CAE of Sensors and Actuators
Date: December 2020
Assignment due: December 23, 2020 at 11:55 p.m.

Lehrstuhl für Sensorik
Universität Erlangen-Nürnberg
Daniel Kiefer

Assignment 0: Basic FEM Implementation – Part 1

Consider the boundary value problem (BVP) given by Poisson's equation in 1D (x-coordinate) and Dirichlet boundary conditions (DBCs) on both domain boundaries:

$$-ku''(x) = f(x) \quad \text{on } x \in \Omega = [a, b] \tag{1}$$
$$u(a) = g_a \tag{2}$$
$$u(b) = g_b\,. \tag{3}$$

The above partial differential equation (PDE) is also called the "steady-state heat equation in 1D", with k the thermal conductivity and $f(x)$ the heat-flux density. You have derived the corresponding weak form in the exercise class for $k = -1$. The aim of this assignment is to solve this equation by implementing your own FE code. For this assignment, we will consider only the case where

- $g_a = 0$ and $g_b = 0$ (homogeneous DBCs), and
- f is constant (independent of x).

General hints and remarks

Please make sure to:

- Save all relevant plots you generate (e.g., as `png`-file) while working on the assignment.
- Always label the axes of the plots and add a legend if necessary.
- Format your plots so that they can be read easily (e.g., appropriate font size).
- Prepare a `Results.pdf` file with the plots and short answers to all questions (label each answer with its task number) for submission.
- If you are unsure on how to perform the tasks, refer to
  - The Octave tutorial linked on StudOn
  - The first part of the assignment
  - The exercise notes

1 Preparation

1.1. Compute the analytical solution to the above BVP for constant f. This is done by integrating twice and determining the integration constants according to the BCs.

1.2. Write down the weak form of the above BVP

1.3. Use Galerkin's approximation to discretize the problem

1.4. Write down the expressions for the

1.4.1. entries K_{ij} of the global stiffness matrix $\mathbf{K}$

1.4.2. entries f_i of the global force vector $\mathbf{f}$

In the following,

- set $f(x) = 1$,
- use an equidistant mesh with element size h and
- use linear shape functions:

$$N_i(x) = \begin{cases} 0, & a \le x \le x_{i-1} \\ (x - x_{i-1})/h, & x_{i-1} < x \le x_i \\ (x_{i+1} - x)/h, & x_i < x \le x_{i+1} \\ 0, & x_{i+1} < x \le b\,. \end{cases}$$

Note that the above shape functions $N_i(x)$ have local support. Therefore, only a limited number of shape functions are non-zero on the domain $[x_e, x_{e+1}]$, which corresponds to the element e. In our case only two shape functions are involved, namely, N_e and N_{e+1}. This yields an *element stiffness matrix* $\mathbf{k}_\mathrm{e}$ of size 2×2 and an *element force vector* $\mathbf{f}_\mathrm{e}$ of size 2×1.

1.5. Split the integral in K_{ij} and f_i – which are over the whole domain – into a sum of integrals over each element domain.

1.6. Perform the integrations for one element and write down the

1.6.1. element stiffness matrix $\mathbf{k}_\mathrm{e}$ and

1.6.2. element force vector $\mathbf{f}_\mathrm{e}$

Note 1: These depend only on the element size h.
Note 2: They are the same for all elements because we use the same shape functions on each element and we have an equidistant mesh.

2 FEM Code

We will now use the above results to implement a FE program which solves the given BVP. The FE code should have the following general structure:

- generate mesh
  → in `generateMesh.m`
- compute element stiffness matrix $\mathbf{k}_\mathrm{e}$
  → in `elementStiffness.m`
- compute element force vector $\mathbf{f}_\mathrm{e}$
  → in `elementForce.m`
- assemble global matrices/vectors
  → in main script (`fem_equidistant_hdbc.m`)
- solve
  → in main script (`fem_equidistant_hdbc.m`)
- plot solution
  → in main script (`fem_equidistant_hdbc.m`)

2.1 The FE mesh

The mesh includes all information that is necessary and/or useful to describe the geometry and its discretization. To describe the mesh, use a structure (name the variable `mesh`) with the following fields:

- `nElems`: total number of elements

- `nNodes`: total number of nodes
- `h`: element size
- `x`: node coordinates as a vector of size $nNodes \times 1$
- `nodesOfElem`: array that maps the element index to its global node indices. It should be of size $nElems \times 2$ (because each element has two nodes). This array describes the connectivity between the nodes and will be used to assemble the global FE matrices.
- `nodesOfDBC`: vector of size 1×2 containing the indices of nodes at which Dirichlet boundary conditions are imposed.

