ME 257: Homework 5 (Computing Assignment)

Rules: Code yourself. No copy/paste.
Coding is translation. Think first, code next, optimize last.
Post on piazza if you have questions. Have fun.

In this computing assignment, you will write a C/C++ code to compute the finite element approximation of the 2D model problem:

$$\begin{aligned} \Delta u + f &= 0 \quad \text{on } \Omega, \\ u &= 0 \quad \text{on } \partial\Omega, \end{aligned}$$

where $f : \Omega \to \mathbb{R}$ is the imposed forcing. For simplicity, we will assume that Ω is the unit square $[0,1] \times [0,1]$.

1. Using Matlab's pde toolbox or any other mesher of your choice, generate a mesh of triangles over the square domain Ω. For the sake of uniformity, save the mesh in the following format while numbering nodes from 0. `coordinates.dat`

   ```
   nNodes (number of nodes)
   x1 y1 (coordinates of node 1)
   x2 y2 (coordinates of node 2)
   ...
   xN yN (coordinates of node N)
   ```

   `connectivity.dat`

   ```
   nElements (number of elements)
   v1_1 v1_2 v1_3  (node numbers of triangle 1)
   v2_1 v2_2 v2_3  (node numbers of triangle 2)
   ....
   vM_1 vM_2 vM_3  (node numbers of triangle 3)
   ```

   `boundarynodes.dat`

   ```
   nBoundaryNodes (number of boundary nodes)
   n_1            (node on the boundary)
   ...
   ```

2. Define the function

```
ReadMesh(std::string coord_filename, std::string conn_filename, Mesh& M)
```

that reads the nodal coordinates and element connectivities into an instance of the mesh type defined as:

```
struct Mesh
{
  int                  nNodes;
  std::vector<double>  coordinates;
  int                  nElements;
  std::vector<int>     connectivity;
  std::vector<int>     bdNodes;
};
```

3. Define the function

```
int Local2GlobalMap(const Mesh& M, int elm_num, int loc_node_num)
```

that computes the local to global map, i.e., returns the global index corresponding to a given node of an element.

4. Define the function

```
void ComputeKe(const std::vector<double>& nodal_coords, double* Ke)
```

that computes the element stiffness matrix K_e using the nodal coordinates provided for an element.

5. Define the function

```
void ComputeFe(const std::vector<double>& nodal_coords, double* Fe)
```

that computes the element force vector F_e using the nodal coordinates provided for the element. For different choices of the forcing, you may hard-code the integrals.

6. Define the function

```
void SetDirichletBCs(const Mesh& M, EigenMatType& K, EigenVecType& F)
```

that imposes the Dirichlet boundary conditions by setting the n^{th} row of K and F to zero, followed by $K_{nn} = 1$, for each node n on the boundary.

7. Define the function

```
void PrintSolution(std::string filename, const Mesh& M, const double* U)
```

that prints the mesh and the computed solution in a format that can be used for visualization using Matlab's `pdeplot` routine.

8. In `main()`, do the following:

   (i) Read the mesh data structure.

   (ii) Initialize Eigen data structures for the stiffness matrix and the force vector assuming that the number of degrees of freedom equals the number of nodes in the mesh.

   (iii) Loop over elements in the mesh to assemble the global stiffness matrix and force vector using Eigen data structures.

(iv) Impose the Dirichlet boundary conditions.

(v) Solve the linear system $KU = F$ to compute the degrees of freedom.

(vi) Print the mesh and the computed solution to a file for visualization.

9. List down all the checks performed in your code to verify correctness (e.g., are the shape functions and derivatives coded right?).

10. Explain how Dirichlet boundary conditions are imposed. Does K remain symmetric after setting the constraints?

11. What solution do you expect for the choice $f = 0$? Does your code reproduce this solution?

12. Is it possible to choose f such that the finite element solution coincides with the exact solution?

13. Choose a forcing f for which you can work out the exact solution. Compute the finite element approximation u_h using a sequence of progressively (and uniformly) refined meshes.

14. For the solutions computed in part (8), plot the norm of the error measure

$$e = \left(\frac{1}{N_{\text{nodes}}} \sum_{a=1}^{N_{\text{nodes}}} (u(\mathbf{x}_a) - u_h(\mathbf{x}_a))^2 \right)^{1/2}$$

as a function of the mesh size on a log-log scale. Record your observations.