UNIVERSITAT POLITÈCNICA DE CATALUNYA
BARCELONATECH
Escola Superior d'Enginyeries Industrial,
Aeroespacial i Audiovisual de Terrassa

Computational aerospace engineering

Assingment 2

Computational heat conduction analysis

Andreu Craus Vidal

Roger Tatjé i Ramos

Greta- Group 1

Professor

Joaquín A. Hernández Ortega
Raúl Rubio Serrano

Thursday 2nd January, 2025

Contents

List of Figures

Statement of the assignment

The goal of this assignment is to develop a Matlab program able to solve any two-dimensional heat conduction problem using bilinear quadrilateral elements.

To test the performance of the program, consider the heat conduction problem depicted in Figure 1. The coordinates are given in metres. The conductivity matrix is isotropic, with

$$\boldsymbol{\kappa} = \kappa \cdot \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix},$$

and $\kappa = 10\,\mathrm{W/^\circ C}$. The temperature $u = 0$ is prescribed along edges AB and AD. The heat fluxes $\mathbf{q} \cdot \mathbf{n} = 0$ and $\mathbf{q} \cdot \mathbf{n} = 20\,\mathrm{W/m}$ are prescribed on edges BC and CD, respectively. A constant heat source $f = 4\,\mathrm{W/m^2}$ is applied over the plate.

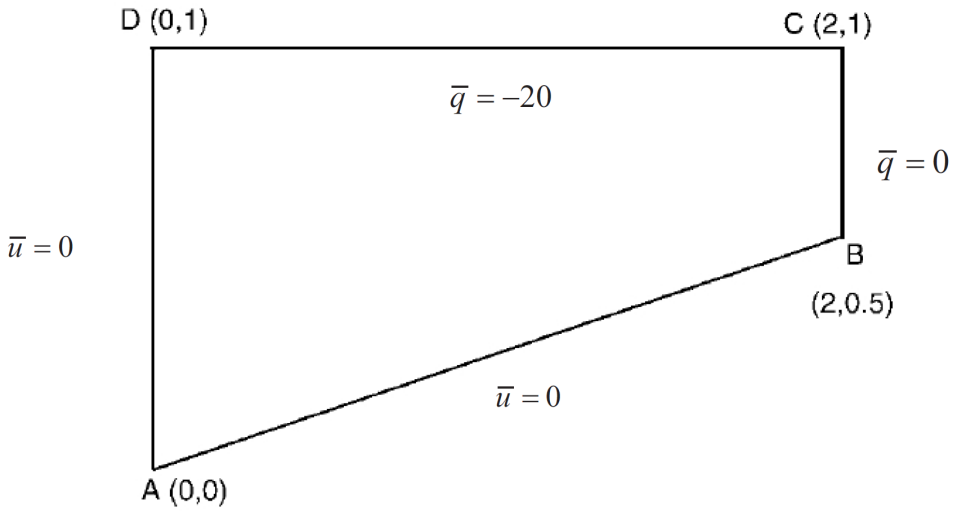

Figure 1: Heat conduction problem geometry.

To assess convergence upon mesh refinement, launch 4 different analyses with increasing numbers of finite elements. In particular, use structured meshes with:

- $n_{\mathrm{el}} = 1$ element,
- $n_{\mathrm{el}} = 5 \times 5 = 25$ elements,
- $n_{\mathrm{el}} = 10 \times 10 = 100$ elements,
- $n_{\mathrm{el}} = 30 \times 30 = 900$ elements.

For these three cases, plot the distribution of temperature along the edge DC in the same graph.

1 Part 1: Basic Tasks

A 2D heat conduction analysis using finite elements showed that finer meshes improve accuracy in temperature and heat flux distributions, with errors diminishing as mesh density increases. The 30x30 mesh provided the most detailed and accurate results.

1.1 Pre-Processing - Mesh generation

The first step involves generating the finite element mesh on the specified surface while assigning the geometric and physical parameters, as well as the boundary conditions, defined in the problem statement. For this purpose, the GID CIMNE finite element software is utilized.

The process begins by defining the geometry of the surface, which is illustrated in Figure 2.

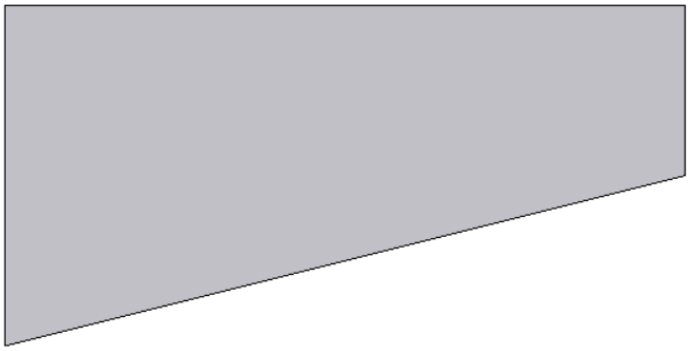

Figure 2: Geometric representation of the surface using GID CIMNE.

After the geometry has been specified, the next step is to create the mesh. The mesh is structured and consists of quadrilateral elements. Figure 3 provides an example of a mesh for the case $n_{\text{el}} = 10 \times 10 = 100$ elements, where the elements and nodes are labeled and the material properties are assigned appropriately.

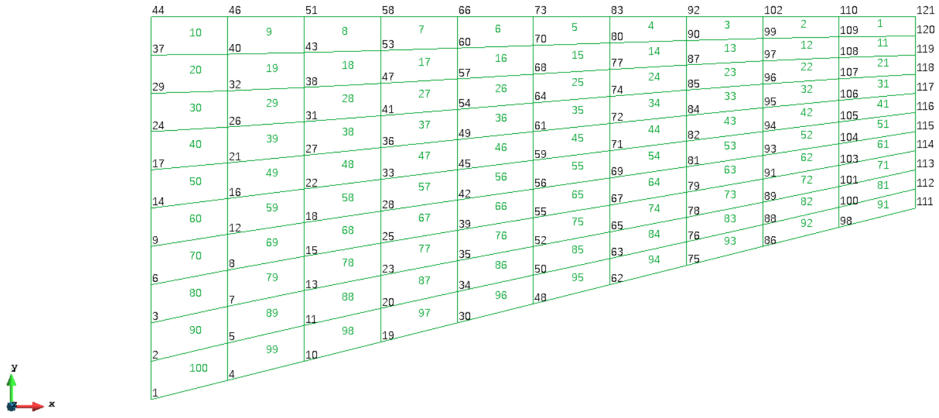

Figure 3: Structured quadrilateral mesh with material properties assigned.

Once all required meshes have been created, the element coordinates are exported into a `.msh` file. This file will be utilized in MATLAB to perform computations. The next phase involves developing MATLAB routines for finite element analysis to examine heat conduction over the surface under the given conditions.

