CAE of Sensors and Actuators
Date: December 2020
Assignment due: February 3rd, 2021 at 23:55

Lehrstuhl für Sensorik
Universität Erlangen-Nürnberg
Daniel Kiefer

Assignment 0: FEM Implementation – Part 2

In *Assignment 0 – Part 1* you created a Finite Element code which solves the steady-state heat equation in 1D. The general boundary value problem (BVP) is given by

$$-ku'' = f \quad \text{on } x \in \Omega = [a, b] \tag{1a}$$
$$u(x) = g(x) \quad \text{on } x \in \Gamma_D \subset \partial\Omega \tag{1b}$$
$$(u'n)|_x = h(x) \quad \text{on } x \in \Gamma_N \subset \partial\Omega\,, \tag{1c}$$

where Γ_D and Γ_N denote the Dirichlet and Neumann boundaries, respectively. $u'n$ is the normal derivative in 1D, where n represents the "unit normal to the boundary", i.e., $n = 1$ at $x = b$ and $n = -1$ at $x = a$.

In this assignment you will extend your Finite Element code

- to support non-equidistant meshes,
- to handle general forcing functions $f(x)$, and
- implement general Dirichlet and Neumann boundary conditions (BCs).

Note: For many of the tasks you can re-use/adapt solutions of Assignment 0 – Part 1. You should do this whenever you consider it convenient.

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: Galerkin discretization

1.1. Write down the weak form of the BVP (1).

1.2. Perform a Galerkin discretization of the resulting weak form. Thereby, do not consider the BCs directly. Instead, use the ansatz

$$u^h(x) = \sum_{j=1}^{N} N_j(x) u_j$$
$$v^h(x) = \sum_{i=1}^{N} N_i(x) v_i$$

for the test function $v^h(x)$ and the trial function $u^h(x)$. You should end up with an expression similar to *exercise sheet 2*, problem 2.

2 Preparation: Isoparametric Finite Elements

In Part 1 we computed the stiffness matrix and forcing vector as a sum of integrals over the element domains. Thereby, you relied on the fact that we were using a regular mesh, where all the elements had the same size and shape. With an irregular mesh, however, each of the integrals has a different integration domain, which would require a different treatment of each of the elements. To avoid this difficulty, in practice the elements are mapped onto a so-called *reference* or *parent* element and the integration is then performed in a local coordinate system. Such a procedure is well suited for computer implementation because the mapping can be performed automatically. The system matrices are then obtained with the following steps:

- Define a map $F_e : \hat{\Omega} \to \Omega_e$, which maps the local coordinate $\xi \in \hat{\Omega}$ of the reference element to the global coordinate $x \in \Omega_e$.
- Express the global derivatives $\frac{\partial}{\partial x}$ in terms of the local coordinate derivatives $\frac{\partial}{\partial \xi}$.
- Express all functions appearing in the integrals in terms of the local coordinate.
- Integrate on the reference element $\hat{\Omega}$.

For our 1D case:

2.1. Write down the *isoparametric* ansatz for the mapping F_e.

2.2. Compute the corresponding linear basis functions $\hat{N}_a(\xi)$ for the reference element.

2.3. Determine the Jacobi matrix J_e, as well as J_e^{-T} and $\det J_e$.

2.4. Transform $N_a'(x) = \frac{\partial N_a(x)}{\partial x}$ to local coordinates, i.e., $\frac{\partial \hat{N}_a(\xi)}{\partial \xi}$. Do the same for $N_b'(x)$.

2.5. The entries k_{ab}^e of the element stiffness matrix have been derived in Part 1 of the assignment and are given in terms of the global coordinate x as

$$k_{ab}^e(x) = k \int_{\Omega_e} N_a'(x) N_b'(x) \, \mathrm{d}x \,. \tag{2}$$

Transform $k_{ab}^e(x)$ to the reference element $\hat{\Omega}$. Make sure, that the expression you obtain depends only on $\hat{N}_a(\xi)$, $\frac{\partial}{\partial \xi}$, J_e and k.

2.6. Transform the forcing function $f(x)$ to the local coordinate ξ. Denote the transformed function as $\hat{f}(\xi)$.

2.7. The entries f_a^e of the element force vector are determined in global coordinates by

$$f_a^e(x) = \int_{\Omega_e} f(x) N_a(x)\,\mathrm{d}x\,. \tag{3}$$

Transform $f_a^e(x)$ to the reference element $\hat{\Omega}$. Make sure, that the expression you obtain depends only on $\hat{N}_a(\xi)$, J_e and $\hat{f}(\xi)$.

