Transient Two-Dimensional Conduction Heat Transfer

Course Project

ME 498 001 — Dr. V. Prantil

Paul Gessler

Mechanical Engineering Department
Milwaukee School of Engineering

August 11, 2012

Abstract

In this course project, a transient thermal analysis code was developed and implemented in MATLAB using the Finite Element Method (FEM). The code employs a Newmark-Beta integration algorithm to compute the transient temperature response of a two-dimensional domain discretized with three-noded triangular elements. Provisions are made for fixed temperature Dirichlet boundary conditions, which may vary spatially as well as temporally. Two validation cases were studied to evaluate the accuracy of the code developed here. The first case was a steady-state validation, and the MATLAB code agreed to an analytical solution to within 0.1% for reasonable element counts. The second case was a transient validation, with a time-variant temperature boundary condition applied to one edge of a square domain. The other three edges were held at specified constant temperatures. In the transient validation case, the MATLAB code developed here agreed to within 1.4% with an equivalent analysis computed using commercial ANSYS Workbench software.

1 Introduction

A finite element analysis (FEA) code was written to solve two-dimensional heat transfer problems involving conduction in homogeneous materials. The code was implemented to handle potentially time-variant boundary conditions, and hence employs time integration to compute the transient solution. No temperature variation or heat transport in the third dimension were considered. Additionally, heat generation within the domain was not considered. Constant strain triangle (CST) elements were used exclusively.

The general configuration of the domain studied using the code is shown in Fig. 1.

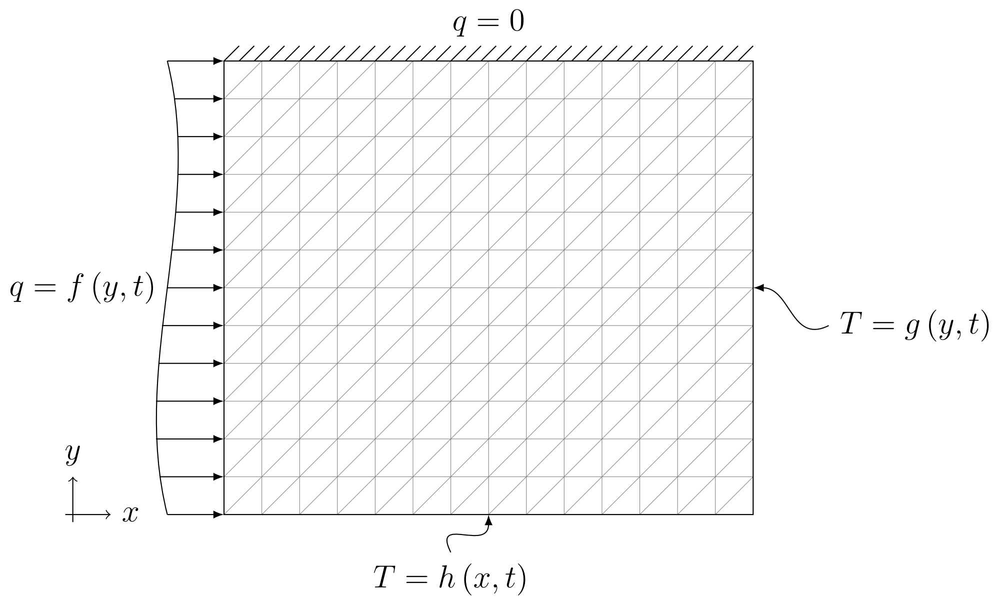

Figure 1: Two-Dimensional Heat Conduction Problem Configuration

The finite element solution was compared to that given by ANSYS, a commercial FEA software package, for verification purposes. Additionally, a steady-state analysis was conducted and compared to an analytical closed-form solution for validation.

2 Solution Methodology

The governing differential equation to be solved can be derived for an arbitrary control volume from the general form of the heat diffusion equation [1],

$$\frac{\partial}{\partial x}\left(k\,\frac{\partial T}{\partial x}\right) + \frac{\partial}{\partial y}\left(k\,\frac{\partial T}{\partial y}\right) + \frac{\partial}{\partial z}\left(k\,\frac{\partial T}{\partial z}\right) + \dot{q} = \rho c_p\,\frac{\partial T}{\partial t}\,, \tag{1}$$

where x, y, and z are the Cartesian coordinates, k is the thermal conductivity of the material, $T = T(x, y, z, t)$ is the temperature distribution, $\dot{q}$ represents any thermal energy generation within the control volume, ρ is the density of the material, c_p is the material's specific heat capacity, and t represents time.

Applying the homogeneous material and no heat generation assumptions, we take $\partial k/\partial x = \partial k/\partial y = \partial k/\partial z = 0$ and $\dot{q} = 0$, so Eq. (1) reduces to

$$\frac{\partial^2 T}{\partial x^2} + \frac{\partial^2 T}{\partial y^2} + \frac{\partial^2 T}{\partial z^2} = \frac{\rho c_p}{k}\frac{\partial T}{\partial t}\,. \tag{2}$$

Finally, taking $\partial^2 T/\partial z^2 = 0$ since temperature variation and heat transport in the third dimension are neglected, we have

$$\frac{\partial^2 T}{\partial x^2} + \frac{\partial^2 T}{\partial y^2} = \frac{\rho c_p}{k}\frac{\partial T}{\partial t}\,. \tag{3}$$