1.2 Development of the Finite Element Program

While most of the core routines required for analyzing a generic 2D heat conduction system, as described in the problem statement, were provided, additional routines needed to be developed to complete the analysis. Specifically, new functions were implemented for assembling the global conductance matrix, constructing the global flux source vector, defining the shape functions routine, solving the resulting system of equations, and calculating the heat flux vector at the Gauss points of the system. The following subsections provide a detailed explanation of the implementation of these functions, which can also be found in the Appendix of this report.

1.2.1 Global Conductance Matrix Assembly ($\boldsymbol{K}$)

The code developed for assembling the global conductance matrix is detailed in Appendix A.1. This function computes the overall conductance matrix ($\boldsymbol{K}$) by utilizing the conductance matrix of each element, obtained through the pre-existing function `ComputeKeMatrix`. The assembly process involves iterating through the number of elements, with the global matrix $\boldsymbol{K}$ sequentially storing the conductance matrix entries corresponding to each element.

It is worth noting that to ensure the proper functioning of the `ComputeKeMatrix` routine, a new function, `Quadrilateral4NInPoints`, had to be implemented. This function calculates the shape functions for bilinear quadrilateral elements. The outputs of `Quadrilateral4NInPoints` are then utilized by the function `ComputeElementShapeFun`, which is essential for deriving the element-level conductance matrix ($\boldsymbol{K}_e$).

The details of the implementation for the `Quadrilateral4NInPoints` function are described in the next subsection.

1.2.2 Shape Function Routine for Bilinear Quadrilateral Elements

The MATLAB script created for this functionality is detailed in Appendix A.2. This routine calculates the weights and positions required for Gaussian quadrature as applied to 2D bilinear quadrilateral elements. Using the positional values of the Gauss points, the function generates the elemental shape function matrices, denoted as $\boldsymbol{N}$ and $\boldsymbol{B}$, which are defined as follows:

$$\boldsymbol{N} = \frac{1}{4}\begin{bmatrix}(1-\xi)(1-\eta) & (1+\xi)(1-\eta) & (1+\xi)(1+\eta) & (1-\xi)(1+\eta)\end{bmatrix}$$

$$\boldsymbol{B} = \frac{1}{4}\begin{bmatrix}-(1-\eta) & (1-\eta) & (1+\eta) & -(1+\eta) \\ -(1-\xi) & -(1+\xi) & (1+\xi) & (1-\xi)\end{bmatrix}$$

1.2.3 Assembly of the Global Flux Source Vector (F_s)

The MATLAB code developed for constructing the global flux source vector is provided in Appendix A.3. Similar to the assembly process for the global conductance matrix, the routine loops

through each element to compute and store the corresponding flux source values. This is accomplished using the `ComputeFseVector` function, which operates similarly to the `ComputeKeMatrix` routine described earlier.

1.2.4 Solution of the Final System of Equations

The `SolveHe` function, described in Appendix A.4, is responsible for solving the final system of equations. The system is expressed in matrix form as:

$$\begin{bmatrix} K_{RR} & K_{RL} \\ K_{LR} & K_{LL} \end{bmatrix} \begin{bmatrix} d_R \\ d_L \end{bmatrix} = \begin{bmatrix} f_R \\ f_L \end{bmatrix}$$

The solution procedure carried out by this function is described mathematically as follows:

$$d_L = K_{LL}^{-1} \cdot (f_L - K_{LR} \cdot d_R)$$

1.2.5 Computation of the Heat Flux Vector at Each Gauss Point

The implementation of this function, detailed in Appendix A.5, calculates the heat flux vector at the Gauss points of the system. This computation is based on the following equations:

$$\mathbf{c}^T \cdot (K \cdot \mathbf{d} - \mathbf{F}) = 0$$

$$\nabla u^e = \boldsymbol{B}^e \cdot \mathbf{d}^e$$

1.3 Program Results: Post-Processing

After implementing the required functions and ensuring the MATLAB code operates as expected, this section presents the post-processing results obtained using the GID CIMNE software, which processes the output from the MATLAB program. The analysis is conducted for the various meshes specified in the problem statement to evaluate the convergence of the results as the number of discretization elements increases.

1.3.1 First Mesh: 1x1 Quadrilateral Element

For the initial case, the generated mesh is depicted in Figure 4.

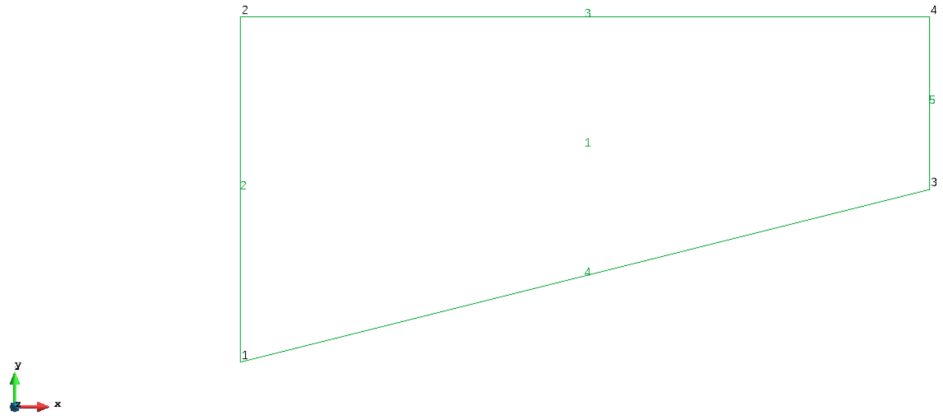

Figure 4: Mesh generated for the surface using a single quadrilateral element (1x1 mesh).

By running the program, the temperature distribution over the surface can be visualized using the GID post-processing tool. Figure 5 illustrates the temperature field simulation produced by the program.

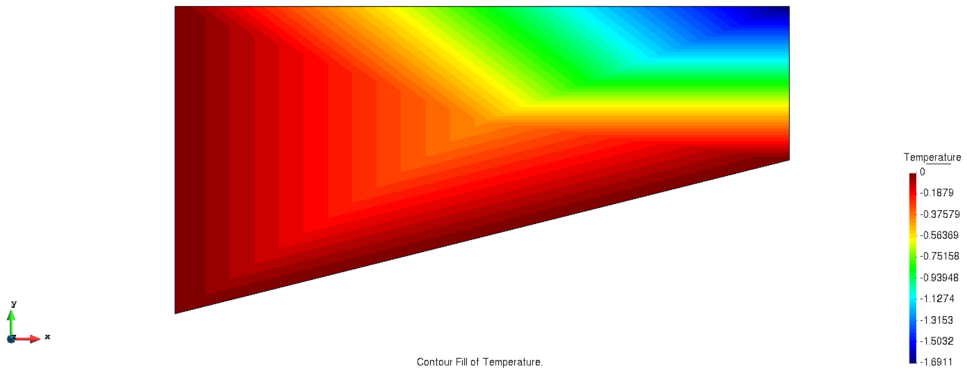

Figure 5: Simulated temperature gradient across the surface for a single quadrilateral element (1x1 mesh).

