Engineering Analysis using the Finite Element Method

University of California, Berkeley, Spring 2019
Instructor: Sanjay Govindjee
GSI: Linus Mettler

Lab 3

In this lab, you will write your own FE code for 1-D thermal problems. We will start by implementing linear element shape functions and proceed next week with second order elements. For both, you will examine the convergence behavior.

Your FE code will be quite general in that it can solve any 1-D steady-state thermal problem with

- inhomogeneous conductivity and cross-section area
- inhomogeneous distributed heat source
- variable element sizing
- essential and natural boundary conditions
- first and second order elements *(second order next week)*

To this end, you will implement isoparametric shape functions, location matrices and Gaussian quadrature.

To help you get started, you are provided with a Matlab template. You are **not** required to use the template. You may want to replace parts of it with your own coding strategy.

Use the Lab 2 steady-state thermal load case to test your implementation.

In order to check off this lab you will provide a plot of your FE result vs. the exact solution (for both linear and quadratic elements), as well as a convergence plot for each element order. No report is required.

1 Problem statement

Consider a slender bar of length $L = b-a$, inhomogeneous thermal conductivity $K(x)$ and cross sectional area $A(x)$ subjected to a distributed heat source $Q(x)$ (with units of power per unit length). The governing differential equation is

$$\frac{d}{dx}\left(A(x)K(x)\frac{dT(x)}{dx}\right) + Q(x) = 0 \quad \text{on} \quad \Omega = (a,b) \tag{1}$$

The boundary conditions can be either essential or natural[1]

$$T(a) = \bar{T}_a \qquad \text{or} \qquad -\left(K(x)\frac{dT(x)}{dx}\right)_{x=a} = \bar{q}_a \tag{2}$$

$$T(b) = \bar{T}_b \qquad \text{or} \qquad \left(K(x)\frac{dT(x)}{dx}\right)_{x=b} = \bar{q}_b \tag{3}$$

where $\bar{T}_a$ and $\bar{T}_b$ are the temperatures specified at the boundaries $x = a$ and $x = b$, and $\bar{q}_a$ and $\bar{q}_b$ are the prescribed heat fluxes, respectively. Of course, at any given boundary one can only prescribe *either* the temperature *or* the heat flux.

Derive the weak form statement. Be sure to include the boundary terms arising from the natural boundary conditions.

Derive expressions for the element stiffness matrix $\mathbf{k^e}$ and element load vector $\mathbf{f^e}$ in terms of integrals over the spatial variable x using linear and quadratic element shape functions $N_i^e(x)$ to approximate the temperature distribution within the element.

Recalling the isoparametric map from x to ξ, rewrite the integrals in terms of the parametric spatial variable ξ.

2 FE code

Your finite element code will be roughly divided into the following modules:

- setting all parameters, boundary conditions, loads etc.
- mesh generator:
  - (i) nodal locations
  - (ii) location matrix `LM`

[1] Recall, however, that at least one essential boundary condition is necessary for the solution to the differential equation to be unique.

(iii) `idd` (and `idf`) logical mask arrays

- loop over all elements:
  - integrate element stiffness and element load vector
    (i) linear and quadratic shape functions
    (ii) isoparametric mapping
    (iii) Gaussian integration
  - add element contribution to global stiffness matrix and global load vector
- solution taking into account essential and natural boundary conditions
- post-processing (already coded for you):
  (i) plotting results
  (ii) error in L^2 norm

The template provided to you will help you organize the structure of your code. It is often helpful to create functions for specific tasks, thereby creating a modular structure to your code.

2.1 Mesh generator

Create a function to generate your mesh. The function takes as input

(i) geometry parameters (a, b)
(ii) number of elements
(iii) element order

and creates as output a structure `mesh` with fields

(i) `lm` location matrix LM
(ii) `n` array containing the nodal locations
(iii) `nen` number of nodes per element
(iv) `numnod` number of nodes in the mesh
(v) `numel` number of elements in the mesh
(vi) `idf` mask array for free degrees of freedom
(vii) `idd` mask array for driven degrees of freedom
(viii) `val` given values of heat flow (on natural boundary)[2]

[2]This is a column vector of the same size as the number of nodes. All entries except for those at the natural boundary nodes will be zero.

2.2 Integration of element stiffness matrix and load

For the integration of the stiffness matrix always start with the weak form of the problem statement. Transform the integral over the element from x to parametric space ξ in order to set up the Gaussian integration.

Note that we don't store the element stiffness matrices. Once computed, we insert the values into the global stiffness matrix and discard the element stiffness matrix as we move on to integrate the next element.

2.3 Assembly of global equations

Make use of the location matrix to insert the element stiffness values at the correct location of the global stiffness matrix. The same goes for the assembly of the load vector. Your code should work even if the element / node numberings are messy.

2.4 Solution

If natural boundary conditions are present, there are additional terms in the load vector that have to be taken care of. Also, remember that the stiffness matrix is singular to begin with. You must incorporate the natural boundary conditions in order to arrive at a solution to the system of equations. This is done most conveniently using mask arrays as shown in class.

2.5 Post-processing

No additional work should be required for post-processing (unless perhaps you used different variable names etc.). For your own interest, you are of course welcome to do your own post-processing on the FE result.

Appendices

A Review: From weak form to FE solution

Starting with the weak form, we used the following procedure to arrive at our finite element formulation of the boundary-value problem:

(a) Break up domain into N^e elements. Pick the number of nodes according to the desired order of the element. The nodes can, but don't need to be, distributed equidistantly.