The boundary conditions outlined in Fig. 1 can be described formally by

$$T = T_b = f(x,y) \text{ on } \Gamma_{\mathrm{b}} \tag{4}$$

for the boundary with a specified temperature (Γ_{b}) and

$$k\left(\frac{\partial T}{\partial x}\,a + \frac{\partial T}{\partial y}\,b\right) + q = 0 \text{ on } \Gamma_{\mathrm{q}} \tag{5}$$

for the boundary with a specified heat transport $\left(\Gamma_{\mathrm{q}}\right)$, where a and b are the direction cosines. Note that convection or radiation from the boundaries are not considered, and thus have not been included in the boundary condition descriptions. Since these are the only types of boundary conditions considered, $\Gamma_{\mathrm{b}} \cup \Gamma_{\mathrm{q}} = \Gamma$ and $\Gamma_{\mathrm{b}} \cap \Gamma_{\mathrm{q}} = 0$, where Γ represents the entire boundary of the domain. The initial conditions for the problem are a uniform temperature throughout the domain, such that

$$T(x,y) = T_0 \text{ at } t = 0. \tag{6}$$

The Galerkin weighted residuals approach is applied to Eq. (3), following the development shown in [2, 3] but incorporating the assumptions detailed above. Following the Galerkin approach, we wish to represent the temperature throughout the domain as

$$T(x,y,t) = \sum_{i=1}^{n} N_i(x,y,t)\,T_i(t)\,, \tag{7}$$

where N_i are the shape functions listed in Section 2.1, n is the number of nodes in an element (for CST elements, $n = 3$), and $T_i(t)$ are the nodal temperatures (time-dependent).

The weak form of this problem is then given by [4]

$$\Big[C\Big]\Big\{\dot{T}\Big\} + \Big[K\Big]\Big\{T\Big\} = \Big\{F\Big\}\,. \tag{8}$$

The formulation of $[C]$ and $[K]$ for the CST element are listed in Section 2.1.

The transient solution in time is captured through the use of a time integration algorithm. In this work, the Newmark-Beta approach is used, since it provides solutions for a wide variety of integration schemes (explicit Euler, Crank-Nicholson, fully implicit, etc.) by simply changing the value of a constant β.

Newmark's equations [5] reduce to

$$\left(\frac{1}{\Delta t}\left[C\right]+\beta\left[K\right]\right)\left\{T\right\}_{n+1}=\left(\frac{1}{\Delta t}\left[C\right]-\left(1-\beta\right)\left[K\right]\right)\left\{T\right\}_{n}+\left(1-\beta\right)\left\{F\right\}_{n}+\beta\left\{F\right\}_{n+1} \quad (9)$$

for the weak form given in Eq. (8), where Δt is the time step, subscripts n and $n+1$ represent the current and next (to be computed) time indices, respectively, and the parameter β determines the integration scheme to be used. Here, we take $\beta = 1/2$, corresponding to Crank-Nicholson (trapezoidal) integration. This selection (and, in fact, any $\beta >= 1/2$) renders the solution *unconditionally stable*, meaning that the solution remains bounded regardless of the time step Δt chosen.

2.1 Element Formulation

For this analysis, the three-noded triangle element was selected, with linear shape functions. The generalized configuration of the element, showing nomenclature and degrees of freedom, is shown in Fig. 2. The shape functions N_i of Eq. (7) for the three-noded

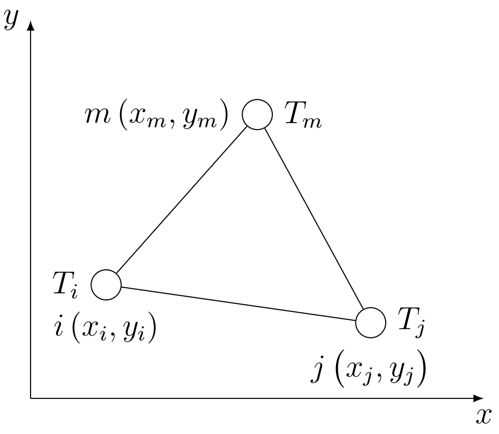

Figure 2: Basic triangular element nomenclature and DOF. Adapted from [5].

triangle are listed by Logan [5] and can be represented by

$$N_i = \frac{1}{2A}\left(\alpha_i + \beta_i x + \gamma_i y\right), \quad (10)$$

with similar relationships for nodes j and m, where A is the area of the triangle and the coefficients α, β, and γ are

$$\begin{array}{lllllllll} \alpha_i & = & x_j y_m - y_j x_m & \alpha_j & = & y_i x_m - x_i y_m & \alpha_m & = & x_i y_j - y_i x_j \\ \beta_i & = & y_j - y_m & \beta_j & = & y_m - y_i & \beta_m & = & y_i - y_j \\ \gamma_i & = & x_m - x_j & \gamma_j & = & x_i - x_m & \gamma_m & = & x_j - x_i \end{array} . \quad (11)$$

Then the elemental stiffness matrix $\left[K^{(e)}\right]$ [6] is given by

$$\left[K^{(e)}\right] = tA\left[B\right]^T\left[D\right]\left[B\right], \quad (12)$$

where t is the element thickness (constant),

$$[B] = \frac{1}{2A} \begin{bmatrix} \beta_i & \beta_j & \beta_m \\ \gamma_i & \gamma_j & \gamma_m \end{bmatrix}, \tag{13}$$

and

$$[D] = k \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}, \tag{14}$$

with k representing the (constant) thermal conductivity of the isotropic material. The consistent capacitance matrix for the three-noded triangular element [2] is

$$\left[C^{(e)}\right] = \frac{\rho c_p A t}{12} \begin{bmatrix} 2 & 1 & 1 \\ 1 & 2 & 1 \\ 1 & 1 & 2 \end{bmatrix}, \tag{15}$$

where ρ is the material density and c_p is the thermal capacitance of the material.

The form of $\{F\}$ for this problem configuration is simply the prescribed amount of heat transfer at each node. Since no internal heat generation is considered here, boundary sides with fixed (zero or otherwise) heat transfer will have that amount of heat transport applied to the side. In practice, the heat transport is applied in two parts: one half to the node at one end of the element side, and the other half to the other adjacent node, resulting in the following form of $\left\{F^{(e)}\right\}$ for the element:

$$\frac{q^* L_{i-j} t}{2} \begin{Bmatrix} 1 \\ 1 \\ 0 \end{Bmatrix}, \tag{16}$$

where q^* is the applied heat flux and L_{i-j} is the length of the side (in this case side i-j). Similar results follow for applied heat transports on the other two faces of the element.