The isotherms for the single quadrilateral element case (1x1 mesh) are plotted to represent the lines of constant temperature across the surface. These lines provide a clear visualization of the temperature gradient distribution in this low-resolution discretization.

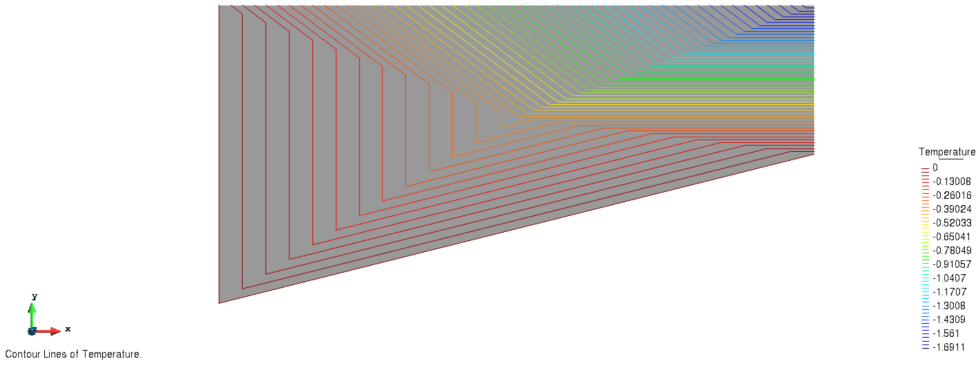

Figure 6: Isotherms of the temperature field for a single quadrilateral element (1x1 mesh).

1.3.2 Second Mesh: 5x5 Quadrilateral Elements (25 Elements)

For the second analysis, a mesh consisting of 25 elements has been created. Similar to the previous case, this is a structured mesh composed of quadrilateral elements, with 5 elements along each edge of the surface. The resulting mesh generated in GID is displayed in Figure 7.

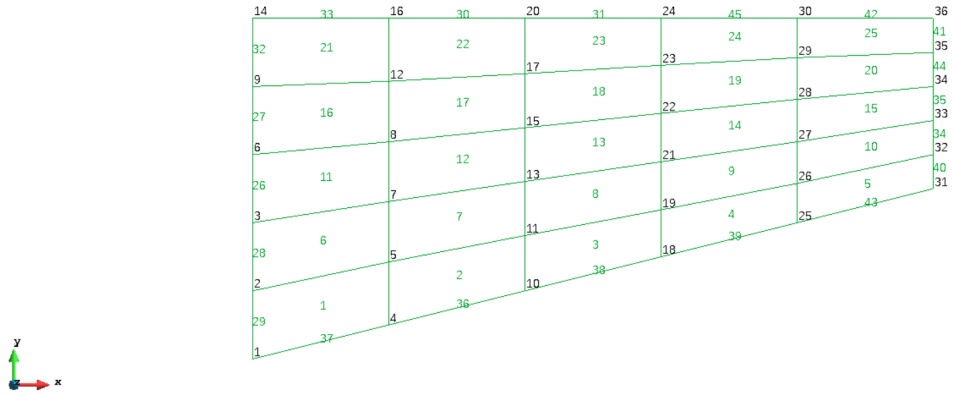

Figure 7: Mesh created for the surface with 25 quadrilateral elements (5x5 structured mesh).

Running the finite element program and processing the temperature field results using the GID post-processing tool produces the temperature distribution over the surface shown in Figure 8.

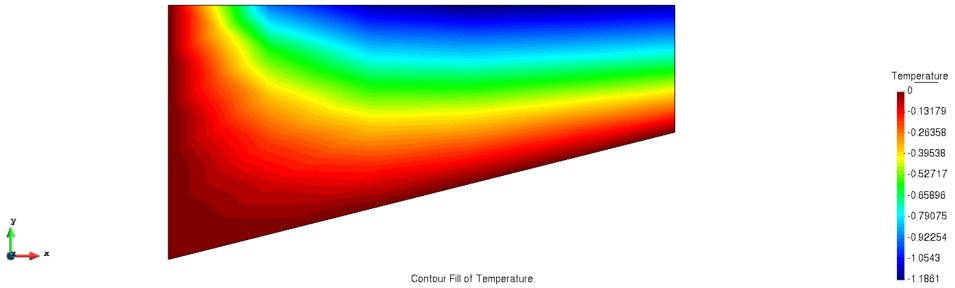

Figure 8: Simulated temperature gradient on the surface for 25 quadrilateral elements (5x5 mesh).

In the 5x5 quadrilateral element mesh, the isotherms become more defined compared to the previous case, showing a more detailed representation of the temperature distribution. This refinement highlights the improvement in capturing the temperature gradient.

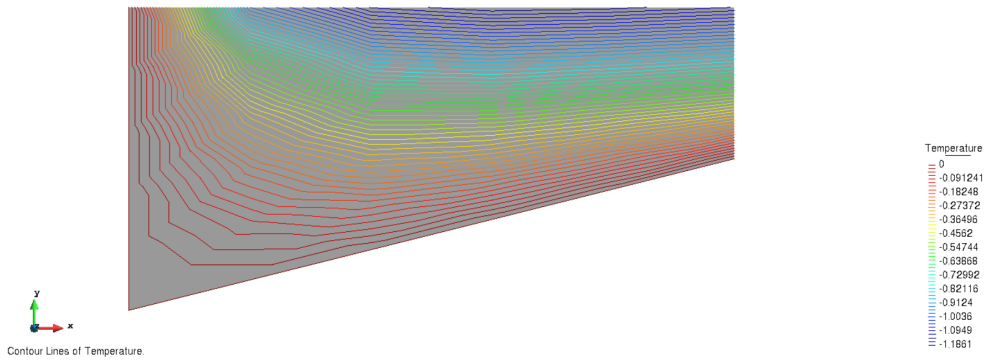

Figure 9: Isotherms of the temperature field for 25 quadrilateral elements (5x5 mesh).

1.3.3 Third Mesh: 10x10 Quadrilateral Elements (100 Elements)

The third analysis involves a structured mesh of quadrilateral elements, consisting of 100 discretization elements with 10 elements along each edge of the surface. The mesh created in GID is shown in Figure 10.

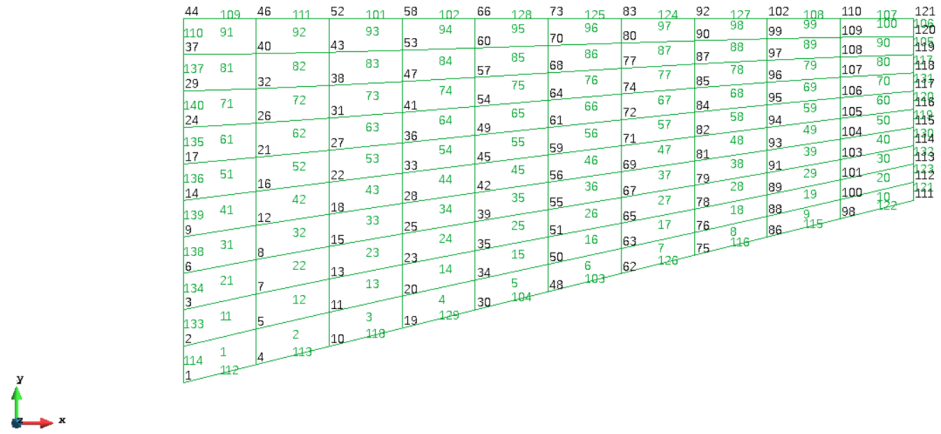

Figure 10: Mesh generated for the surface with 100 quadrilateral elements (10x10 structured mesh).