(b) Define two functions (an approximation $T^N(x)$ to the true solution and a test function $v^N(x)$)

$$T^N(x) = \sum_{j=1}^{N} a_j N_j(x)$$

$$v^N(x) = \sum_{i=1}^{N} b_i N_i(x)$$

(c) Evaluate the weak form integral equation (sum over all i and j). Here, $\bar{Q}\big|_x$ is the rate of heat moving across the boundary where natural boundary conditions are applied.

$$\int_a^b \frac{d}{dx}\Big(\sum_{i=1}^{N} b_i N_i(x)\Big) A(x)K(x) \frac{d}{dx}\Big(\sum_{j=1}^{N} a_j N_j(x)\Big) dx =$$

$$= \int_a^b Q(x) \sum_{i=1}^{N} b_i N_i(x) dx + \left[\bar{Q}\Big|_x \sum_{i=1}^{N} b_i N_i(x)\right]_a^b$$

(d) Noting that the summation and integration can be interchanged and that b_i in the above equation can be factored out, we arrive at a linear algebraic equation

$$\{b\} \cdot \Big([K]\{a\} - \{F\}\Big) = 0$$

which must be valid for any choice of b_i (the coefficients of our arbitrary test function). This implies that $\big([K]\{a\} - \{F\}\big) = 0$. Here, the components of the so-called stiffness matrix are given by

$$K_{ij} = \int_a^b \frac{dN_i(x)}{dx} A(x)K(x) \frac{dN_j(x)}{dx} dx$$

and the external load is represented by

$$F_i = \int_a^b Q(x)N_i(x)dx + \text{boundary term if applicable}$$

(e) Break up the domain of the integrals into the individual elements.

$$K_{ij} = \int_\Omega \frac{dN_i(x)}{dx} A(x)K(x) \frac{dN_j(x)}{dx} dx = \sum_{elements} \int_{\Omega^e} \ldots dx = \sum_{elements} k_{ij}^e$$

$$F_i = \sum_{elements} \int_{\Omega^e} \ldots dx + \text{boundary term if applicable} = \sum_{elements} f_i^e$$

(f) For every element, integrate to obtain the element stiffness matrices (which we then assemble into the global stiffness matrix) and element load vector. Using the isoparametric map

$$\hat{x}(\xi) = \sum_{i=1}^{p+1} x_i \hat{N}_i(\xi)$$

we can convert the integrand to the parametric space

$$\begin{aligned} k_{ij}^e &= \int_a^b \frac{dN_i^e(x)}{dx} A(x) K(x) \frac{dN_j^e(x)}{dx} dx \\ &= \int_{-1}^1 \frac{dN_i^e(\hat{x}(\xi))}{d\xi} \frac{d\xi}{dx} A(\hat{x}(\xi)) K(\hat{x}(\xi)) \frac{dN_j^e(\hat{x}(\xi))}{d\xi} \frac{d\xi}{dx} \frac{dx}{d\xi} d\xi \\ &= \int_{-1}^1 \frac{d\hat{N}_i(\xi)}{d\xi} \frac{1}{\hat{J}(\xi)} \hat{A}(\xi) \hat{K}(\xi) \frac{d\hat{N}_j(\xi)}{d\xi} \frac{1}{\hat{J}(\xi)} \hat{J}(\xi) d\xi \end{aligned}$$

where $\hat{J}(\xi)$ is the Jacobian, which is in general a function of ξ.[3] The Jacobian can be computed from the nodal locations x_i and the derivatives of the element shape functions according to

$$\hat{J}(\xi) = \frac{d}{d\xi}\hat{x}(\xi) = \sum_{i=1}^{p+1} x_i \frac{d}{d\xi}\hat{N}_i(\xi)$$

where p denotes the order of the element (so $p+1$ is the number of nodes, i.e. the number of shape functions, per element). The element load vector is similarly

$$\begin{aligned} f_i^e &= \int_a^b Q(x) N_i^e(x) dx + \text{boundary term} \\ &= \int_{-1}^1 Q(\hat{x}(\xi)) N_i^e(\hat{x}(\xi)) \frac{dx}{d\xi} d\xi + \text{boundary term} \\ &= \int_{-1}^1 \hat{Q}(\xi) \hat{N}_i(\xi) \hat{J}(\xi) d\xi + \text{boundary term} \end{aligned}$$

(g) In the parametric space, evaluate the above integrals using Gauss integration. In the choice of number of integration points, take into account the polynomial orders of each integrand.

[3] Note: for linear elements, the Jacobian is a constant.

(h) Reassemble the global stiffness matrix and the global load vector. The element stiffness matrix entry of a node, which is shared between two adjacent elements, must be summed with the corresponding entry of the other element stiffness matrix. The same principle holds for the load vector.

(i) Take into account the boundary conditions. More specifically, add the $\bar{Q}$ term to the load vector at the boundary node where a natural boundary condition applies.

(j) Solve the linear system of equations for the unknown a_i. Take into account that some a_i are known from essential boundary conditions (temperature prescribed). We recommend you use mask arrays to deal with the essential boundary conditions.

(k) Post-processing: Show the result by mapping the solution at a fixed number of points from the parametric domain onto the real domain. Some additional post-processing is needed to display, for example, the strain distribution (taking the derivative in the parametric domain and map it back to the real domain).

(l) Compute the L^2 norm as defined in the previous assignment. Note that for the integration we employ the very same strategy of breaking the integral into the individual elements, mapping to the local parametric coordinate system and performing the integrations there.