2.1. Implement the function `mesh = generateMesh(domain, nElems)`, which generates and returns the structure described above. The arguments are

- `domain`: 1×2 vector containing the domain boundaries, i.e., [a, b]
- `nElems`: the desired number of elements on that domain

2.2 The element matrices/vectors

2.2. Implement the function `ke = elementStiffness(h)`. Given the element size `h`, it should return the element stiffness matrix $\mathbf{k}_\mathrm{e}$ as previously computed in 1.6.1..

2.3. Implement the function `fe = elementForce(h)` which returns the element force vector $\mathbf{f}_\mathrm{e}$ for $f(x) = 1$ as computed in 1.6.2..

2.3 Assemble the global matrices and vectors

2.4. Allocate memory for the global stiffness matrix $\mathbf{K}$ and the global force vector $\mathbf{f}$. What is their size?

2.5. Assemble the global stiffness matrix $\mathbf{K}$.

2.6. Assemble the global force vector $\mathbf{f}$.

Note 1: Assembling is done by iterating over each of the elements and adding the element matrix $\mathbf{k}_\mathrm{e}$/force $\mathbf{f}_\mathrm{e}$ to the correct positions in the global matrix/vector.

Note 2: The element matrices/vectors contribute to the global matrices/vectors at the degrees of freedom (DOFs) corresponding to that element. For example, element number 2 is connected to node number 2 and 3. Therefore, it contributes to rows 2-3 and columns 2-3 of the global stiffness matrix $\mathbf{K}$ (DOFs 2 and 3). You can find the DOFs related to the element e in the corresponding row of the mesh connectivity matrix, i.e., `mesh.nodesOfElem(e, :)`.

2.4 Solve

2.7. Allocate memory for the solution vector $\mathbf{u}$ and set the first and last entries to zero (this is our Dirichlet boundary condition for this example).

2.8. How many DOFs do we need to solve for if we have N nodes? Extract the unknown DOFs from the global matrix/vector, which results in $\mathbf{K}_\mathrm{free}$ and $\mathbf{f}_\mathrm{free}$.

2.9. Solve the linear system of equations $\mathbf{K}_\mathrm{free}\mathbf{u}_\mathrm{free} = \mathbf{f}_\mathrm{free}$ using MATLAB's backslash operator and assign the solution to the correct positions in the solution vector $\mathbf{u}$.

This concludes the implementation of this simple FE example. In the following, we shall validate the result.

3 Validate

3.1. Plot your computed solution $\mathbf{u}$ over the x-coordinates (submit this file when handing in your assignment).

3.2. Are the boundary conditions fulfilled?

3.3. Does the solution have the shape that would be expected by the given problem? How well does it approximate the analytical solution?

3.4. Does the procedure still work with a different number of elements on the same domain?

3.5. Does your implementation work for a different domain, especially where $a \neq 0$?

General hints and remarks

Please make sure to:

- Save all relevant plots you generate (e.g., as `png`-file) while working on the assignment.
- Always label the axes of the plots and add a legend if necessary.
- Format your plots so that they can be read easily (e.g., appropriate font size).
- Verify that your code executes without need of changes and without the need of additional files/libraries. Executing `fem_equidistant_hdbc.m` should yield the computed solution and plot the results on any computer running the 2020 version of Octave/Matlab/Python.

Hand in your

- code and
- a `Results.pdf` file with the plots and short answers to all questions (label each answer with its task number),

before the due date by uploading to the corresponding StudOn section.

When handing in for the first time you need to create a group. Please add all members when doing so! Once the group has been create, you can and should re-use it for the subsequent submissions.

Important: Make sure that your code **executes** without need of changes. Executing `fem_equidistant_hdbc.m` should yield the computed solution and plot the results on any computer running the current version of Octave/MATLAB.