In the validation cases studied here, prescribed temperatures on the left-hand side of Eq. (8) must be multiplied with the stiffness matrix and moved to the right-hand side. These terms become part of the applied 'forces' on the free degrees of freedom solved once the matrices are reduced.

2.2 Discretization and Assembly

The element matrices and vectors described in Section 2.1 must be assembled into global formulations to solve for the response of the structure as a whole. In order to do this, we must store information about each element, its vertices, and neighboring elements sharing vertices and edges. Additionally, it will be useful to know if a particular element is part of the boundary of the domain, so that boundary conditions may be applied in an appropriate manner.

To illustrate the assembly process, a coarse mesh will be assembled here. The code uses a generalized procedure to allow for different meshes to be compared. The mesh used to illustrate the assembly process is shown in Fig. 3, and consists of 8 elements. The node numbers are shown as encircled numerals, while the element numbers are boxed. Note that based on the boundary conditions of Fig. 1, the nodal temperature along the

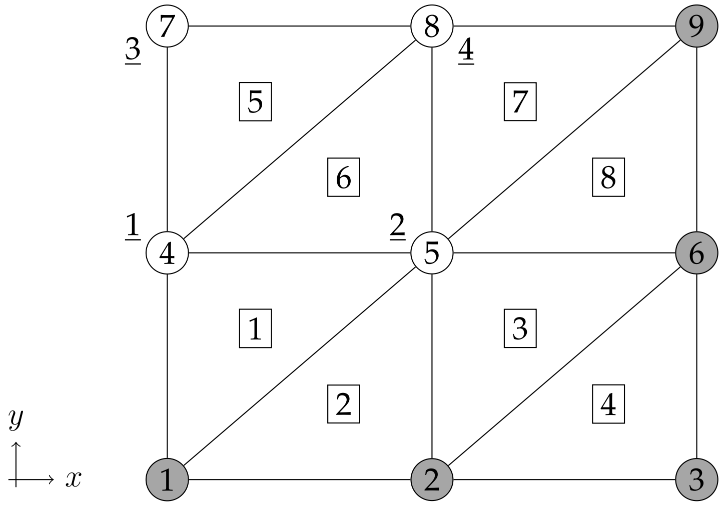

Figure 3: Coarse Mesh Configuration and Numbering Scheme

bottom and right edges of the domain is fixed. This is denoted by a gray shading on the nodes in Fig. 3. The unshaded nodes are the active degrees of freedom in the problem. Using this information, we can assemble an ID array numbering the active degrees of freedom and placed in a position corresponding with the node number at that location. The array has dimensions `nDOF`×`nNodes`, where `nDOF` represents the number of degrees of freedom at a node and `nNodes` represents the number of nodes in the mesh. For a thermal problem, there is only one degree of freedom (temperature) at each node, so for this example, the ID array becomes a row vector of the form

$$[\mathrm{ID}] = \begin{array}{c} \begin{array}{ccccccccc} \mathrm{n1} & \mathrm{n2} & \mathrm{n3} & \mathrm{n4} & \mathrm{n5} & \mathrm{n6} & \mathrm{n7} & \mathrm{n8} & \mathrm{n9} \end{array} \\ \left[\begin{array}{ccccccccc} 0 & 0 & 0 & 1 & 2 & 0 & 3 & 4 & 0 \end{array}\right] \end{array}, \tag{17}$$

meaning that there are 4 active degrees of freedom, located at nodes 4, 5, 7, and 8. These active degrees of freedom are denoted by underlined numerals in Fig. 3.

Then, using the elemental node numbering convention of Fig. 2, that is, numbering in a counter-clockwise fashion beginning with the lower left-most node of the element, we can generate the LM array, which contains information about the active degrees of freedom within each element. The LM array has dimensions `nen`×`nel`, where `nen` is the total number of nodes per element and `nel` is the total number of elements in the mesh. For this example, the LM array is

$$[\mathrm{LM}] = \begin{array}{c|cccccccc|} & \mathrm{e1} & \mathrm{e2} & \mathrm{e3} & \mathrm{e4} & \mathrm{e5} & \mathrm{e6} & \mathrm{e7} & \mathrm{e8} \\ \mathrm{n1} & 0 & 0 & 0 & 0 & 1 & 1 & 2 & 2 \\ \mathrm{n2} & 2 & 0 & 0 & 0 & 4 & 2 & 0 & 0 \\ \mathrm{n3} & 1 & 2 & 2 & 0 & 3 & 4 & 4 & 0 \end{array}. \tag{18}$$

From the LM array we can ascertain that active degree of freedom number 2 is associated with node 1 of elements 7 and 8, node 2 of elements 1 and 7, and node 3 of elements 2 and 3. This agrees with Fig. 3. The LM array allows the appropriate contributions from each element to be assembled into the global matrices. For example to find the contribution to the global stiffness $[K]$ from $K_{ij} = K_{11}$ of element 5, K_{11}^5, we examine column 5 (corresponding to element 5) of the LM array and index to the first row. Here we find a 1, so we know that K_{11}^5 contributes directly to K_{11} of the global matrix. Similarly, if we wanted to find the contribution of K_{12}^5 to the global stiffness matrix, we find that it contributes to K_{14} of the global stiffness matrix.

For the global forcing (imposed heat flux and temperature) vectors, a similar approach is used to determine the row(s) in which a particular element's heat fluxes and fixed nodal temperatures appear in the global formulation.

3 Results

To verify and validate the operation of the transient thermal analysis code (listed in the Appendix), two cases were considered, both with fixed boundary temperatures along the entire boundary. The first case is a steady-state solution, the results of which are detailed in Section 3.1. Here, the MATLAB code was given a time-invariant set of boundary temperatures and the transient solution was computed until steady-state conditions had been achieved. This result was compared to an analytical closed-form solution as well as a steady-state analysis in ANSYS Workbench.