After running the MATLAB program and solving the system for this refined mesh, the output is saved in a `.res` file, which is processed using GID's post-processing functionality. The resulting temperature distribution across the surface is shown in Figure 11.

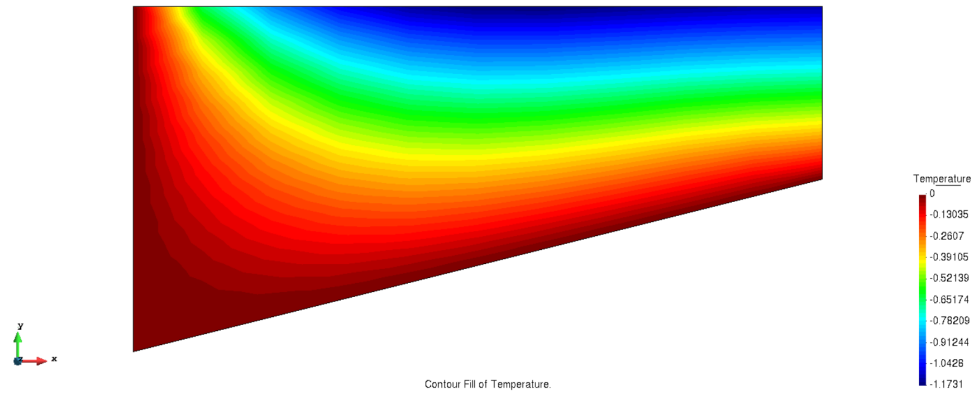

Figure 11: Simulated temperature gradient across the surface for 100 quadrilateral elements (10x10 mesh).

With the 10x10 quadrilateral element mesh, the isotherms are even more detailed, illustrating the convergence of the temperature field towards a more precise solution. The distribution aligns closely with the expected theoretical results.

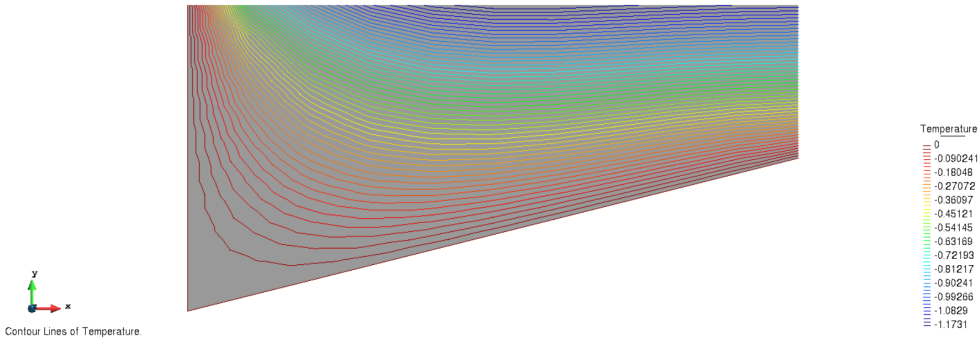

Figure 12: Isotherms of the temperature field for 100 quadrilateral elements (10x10 mesh).

1.3.4 Fourth Mesh: 30x30 Quadrilateral Elements (900 Elements)

The final mesh used for the heat conduction analysis is a structured quadrilateral elements mesh consisting of 900 elements, with 30 elements along each edge of the surface. The generated mesh for this case is depicted in Figure 13.

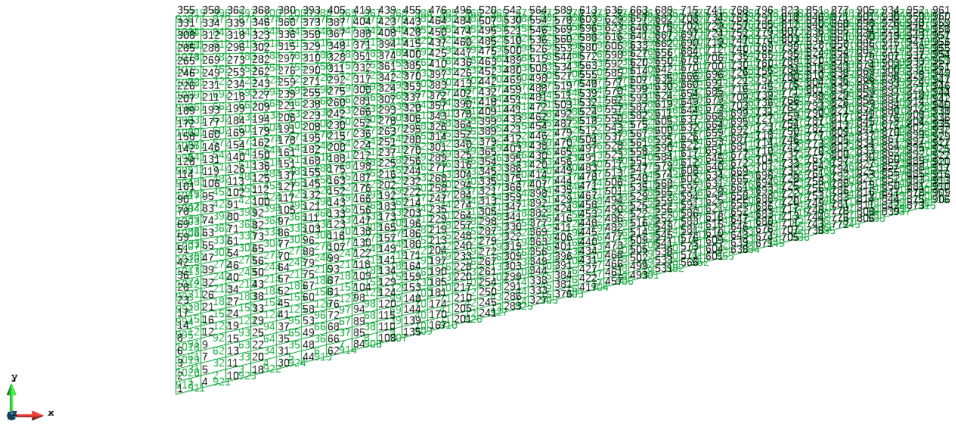

Figure 13: Mesh created for the surface with 900 quadrilateral elements (30x30 structured mesh).

Following the same procedure applied to the previous cases, the temperature gradient over the surface is simulated using the MATLAB program and processed through GID's post-processing tool. The resulting temperature field is shown in Figure 14.

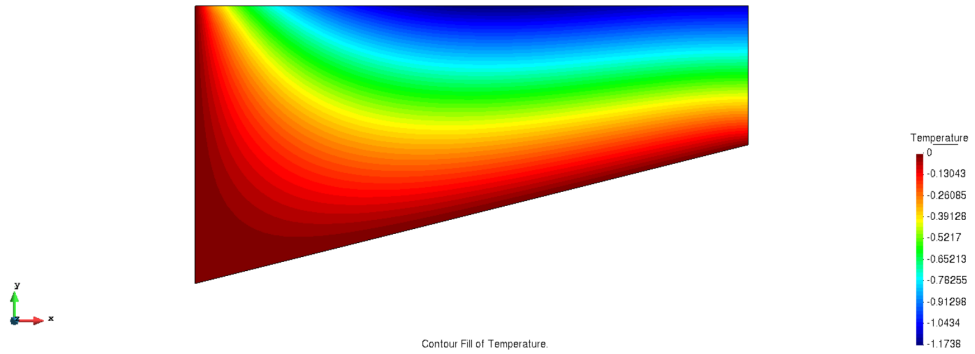

Figure 14: Simulated temperature gradient on the surface for 900 quadrilateral elements (30x30 mesh).

