i,i) = Rmax`, `Rmax` $= 10^{15}$ in this case.

6. **Solve the Linear System:**

   Following recommendations I used Matlab's PCG method with tolerance 1×10^{-8}, Jacobi preconditioner as `M = sparse(diag(diag(H)))` and Cholesky preconditioner as `L = ichol(H)`.

7. **Error computation:**

   Using a for loop to visit each node:

   1. Get the coordinates of the node;

   2. Compute the analytical solution as $u(x, y) = x^2 + y^2 - x^2y^2 - 1$;

   3. Sum surface measures of each element that have this node as a vertex;

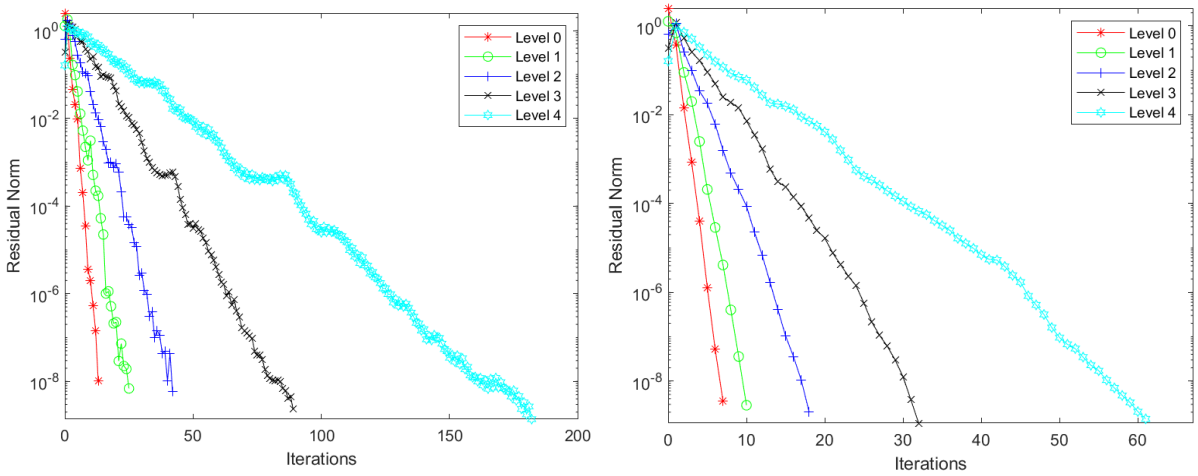
   4. Compute local error as $(u_i - u(x_i, y_i))^2 \ \frac{\sum_e \Delta_e}{3}$;

   5. Sum all the local errors and ϵ will be its square root.

3 Results

3.1 Convergence Plots

The following images show the convergence plots for all refinement levels, one for each preconditioner.



(a) Jacobi

(b) Cholesky

Figure 1: Convergence plots - Residual norms vs. iterations

There are clear differences in convergence regarding the preconditioner used. The Cholesky preconditioner significantly improves convergence by reducing the number of iterations needed by up to a factor of 3.

There is also a clear slowdown of convergence when increasing the level of refinement.

These results are as expected, increasing the preconditioner complexity improves efficiency of the PCG and increasing the level of refinement increases complexity of the FEM code and thus requires more computational time.

I also recorded the computational time needed to solve each PCG method but found no relevant differences.

3.2 Error

Level	**Error ϵ**	**Error Ratio**
0	6.912×10^{-2}	N/A
1	1.631×10^{-2}	0.236
2	3.984×10^{-3}	0.244
3	9.883×10^{-4}	0.248
4	2.465×10^{-4}	0.249

Table 1: FEM Convergence table

Each level of refinement decreases l by a factor of 2, $l_{i+1} = \frac{1}{2} l_i$. The error is proportional to l^2 so $\epsilon_{i+1} = \frac{1}{4}\epsilon_i$. So we can conclude that the results are as expected.

4 Conclusion

We can hereby conclude that the Finite Element Method is a good approach to solve these types of problems.

The method converged relatively fast, in less than 1 second. Using even more refined meshes would produce smaller errors and still be fast enough, although this does not take into account the mesh computation.

The best results were obtained from the most refined mesh and the Cholesky preconditioner but this scenario is also the slowest to finish when including the factorization.