The second case is a transient solution, detailed in Section 3.3, where the temperatures on three sides of the domain were fixed and the temperature on the final side varied with time. The results of this analysis were compared to the results of a transient analysis in ANSYS Workbench, using the same boundary conditions.

For both cases, the material was taken to be Copper, with material properties evaluated at 300 K. The property values are a density $\rho = 8933\,\mathrm{kg/m^3}$, thermal conductivity $k = 401\,\mathrm{W/m/K}$, and specific heat capacity $c_p = 385\,\mathrm{J/kg/K}$. These values were used in the MATLAB analysis code listed in the Appendix as well as the ANSYS analyses.

3.1 Steady-State Validation

The results of the code listed in the Appendix were compared to both ANSYS Workbench and an analytical solution for a steady-state case. The analytical solution, developed by Carslaw and Jaeger [7] holds for a rectangular plate with the left, top, and right edges fixed at $T = 0\,^\circ\mathrm{C}$ and the bottom edge fixed at an elevated temperature T_b. The temperature distribution in the plate takes the infinite series form

$$T(x,y) = \frac{4T_b}{\pi}\sum_{n=0}^{\infty}\frac{1}{2n+1}\sin\frac{(2n+1)\,\pi x}{a}\sinh\frac{(b-y)\,(2n+1)\,\pi}{a}\operatorname{cosech}\frac{(2n+1)\,\pi b}{a}, \quad (19)$$

where a and b are the x- and y-dimensions of the plate, respectively. For the steady-state validation case, we take $a = b = 1\,\mathrm{m}$ and $T_b = 100\,^\circ\mathrm{C}$.

A MATLAB m-code script was written to compute the analytical series solution, using convergence criteria to stop computing terms in the series when the temperature had converged to an acceptable level of accuracy. An absolute temperature tolerance $\epsilon_{\text{abs}} = \pm 0.001\,^\circ\text{C}$ was used to generate the temperature contours shown in Fig. 4. Also in Fig. 4, the results of the MATLAB transient thermal analysis code are shown for comparison with the analytical solution. For these FEM results, `nel`$= 800$ was used. A quantitative comparison with the analytical solution is presented as part of the mesh independence study of Section 3.2.

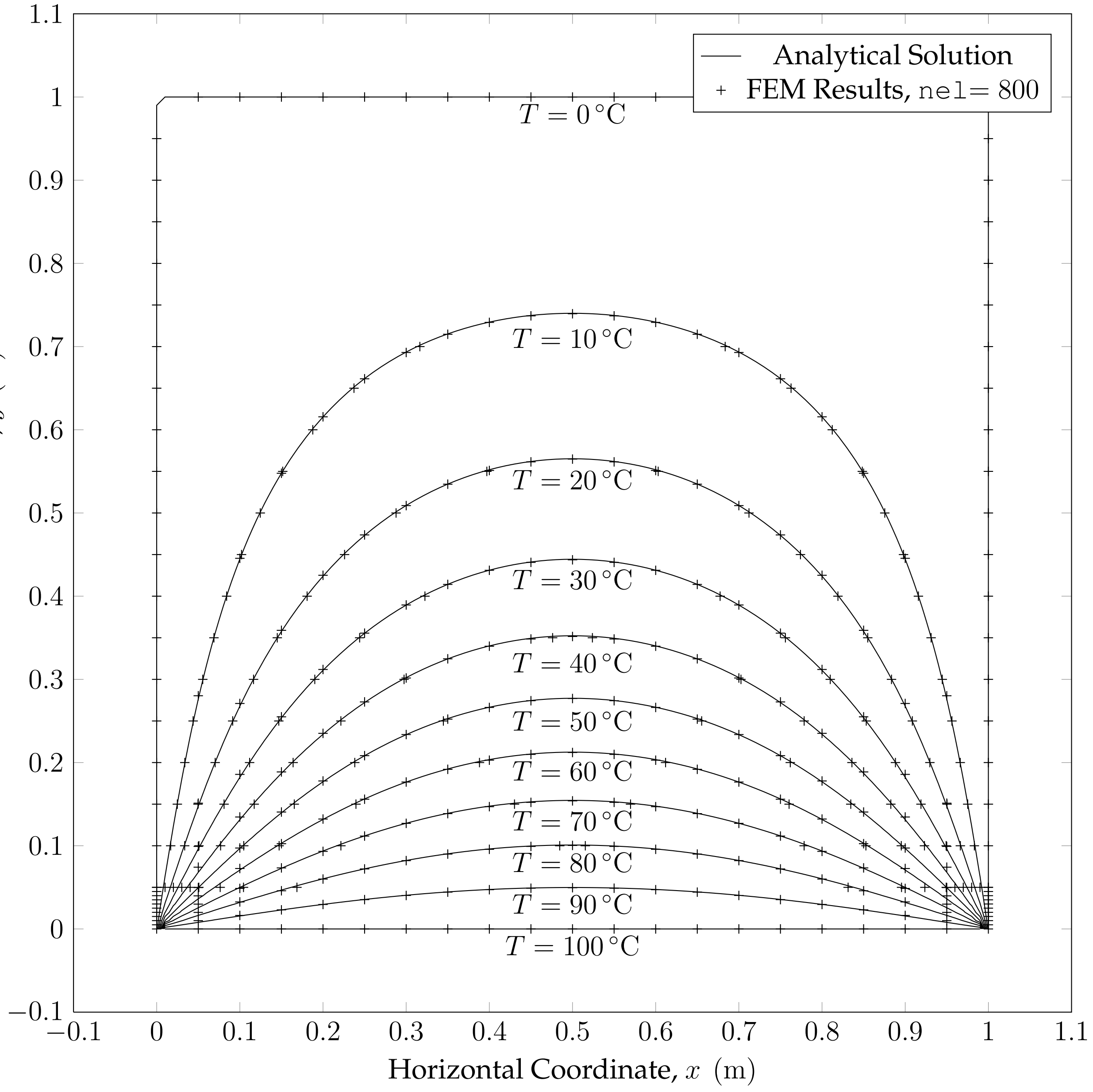

Figure 4: Temperature distribution, steady-state. Comparison of analytical solution and FEM code implemented in MATLAB, with 800 three-noded triangular elements.