In the 30x30 quadrilateral element mesh, the isotherms depict a highly accurate temperature distribution. The finer discretization provides a detailed visualization, demonstrating minimal variations compared to the previous case.

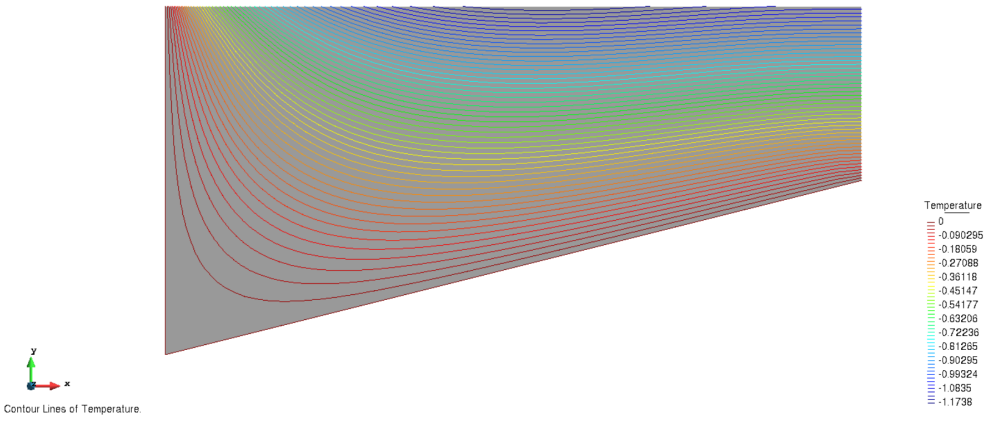

Figure 15: Isotherms of the temperature field for 900 quadrilateral elements (30x30 mesh).

1.4 Analysis of Convergence in the Finite Element Program

Finally, the temperature variation along the edge CD is analyzed using a post-processing graph generated by GID.

Based on the simulation results obtained in the previous section for different mesh discretizations, it was observed that the temperature distributions strongly depend on the characteristics of the discretization. When comparing the results across different discretizations, a significant difference in precision is noted between the first three cases. However, between the third and fourth discretizations, the differences are minimal, despite the considerable increase in the number of elements. This indicates that the relationship between the number of elements and mesh precision is not linear, and the results tend to converge as the number of finite elements increases.

To evaluate this convergence more effectively, the temperature distributions along edge CD from all simulations have been plotted together, as shown in Figure 16.

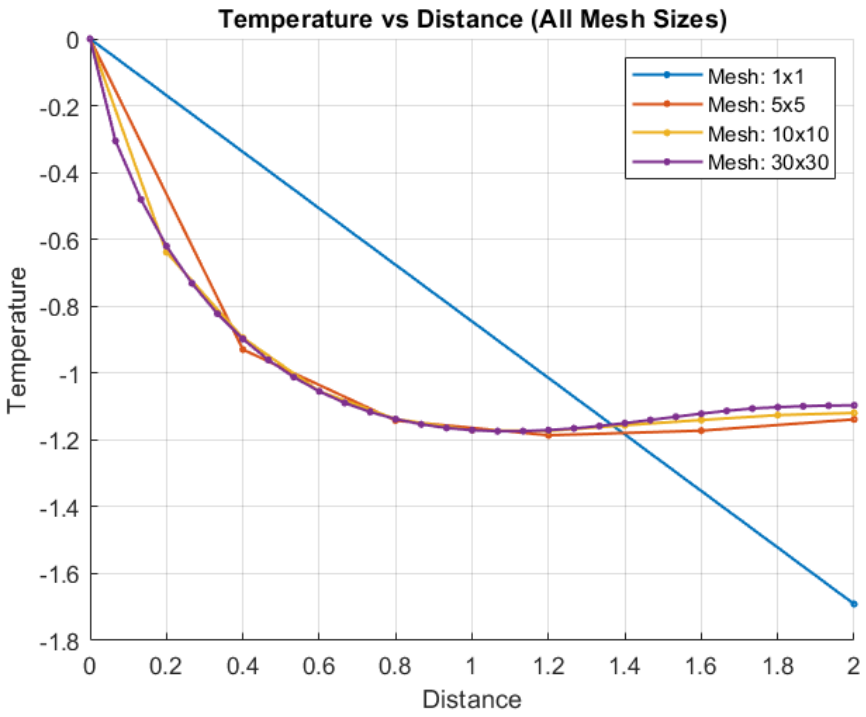

Figure 16: Comparison of temperature distributions along edge CD for different mesh discretizations.

As shown in the figure, the temperature values demonstrate a clear convergence as the number of discretization elements increases. The relative error between consecutive meshes decreases significantly, particularly for higher discretization levels, confirming that the results stabilize with finer meshes.

1.5 Heat Flux Across the Surface

This section presents the simulation results for the heat flux across the surface, as computed by GID. These results were obtained through the implementation of the `HeatFlux` function, which was described in the initial sections of this report.

The heat flux simulations for the various mesh discretizations analyzed are displayed in the following figures:

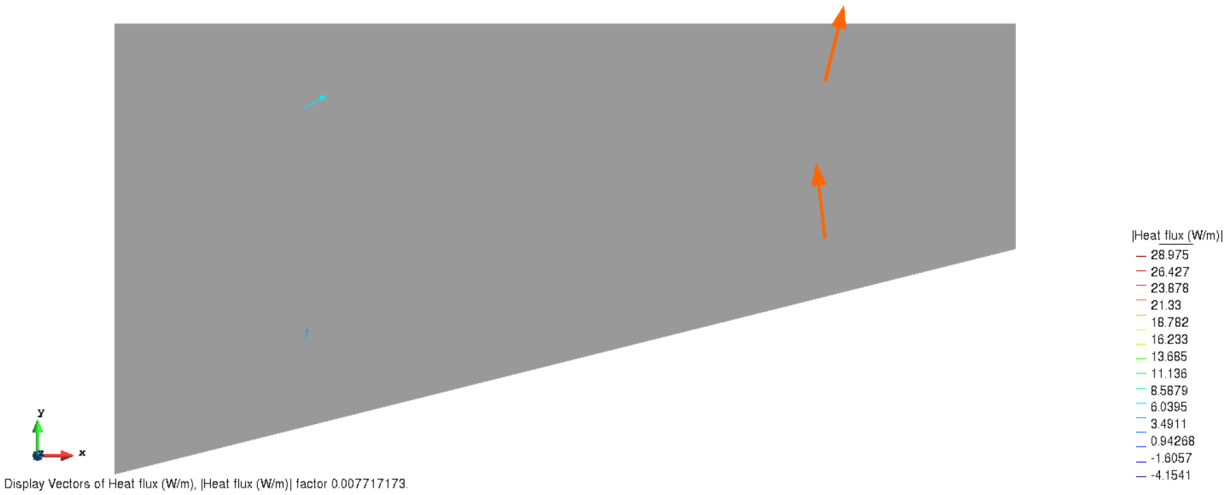

Figure 17: Simulated heat flux on the surface for 1 quadrilateral element (1x1 mesh).