3 Implementing the isoparametric FE with numerical integration

You will implement this version of your FE code based on your implementation of Assignment 0, Part 1. Move your previous code into a subfolder called `part1`. Then duplicate this folder and name it `part2`. You should now have two folders with the exact same code. For this part of the assignment, you will be doing all the work in the `part2` folder.

3.1. Implement the function `Je = jacobi(mesh, elem)` in the file `jacobi.m` which computes the Jacobi matrix (of size 1×1 for the 1D case) of the element `elem`. Use the `mesh` structure that you generated in Part 1 of the assignment.

3.2. Rename your `fem_equidistant_hdbc.m` to `fem_main.m`.

3.3. Define the basis functions $\hat{N}_a(\xi)$ as anonymous functions in your `fem_main.m`. It might be convenient to collect them in a cell array, e.g.,
`N = {@(xi) 1 - xi; @(xi) xi};`
Make sure to familiarize yourself with anonymous functions and cell arrays before continuing.

3.4. Test your definition of the basis functions:

3.4.1. Create a vector `xi` by sampling the interval $[0, 1]$ ten times.

3.4.2. Plot $N_1(\xi)$ and $N_2(\xi)$ into one figure using the anonymous functions defined in task 3.3. and the vector `xi`.

3.5. Similarly, define the derivatives $\frac{\partial \hat{N}_a(\xi)}{\partial \xi}$ as (a cell array of) anonymous functions in your Matlab code. As these functions need to support vector arguments, we need to make sure to return a vector of the same size as the argument. This can be achieved in the following way:
`Nxi = {@(xi) -1*ones(size(xi)); @(xi) 1*ones(size(xi))};`

3.6. Test your basis function derivatives defined above as done for the basis functions in task 3.4.

Note: You can later call, e.g., the basis function N_1 by typing `N{1}()` and passing the argument in the parenthesis, if necessary.

3.7. Edit the `elementStiffness()` function from Part 1. It should now have the signature `ke = elementStiffness(Nxi, Je)`, where the arguments are the vector of local derivatives `Nxi` of the reference shape functions and the Jacobi matrix `Je`. This function should

3.7.1. Compute $\det J_e$. Hint: Matlab determinant is `det()`.

3.7.2. Compute J_e^{-T}. Hint: Matlab transpose is `.'`, Matlab inverse is `inv()`.

3.7.3. Allocate memory for the element stiffness matrix `ke`.

3.7.4. Compute each of the entries k_{ab}^e of the matrix `ke` by performing a numerical integration on the reference element. You can use the Matlab function `integral()` for the integration procedure. You might need to define first the integrand as an anonymous function and then pass that as an argument to the `integral()` function. Be aware that all your anonymous functions should support vector arguments, i.e., use point-wise multiplication in their definition. Refer to task 2.5. for the expression of k_{ab}^e.

3.8. Test your `elementStiffness()` function. Is it returning the same matrix as you computed by hand in Part 1 of the assignment?

3.9. Define $f(x) = 1$ as an anonymous function of the argument `x` in your Matlab code. Make sure that it supports vector arguments and returns vectors of the same size (as in task 3.5.).

3.10. Create a new function `fhat = toLocalCoord(f, N, mesh, elem)` in the file `toLocalCoord.m`. It should transform given function `f` from global coordinates x to local coordinates ξ. The returned variable `fhat` is an anonymous function of the argument `xi`. Refer to task 2.6. on how to do this.

3.11. Edit the `elementForce()` function. The new signature should be `fe = elementForce(N, Je, fhat)`, where the arguments are the vector of reference shape functions `N`, the Jacobi matrix `Je`, as well as the forcing function `fhat` in local coordinates. This function should

3.11.1. Compute $\det J_e$.

3.11.2. Allocate memory for the element force vector `fe`.

3.11.3. Compute each of the entries f_a^e of the vector `fe`. Do this by performing a numerical integration on the reference element. Refer to task 2.7. for the expression of f_a^e.

3.12. Test your `elementForce()` function. Is it returning the same vector as you computed by hand in Part 1 of the assignment?