The steady-state solution for the conditions described above was also computed in ANSYS Workbench. The temperature contours generated by ANSYS for a mesh composed of triangular elements with `nel`$= 1360$ are shown in Fig. 5.

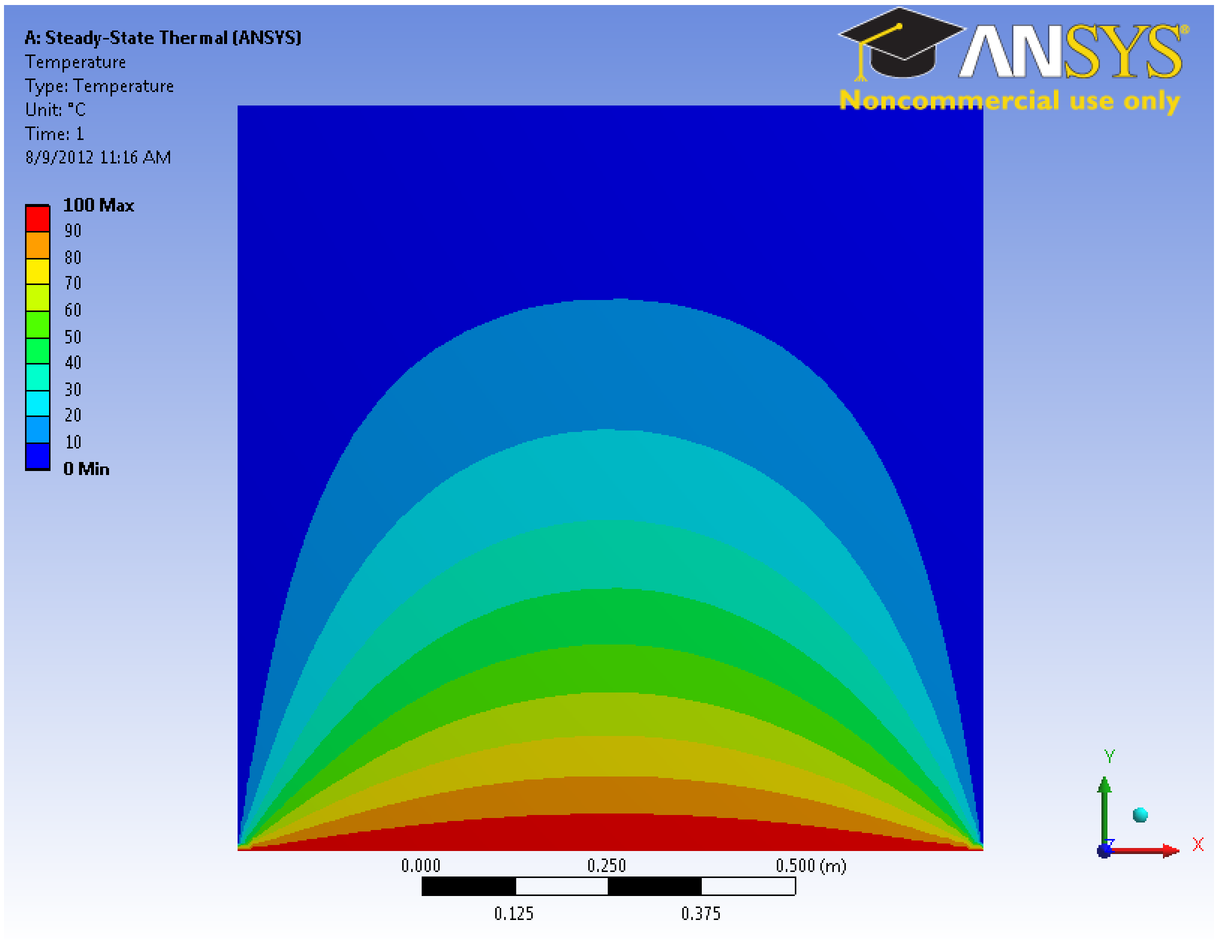

Figure 5: Temperature distribution, steady-state, ANSYS Workbench solution.

3.2 Mesh Independence Study

To evaluate the accuracy of the finite element analysis code, the result for the steady-state temperature at the geometric center of the domain was compared to the analytical and ANSYS results listed in Section 3.1. For the MATLAB results, a time step $\Delta t = 5\,\mathrm{s}$ was selected and the solution was computed until the temperatures differed by less than $\epsilon_{\mathrm{abs}} = 0.001\,^{\circ}\mathrm{C}$ between time steps. A uniform mesh was used for all cases. The results for various meshes are listed in Table 1.

Table 1: Steady-State Analytical Solution Validation Results

Method	T_{ss} ($^\circ$C)	Percent Error (%)
Analytical	25.00	N/A
ANSYS, `nel`= 1360	24.98	0.08
MATLAB, `nel`= 8	24.95	0.20
MATLAB, `nel`= 32	24.97	0.12
MATLAB, `nel`= 72	24.98	0.08
MATLAB, `nel`= 128	24.98	0.08
MATLAB, `nel`= 200	24.98	0.08
MATLAB, `nel`= 288	24.98	0.08
MATLAB, `nel`= 392	24.99	0.04
MATLAB, `nel`= 512	24.98	0.08
MATLAB, `nel`= 648	24.98	0.08
MATLAB, `nel`= 800	24.99	0.04
MATLAB, `nel`= 1152	24.99	0.04
MATLAB, `nel`= 1800	24.99	0.04
MATLAB, `nel`= 5000	24.99	0.04

3.3 Transient Validation

For the transient validation, a time-variant temperature boundary condition was applied to the bottom edge of the domain, given by

$$T_b = 50\,(1 + \cos t)\,, \tag{20}$$

for $0 <= t <= 3000\,\text{s}$. In both analyses, the time step Δt was taken to be 5 s for consistency. ANSYS, by default, uses the Newmark-Beta time integration algorithm with $\beta = 1$, corresponding to fully implicit backward difference integration [8]. This parameter selection was also used in the MATLAB code, but the results did not differ appreciably from results with $\beta = 1/2$ at this small time step. To provide an accurate comparison, the number of degrees of freedom in each mesh should be relatively similar. For this reason, the ANSYS results here have `nel`= 5078 and the MATLAB results have `nel`= 5000. The `PLANE77` element was used within ANSYS, while the element formulation in MATLAB follows that outlined in Section 2.1.