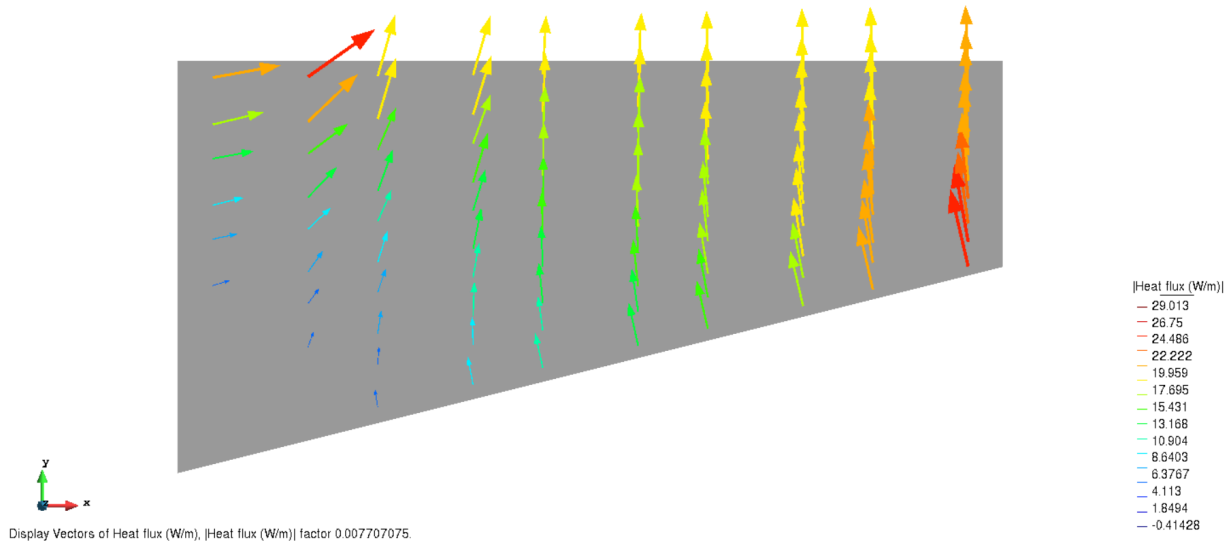

Figure 18: Simulated heat flux on the surface for 25 quadrilateral elements (5x5 mesh).

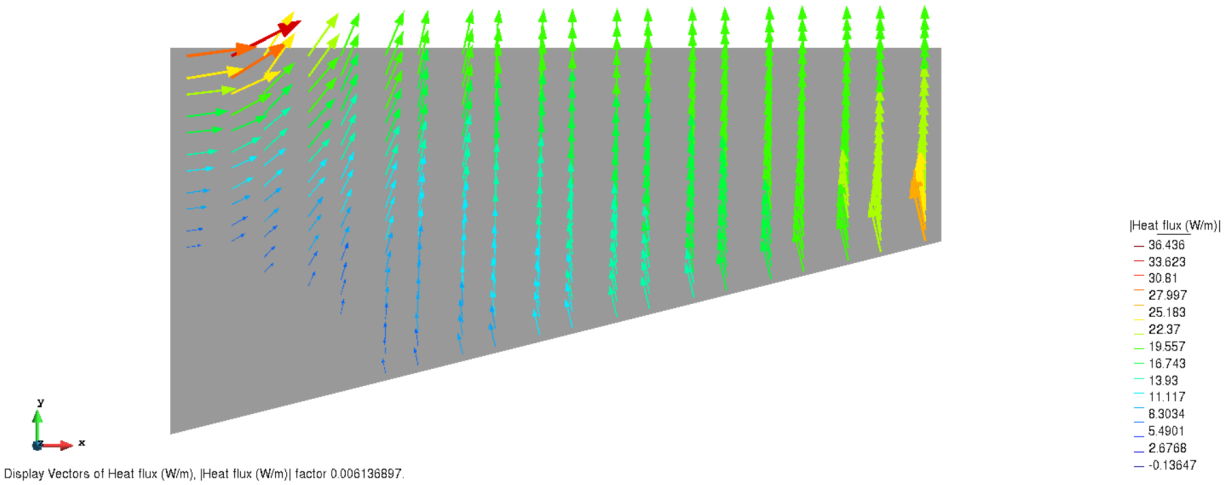

Figure 19: Simulated heat flux on the surface for 100 quadrilateral elements (10x10 mesh).

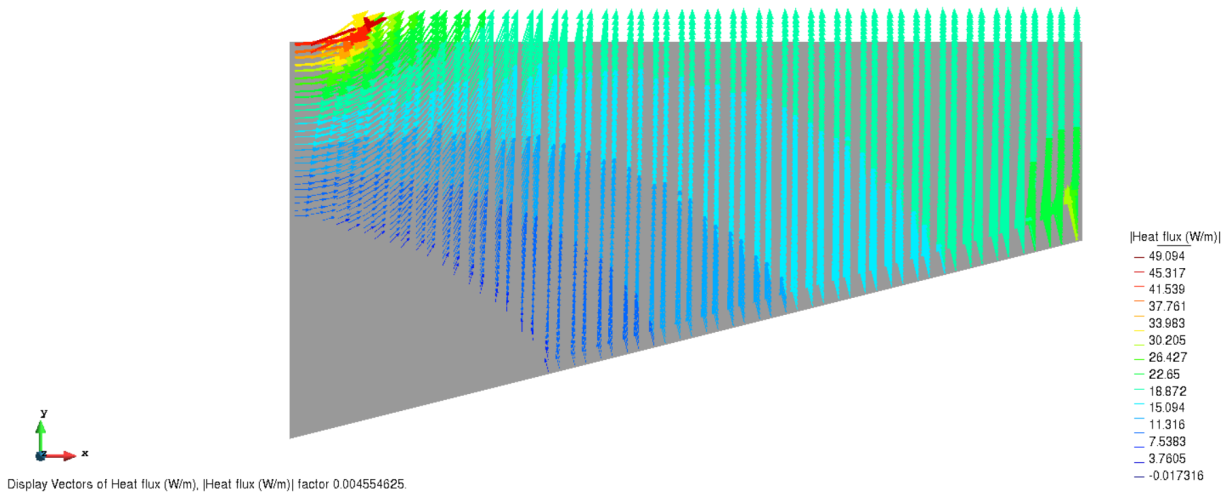

Figure 20: Simulated heat flux on the surface for 900 quadrilateral elements (30x30 mesh).

Based on the simulation results for the various discretizations, it is observed that the distribution of heat flux vectors across the surface becomes increasingly accurate as the number of discretization elements increases. It has been concluded that both the magnitude and direction of the heat flux vectors are represented with greater precision when finer meshes are used. Among the analyzed cases, the 900-element discretization provides the most detailed and accurate representation of the heat flux across the surface.