4 Adapting the assembling procedure

Change the assembling procedure in your `fem_main.m` file in order to use the new functions. When **iterating** over the elements:

4.1. Compute the Jacobi matrix for the current element `Je`.

4.2. Transform functions to local coordinates (only necessary when assembling the force vector).

4.3. Calculate the element matrix/vector by calling your newly created function.

4.4. Add the element matrix/vector to the correct position in the global matrix/vector. This is exactly the same as in Part 1 of the assignment.

5 Testing

Test your implementation with an irregular mesh first. Your results should coincide with your results from part 1 of this assignment.

5.1. Test your implementation with an irregular mesh of your choice. You need to adapt your `generateMesh()` function for this purpose. If your domain is $\Omega = [a, b] \subset \mathbb{R}$, you could for example use the *Chebyshev* points

$$x_k = a + \frac{b-a}{2}\left(1 - \cos\left(\pi\frac{k}{n}\right)\right), \quad \text{with } k \in [0...n], \text{ and } n \text{ the number of elements}. \tag{4}$$

The above points are denser close to the domain boundaries than in the interior and are often used for numeric purposes. Plot your result and save it as a `.png` file.

5.2. Because you are performing a numerical integration to calculate the forcing vector, you should now be able to solve the BVP for any excitation $f(x)$. Test your implementation with

$$f(x) = \sin^2\left(\frac{\pi x}{(b-a)}\right), \tag{5}$$

where $[a, b] \subset \mathbb{R}$ is your 1D domain. Plot your result and save it as a `.png` file. You solution should be slightly different to your result with $f(x) = 1$.

6 Implementing inhomogeneous Dirichlet BCs

In Part 1 of the assignment, you implemented homogeneous Dirichlet BCs. We will now extend the code for inhomogeneous Dirichlet BCs by applying elimination.

Note: You can refer to `slides_dirichlet_node_handling.pdf` on how this works. You can also refer to *exercise sheet 2*, problem 1 for help.

6.1. In `fem_main.m`, define a vector `g` which contains the two Dirichlet values $u(a)$ and $u(b)$. Set them to 1 and -0.5, respectively.

6.2. After assembling the global forcing vector `F`, define a right hand side vector `RHS` which shall contain not only the forcing, but also the contribution of the BCs. For now, initialize `RHS` simply with `F`.

6.3. Now, incorporate the Dirichlet contributions $K_{i1}g_1$ and $K_{i1}g_1$ by subtracting them from the `RHS` vector. This is shown in `slides_dirichlet_node_handling.pdf`.

6.4. After allocating memory for the solution vector `u`, set the boundary values to the known Dirichlet values `g`.

6.5. Solve the system as before and plot the solution. Are the Dirichlet conditions fulfilled? Save your plot as a `.png` file.

7 Implementing Neumann BCs

You can refer to *exercise sheet 2*, problem 2 for help on how to consider Neumann BCs. The corresponding contribution to the right hand side vector should appear in your computed weak form.

7.1. In `fem_main.m`, define a vector `h` which contains the Neumann values $u'n|_{\partial\Omega}$. Set them to whatever values you want.

7.2. Adjust `mesh.nodesOfDBC` to correctly represent on which node you want to impose the Dirichlet BC. When applying only Neumann BCs, it should be an empty vector, i.e., `[]`.

7.3. Define `mesh.nodesOfNBC` to be the Neumann boundary nodes. `setdiff()` might be a useful function for this, but you are not required to use it.

7.4. Incorporate the Neumann BCs to the vector `RHS` by adding the Neumann contributions which appear in your weak form of task 1.1.

7.5. Solve the system for the non-Dirichlet nodes as before.

If you implemented Neumann BCs on both boundary nodes, Matlab will warn you that your matrix is singular during the solution procedure. Why is that? Implement a homogeneous Dirichlet BC on one node and a homogeneous Neumann BC on the other node:

7.6. Adjust the vector of Dirichlet values and Neumann values appropriately.

7.7. Adjust `mesh.nodesOfDBC` and `mesh.nodesOfNBC`.

7.8. Solve the resulting system of equations and plot the solution. Are the desired boundary conditions fulfilled? Save your plot as a `.png` file.

Submitting your assignment

Hand in your group's

- 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_main.m` should yield the computed solution and plot the results on any computer running the current version of Octave/MATLAB.