The transient validation results are plotted in Fig. 6. The temperature at the geometric midpoint of the domain is plotted with respect to time for both the ANSYS Workbench validation and MATLAB code listed in the Appendix.

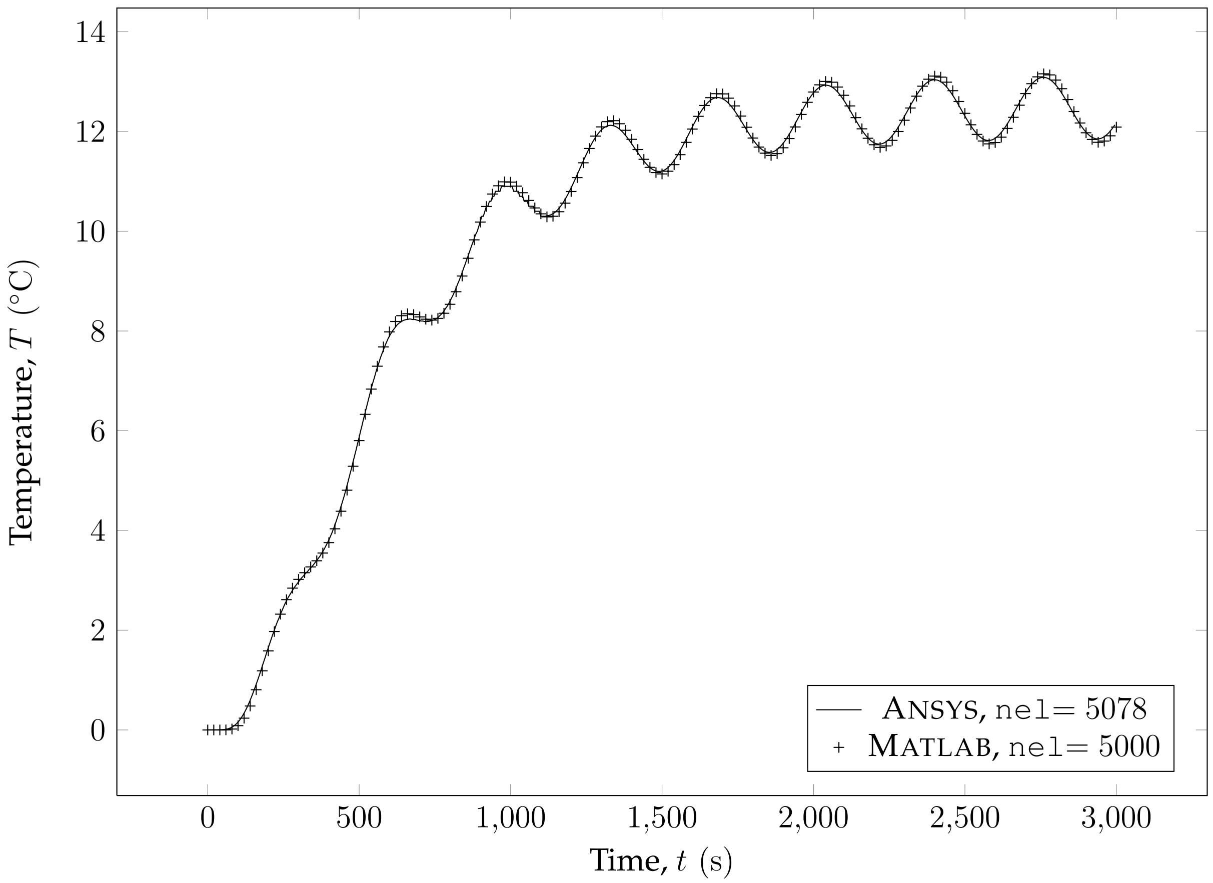

Figure 6: Transient validation results. Temperature history at the geometric center of the domain, ANSYS validation and MATLAB FEM analysis code results. Note that the MATLAB results have been down-sampled to every fourth point for clarity.

4 Discussion

The steady-state validation results of Section 3.1 demonstrate excellent agreement with both the analytical solution and the steady ANSYS analysis, to within 0.1% for reasonable element counts. This validates the formulation and assembly of the thermal stiffness matrix in the MATLAB FEM code listed in the Appendix, since the appropriate solution is computed. It also verifies that the time integration algorithm and thermal capacitance formulation/assembly is functioning correctly, since a steady-state solution was extracted from the transient results by computing the transient temperature field until steady-state conditions were achieved.

The transient results, on the other hand, show slight variation, as shown in Fig. 6. This results from ANSYS's selection of a diagonal capacitance matrix for the `PLANE77` element [9] rather than the consistent capacitance matrix used in the MATLAB FEM code listed in the Appendix. However, this error is also small, amounting to a maximum percent difference over the full 3000 s run time of 1.4%.

References

[1] Incropera, F. P., DeWitt, D. P., Bergman, T. L., and Lavine, A. S., 2007. *Introduction to Heat Transfer*, 5th ed. John Wiley and Sons, Hoboken, NJ.

[2] Lewis, R. W., Nithiarasu, P., and Seetharumu, K. *Fundamentals of the Finite Element Method for Heat and Fluid Flow*. John Wiley and Sons.