2 Part 2: Advanced Analysis

2.1 Convective Heat Transfer

In this advanced section, the problem from statement 1 is modified to include heat transfer through convection. This occurs due to the contact between the surface and a fluid, which generates a temperature gradient along the upper boundary. The updated boundary conditions are illustrated in Figure 21.

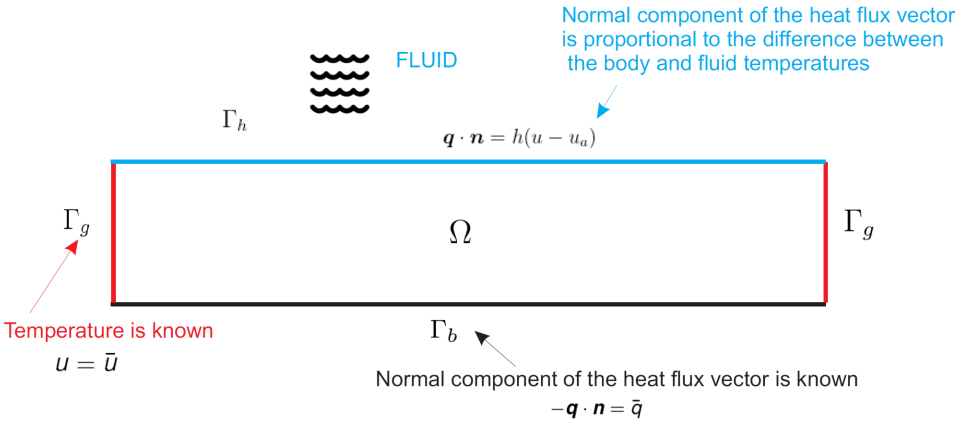

Figure 21: Definition of boundary conditions for the problem in Part 2.

The goal is to solve the Weak Form of this Boundary Value Problem (BVP) to determine u, such that:

$$\int_\Omega \nabla v^T \cdot \kappa \cdot \nabla u \, d\Omega = \int_\Omega v \cdot f \, d\Omega + \int_{\Gamma_b} v \cdot q \, d\Gamma, \quad f : \Omega \to \mathbb{R}, \forall v \in V, \, u \in S \tag{1}$$

The analysis requires examining the modifications in the global stiffness matrix $\boldsymbol{K}$ and the force vector $\boldsymbol{F}$ compared to the simpler heat conduction problem studied in Part 1. In this case, an additional boundary condition accounts for heat transfer via convection along one of the boundaries. The first step involves applying the Weak Formulation of the BVP for a standard linear heat conduction problem, as represented by Equation 1.

In the updated conditions described in this statement, an additional term $q = h \cdot (u - u_a)$ must be incorporated. By adding this term as an integral over the boundary, the Weak Form of the Boundary Value Problem (BVP) is reformulated as follows:

$$\int_\Omega \nabla v^T \cdot \kappa \cdot \nabla u \, d\Omega = \int_\Omega v \cdot f \, d\Omega + \int_{\Gamma_b} v \cdot q \, d\Gamma + \int_{\Gamma_a} v \cdot h \cdot (u - u_a) \, d\Gamma \tag{2}$$

Here, Γ_a represents the boundary surface where convection occurs between the solid and the fluid. Additionally, u_a is defined as the fluid temperature at the Gauss points. The formulation in Equation (2) can be rearranged as:

$$\int_\Omega \nabla v^T \cdot \kappa \cdot \nabla u \, d\Omega - \int_{\Gamma_a} v \cdot h \cdot (u - u_a) \, d\Gamma = \int_\Omega v \cdot f \, d\Omega + \int_{\Gamma_b} v \cdot q \, d\Gamma - \int_{\Gamma_a} v \cdot h \cdot u_a \, d\Gamma \tag{3}$$

This expression can also be reformulated in terms of the shape function matrices $\boldsymbol{N}$ and $\boldsymbol{B}$ as:

$$\begin{aligned} \boldsymbol{v}\Big\{ &\left(\int_\Omega \nabla \boldsymbol{B}^T \cdot \kappa \cdot \boldsymbol{B} \, d\Omega - \int_{\Gamma_a} \boldsymbol{N}^T \cdot h \cdot \boldsymbol{N} \, d\Gamma \right) \cdot \boldsymbol{u} \\ &= \int_\Omega \boldsymbol{N}^T \cdot f \, d\Omega + \int_{\Gamma_b} \boldsymbol{N}^T \cdot q \, d\Gamma - \int_{\Gamma_a} \boldsymbol{N}^T \cdot h \cdot u_a \, d\Gamma \Big\} \end{aligned} \tag{4}$$

From this formulation, the stiffness matrix $\boldsymbol{K}$ and force vector $\boldsymbol{F}$ can be derived as:

$$\boldsymbol{K} = \int_\Omega \nabla \boldsymbol{B}^T \cdot \kappa \cdot \boldsymbol{B} \, d\Omega - \int_{\Gamma_a} \boldsymbol{N}^T \cdot h \cdot \boldsymbol{N} \, d\Gamma \tag{5}$$

$$\boldsymbol{F} = \int_\Omega \boldsymbol{N}^T \cdot f \, d\Omega + \int_{\Gamma_b} \boldsymbol{N}^T \cdot q \, d\Gamma - \int_{\Gamma_a} \boldsymbol{N}^T \cdot h \cdot u_a \, d\Gamma \tag{6}$$

By comparing this formulation with the one provided in the problem statement, the following terms are concluded:

$$\boxed{\boldsymbol{K}_h = - \int_{\Gamma_a} \boldsymbol{N}^T \cdot h \cdot \boldsymbol{N} \, d\Gamma} \tag{7}$$

$$\boxed{\boldsymbol{F}_h = - \int_{\Gamma_a} \boldsymbol{N}^T \cdot h \cdot u_a \, d\Gamma} \tag{8}$$

2.2 Time-Dependent Heat Conduction

In this section, the goal is to determine the capacitance matrix $\boldsymbol{M}$ using the shape functions applied in the interpolation of a time-dependent heat conduction problem. The governing equation for the system is:

$$\boldsymbol{c}^T \cdot (\boldsymbol{M} \cdot \dot{\boldsymbol{d}} + \boldsymbol{L} \cdot \boldsymbol{d} - \boldsymbol{F}) = 0 \tag{9}$$

The Strong Form of the Boundary Value Problem (BVP) for this case is expressed as:

$$\begin{cases} \nabla \boldsymbol{q} - f = \rho \cdot c \cdot \frac{\partial u}{\partial t}, & \text{in } \Omega \times [0, T], \\ u = \bar{u}, & \text{on } \Gamma_g \times [0, T], \\ -\boldsymbol{q} \cdot \boldsymbol{n} = \bar{q}, & \text{on } \Gamma_b \times [0, T], \\ u(x, 0) = u_0(x), & \text{in } \Omega. \end{cases} \tag{10}$$