[3] Komzsik, L. *Applied Calculus of Variations for Engineers*. CRC Press.

[4] Prantil, V., 2012. Weak Form for a Nonlinear Thermal Analysis. Milwaukee School of Engineering.

[5] Logan, D. L., 2012. *A First Course in the Finite Element Method*, 5th ed. Cengage Learning, Stamford, CT.

[6] Prantil, V., 2011. ME 460 Course Notes. Milwaukee School of Engineering.

[7] Carslaw, H., and Jaeger, J., 1959. *Conduction of Heat in Solids*, 2nd ed. Oxford, London.

[8] ANSYS, INC., 2009. *Thermal Analysis Guide*. Canonsburg, PA. Release 12.1.

[9] ANSYS, INC., 2010. *ANSYS Mechanical APDL Element Reference*. Canonsburg, PA. Release 13.0.

Appendix: MATLAB Code Listing

The following MATLAB m-code shows the script and functions used to compute the transient temperature response of the thin copper plate.

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Course Project <HeatTransfer.m>        %
% Course: ME 498 001                     %
% Author: P. Gessler <gesslerp@msoe.edu> %
% Date: 10 August 2012                   %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
clear all; close all; clc; % Clean up from previous run

%% DOMAIN SPECIFICATION
xmin = 0;    % (m) minimum x-extent of domain
xmax = 1;    % (m) maximum x-extent of domain
ymin = 0;    % (m) minimum y-extent of domain
ymax = 1;    % (m) maximum y-extent of domain
thck = 0.01; % (m) plate thickness (uniform)

%% MATERIAL PROPERTY SPECIFICATION
% all properties for Copper at 300 K
rho = 8933; % (kg/m^3) density
k = 401;    % (W/m/K)  thermal conductivity
cp = 385;   % (J/kg/K) specific heat capacity

%% DISCRETIZATION SPECIFICATION
nx = 50; % (-) number of elements, x-direction
ny = nx; % (-) number of elements, y-direction; using uniform mesh

%% BOUNDARY CONDITION SPECIFICATION
% these are set for the validation case
Tl = 0;   % (degC) left wall temperature
Tr = Tl;  % (degC) right wall temperature
Tt = Tl;  % (degC) top wall temperature
Tb = 50;  % (degC) bottom wall temperature

%% INITIAL CONDITION SPECIFICATION
Ti = Tl; % (degC) initial temperature at all nodes

%% SOLVER PARAMETER SPECIFICATION
ti = 0;     % (s) initial time
tf = 3000;    % (s) final time
dt = 5;  % (s) time step
beta = 1/2; % Newmark-Beta integration

%% MESH GENERATION
xdist = abs(xmax-xmin); % (m) size of domain, x-direction
ydist = abs(ymax-ymin); % (m) size of domain, y-direction
nel = 2*nx*ny           % (-) number of elements
nnd = (nx+1)*(ny+1);    % (-) number of nodes
dx = xdist/nx;          % (m) element size, x-direction
dy = ydist/ny;          % (m) element size, y-direction
vert = zeros(nel,6);    % (m) initialize array of vertex locations
```

```
index = 1;              % create element 1 first
for j=1:ny
    for i=1:nx
        % assign vertices at each element in a 'row'
        vert(index,:) = [xmin+dx*(i-1),ymin+dy*(j-1),xmin+dx*i, ...
            ymin+dy*j,xmin+dx*(i-1),ymin+dy*j];
        vert(index+1,:) = [xmin+dx*(i-1),ymin+dy*(j-1),xmin+dx*i, ...
            ymin+dy*(j-1),xmin+dx*i,ymin+dy*j];
        index = index + 2; % jump to next 'row' of elements
    end
end
xvec = xmin:dx:xmax;
yvec = ymin:dy:ymax;

%% ELEMENTAL MATRIX CONSTRUCTION
D = k*[1,0;0,1];      % material property matrix, isotropic
% initialize arrays for element-dependent matrices
A = zeros(nel,1);     % element area
B = zeros(2,3,nel);   % B matrix; intermediate step
Ke = zeros(3,3,nel); % Ke elemental stiffness matrix
Ce = zeros(3,3,nel); % Ce elemental capacitance matrix
for el=1:nel % loop over all elements
    A(el) = GetArea(vert(el,1),vert(el,2),vert(el,3), ...
        vert(el,4),vert(el,5),vert(el,6)); % compute element areas
    % build elemental B matrices, using shape functions
    B(:,:,el) = 1/2/A(el) * ...
        [ GetBeta(vert(el,4),vert(el,6)), ...
          GetBeta(vert(el,6),vert(el,2)), ...
          GetBeta(vert(el,2),vert(el,4));
          GetGamma(vert(el,5),vert(el,3)), ...
          GetGamma(vert(el,1),vert(el,5)), ...
          GetGamma(vert(el,3),vert(el,1)) ];
    % build elemental stiffness matrices
    Ke(:,:,el) = thck*A(el)*transpose(B(:,:,el))*D*B(:,:,el);
    Ce(:,:,el) = rho*cp*A(el)*thck/12 * [ 2, 1, 1;
                                          1, 2, 1;
                                          1, 1, 2 ];
end

%% GLOBAL MATRIX ASSEMBLY
% for this validation case, interior nodes are active DOF
% all boundary nodes have fixed temperature
ID = zeros(1,nnd); % initialize ID vector
bID = zeros(1,nnd); % initialize boundary ID vector
aDOF = 0; % start with the first active degree of freedom
bDOF = 0; % start with the first boundary degree of freedom
for node=1:nnd % loop over all nodes
    if node <= nx+1 || node > nnd-(nx+1) % catch top/bottom nodes
        ID(node) = 0; % no active DOF here
        bDOF = bDOF+1; % increment current boundary DOF count
        bID(node) = bDOF; % boundary DOF here
    elseif any(ismember(nx+2:nx+1:nnd,node)) % catch left edge nodes
        ID(node) = 0; % no active DOF here
        bDOF = bDOF+1; % increment current boundary DOF count
        bID(node) = bDOF; % boundary DOF here
    elseif any(ismember(2*nx+2:nx+1:nnd,node)) % catch right edge nodes
        ID(node) = 0; % no active DOF here
        bDOF = bDOF+1; % increment current boundary DOF count
        bID(node) = bDOF; % boundary DOF here
    else
        aDOF = aDOF+1; % increment current active DOF count
        ID(node) = aDOF; % place aDOF number into ID array at node
        bID(node) = 0; % no boundary DOF here
    end
end
fID = find(ID); % for indexing later
fbID = find(bID); % for more indexing later

LM = zeros(3,nel); % initialize LM array
bLM = zeros(3,nel); % initialize boundary LM array
% map element nodes to global node numbers
nodal = zeros(nel,3);
for row=1:ny % loop over all rows of elements
    for col=1:2*nx % loop over all columns of elements
        nodal(col+(2*nx)*(row-1),1) = ceil(col/2) + (nx+1)*(row-1);
        if floor(col/2) ~= col/2
            nodal(col+(2*nx)*(row-1),2) = nodal(col+(2*nx)*(row-1),1) + nx + 2;
            nodal(col+(2*nx)*(row-1),3) = nodal(col+(2*nx)*(row-1),2) - 1;
        else
            nodal(col+(2*nx)*(row-1),2) = nodal(col+(2*nx)*(row-1),1) + 1;
            nodal(col+(2*nx)*(row-1),3) = nodal(col+(2*nx)*(row-1),2) + 1 + nx;
        end
    end
end
nodal = nodal'; % flip matrix to align with LM array
for node=1:3 % loop through nodes in each element
    for elem=1:nel % loop over elements
        LM(node,elem) = ID(nodal(node,elem)); % place aDOF into LM array
        bLM(node,elem) = bID(nodal(node,elem)); % place bDOF into bLM array
    end
end
% assemble global matrices
Kt = zeros(nnd); % initialize global stiffness sized based on nnd
Ct = zeros(nnd); % initialize global capacitance sized based on nnd
for elem=1:nel % loop over all elements
    for iindex=1:3     % loops over elemental DOF
        for jindex=1:3 % in two directions (Ke, Ce matrices)
            iplace = nodal(iindex,elem);
            jplace = nodal(jindex,elem);
            if iplace ~=0 && jplace ~=0 % only place if active
                Kt(iplace,jplace) = Kt(iplace,jplace) + ...
                    Ke(iindex,jindex,elem);
                Ct(iplace,jplace) = Ct(iplace,jplace) + ...
                    Ce(iindex,jindex,elem);
            end
        end
    end
end
```

```
Kb = zeros(aDOF,bDOF); Kb = Kt(fID,fbID); % create boundary stiffness
Kt = Kt(fID,fID); % reduce stiffness
Ct = Ct(fID,fID); % reduce capacitance

%% SOLUTION COMPUTATION
t = ti; % (s) initialize time
i = 2; % initial time index (initial conditions placed at i=1)
Fq = zeros(nnd,1); % set applied heat fluxes (zero for validation)
T = zeros(nnd,1); % initialize initial temperature
for node=1:nnd % loop over all nodes
    if node <= nx+1 % bottom row of nodes
        T(node) = Tb+50; % set boundary condition
    elseif node > nnd-(nx+1) % top row of nodes
        T(node) = Tt; % set boundary condition
    elseif any(ismember(nx+2:nx+1:nnd,node)) % left edge nodes
        T(node) = Tl; % set boundary condition
    elseif any(ismember(2*nx+2:nx+1:nnd,node)) % right edge nodes
        T(node) = Tr; % set boundary condition
    else % interior nodes
        T(node) = Ti; % set initial condition
    end
end
while max(t) < tf % run to max time
    t(i) = t(i-1) + dt; % compute new time
    T(:,i) = T(:,i-1); % copy solution to new time step
    for node=1:nnd % loop over all nodes
        if node <= nx+1 % bottom row of nodes
            T(node,i) = Tb+50*cosd(t(i)); % set boundary condition
        end
    end
    Tbound = T(fbID,i);
    KbTb = Kb*Tbound;
    Fq(:,i) = zeros(nnd,1); % for validation, applied always 0
    Fq(fID,i) = Fq(fID,i) - KbTb; % add 'forces' from prescribed temperatures
    Cdt = Ct/dt; % intermediate computation
    T(fID,i) = (Cdt + beta*Kt)\((Cdt - (1-beta)*Kt)*T(fID,i-1) + ...
        (1-beta)*Fq(fID,i-1) + beta*Fq(fID,i));
    % prepare plot
    Temp = zeros(nx+1,ny+1);
    Temp(:) = T(:,i);
    Temp = Temp';
    figure(1)
    %contour(xvec,yvec,Temp,0:10:100);
    surf(xvec,yvec,Temp); view(159,24);
    title(num2str(max(t)));
    i = i + 1;
end

%% OUTPUT RESULTS FOR VALIDATION
maxT = T(ceil(nnd/2),:)';
t = t';
output = [t maxT];
dlmwrite('matlab.dat',output,' ');
```

The function GetArea was used to compute the area of an element given its three vertices.

```
function A = GetArea(x1,y1,x2,y2,x3,y3)
% The function GetArea returns the area of a triangle given the
% coordinates of the three vertices of the triangle.
% The function syntax is: GetArea(x1,y1,x2,y2,x3,y3)
A = abs((x1*(y2-y3) + x2*(y3-y1) + x3*(y1-y2))/2);
end
```

The functions GetBeta and GetGamma were used to compute the values within the $[B]$ matrix, listed in Eq. (11). Note that the coefficients $\alpha_{i,j,m}$ were not required since a basic linear basis function was chosen.

```
function b = GetBeta(ynext,ynext2)
b = ynext-ynext2;
end

function c = GetGamma(xprev,xnext)
c = xprev-xnext;
end
```