To derive the Weak Form of this problem, the Strong Form is multiplied by an arbitrary test function v and integrated over the domain Ω, yielding:

$$\int_{\Omega} v \cdot (\nabla \boldsymbol{q} - f)\, d\Omega = \int_{\Omega} v \cdot (\rho \cdot c \cdot \frac{\partial u}{\partial t})\, d\Omega \tag{11}$$

By applying integration by parts, the expression simplifies to:

$$\int_{\Gamma} v \cdot (-\boldsymbol{q})\, d\Gamma - \int_{\Omega} \nabla v \cdot \boldsymbol{q}\, d\Omega - \int_{\Omega} v \cdot f\, d\Omega = \int_{\Omega} v \cdot (\rho \cdot c \cdot \frac{\partial u}{\partial t})\, d\Omega \tag{12}$$

Using the definition $\boldsymbol{q} = -\kappa \nabla u$, the equation can be rewritten as:

$$\int_{\Gamma} v \cdot (-\boldsymbol{q})\, d\Gamma + \int_{\Omega} \nabla v \cdot (\kappa \nabla u)\, d\Omega - \int_{\Omega} v \cdot f\, d\Omega = \int_{\Omega} v \cdot (\rho \cdot c \cdot \frac{\partial u}{\partial t})\, d\Omega \tag{13}$$

Expressing this equation in terms of the shape function matrices $\boldsymbol{N}$ and $\boldsymbol{B}$:

$$\boldsymbol{c}^T \left[\left(-\int_{\Omega} \boldsymbol{N}^T \cdot \rho \cdot c \cdot \boldsymbol{N}\, d\Omega \right) \cdot \dot{\boldsymbol{d}} + \left(\int_{\Omega} \boldsymbol{B}^T \cdot \kappa \cdot \boldsymbol{B}\, d\Omega \right) \cdot \boldsymbol{d} - \left(\int_{\Gamma_b} \boldsymbol{N}^T \cdot \bar{q}\, d\Gamma + \int_{\Omega} \boldsymbol{N}^T \cdot f\, d\Omega \right) \right] = 0 \tag{14}$$

By comparing the terms in this equation with those in Equation (9), the capacitance matrix $\boldsymbol{M}$ is defined as:

$$\boxed{\boldsymbol{M} = -\int_{\Omega} \boldsymbol{N}^T \cdot \rho \cdot c \cdot \boldsymbol{N}\, d\Omega} \tag{15}$$

Appendix A Code

A.1 Code for section 1.2.1

```
    function K = ComputeK(COOR,CN,TypeElement, ConductMglo) ;
%%%%
% This subroutine   returns the global conductance matrix K (nnode x nnode)
% Inputs
% --------------
% 1. Finite element mesh
% -------------------
% COOR: Coordinate matrix (nnode x ndim)
% CN: Connectivity matrix (nelem x nnodeE)
% TypeElement: Type of finite element (quadrilateral,...)
% -----------
% 2. Material
% -----------
%  ConductMglo (ndim x ndim x nelem)  % Array of conductivity matrices
%%%%
 if nargin == 0
     load('tmp1.mat')
 end




%% COMPLETE THE CODE ....
%warning('You must program the assembly of the conductance matrix K !!')


% Dimensions of the problem
nnode = size(COOR,1);  % Number of nodes
ndim = size(COOR,2);   % Spatial Dimension of the problem  (2 or 3)
nelem = size(CN,1);   % Number of elements
nnodeE = size(CN,2) ; %Number of nodes per element

% Determine Gauss weights, shape functions and derivatives
TypeIntegrand = 'K';
[weig,posgp,shapef,dershapef] = ComputeElementShapeFun(TypeElement,nnodeE,
    TypeIntegrand) ;

%

% Assembly of matrix K
% ----------------
K = sparse(nnode,nnode) ;
% ......
% Loop through each element to assemble the global conductance matrix
% Assembly of the K matrix
for elem = 1:nelem
    % Nodes of the current element
    elemNodes = CN(elem, :);

    % Coordinates of the nodes of the current element
    elemCoords = COOR(elemNodes, :)';

    % Conductivity matrix of the current element
    elemConductivity = ConductMglo(:, :, elem);

    % Compute the local conductance matrix of the element
    localK = ComputeKeMatrix(elemConductivity, weig, dershapef, elemCoords);

    % Assemble into the global matrix
```

```
    K(elemNodes, elemNodes) = K(elemNodes, elemNodes) + localK;
end
end
```

A.2 Code for section 1.2.2

```
    function [weig, posgp, shapef, dershapef] = Quadrilateral4NInPoints()

 ndimensions = 2;
 nnodeE = 2^ndimensions;
 weig = ones(1, nnodeE);
 ngauss = length(weig);

 signsMatrix = [ -1  1  1 -1;  % signs of xi
                 -1 -1  1  1]; % signs of eta

 posgp = (1/sqrt(3)) * signsMatrix;

 shapef = zeros(ngauss, nnodeE);
 dershapef = zeros(ndimensions, nnodeE, ngauss);

 for gaussIdx = 1:ngauss
     xiVal = posgp(1, gaussIdx);
     etaVal = posgp(2, gaussIdx);

     shapef(gaussIdx, :) = (1 / nnodeE) * [
         (1 - xiVal) * (1 - etaVal),
         (1 + xiVal) * (1 - etaVal),
         (1 + xiVal) * (1 + etaVal),
         (1 - xiVal) * (1 + etaVal)
     ];

     dershapef(:, :, gaussIdx) = (1 / nnodeE) * signsMatrix .* [
         (1 - etaVal), (1 - etaVal), (1 + etaVal), (1 + etaVal);
         (1 - xiVal),  (1 + xiVal),  (1 + xiVal),  (1 - xiVal)
     ];
 end

 end
```

A.3 Code for section 1.2.3

```
    function Fs = ComputeFs(COOR,CN,TypeElement, fNOD) ;
% This subroutine   returns the  heat source contribution (Fs)    to the
% global flux vector. Inputs
% --------------
% 1. Finite element mesh
% -------------------
% COOR: Coordinate matrix (nnode x ndim)
% CN: Connectivity matrix (nelem x nnodeE)
% TypeElement: Type of finite element (quadrilateral,...)
% -----------
% 2. Vector containing the values of the heat source function at the nodes
% of the mesh
% -----------
%  fNOD (nnode x 1)  %
%%%%

% Dimensions of the problem
nnode = size(COOR,1); ndim = size(COOR,2); nelem = size(CN,1); nnodeE = size(CN
    ,2) ;

```

```
20
21
22
23 % Define the type of integrand for computing shape functions
24 TypeIntegrand = 'RHS';
25 [weig, posgp, shapef, dershapef] = ComputeElementShapeFun(TypeElement, nnodeE,
       TypeIntegrand);
26 Fs = zeros(nnode,1) ;
27
28 % Assembly of the Fs vector
29 for elemIdx = 1:nelem
30     % Nodes of the current element
31     elemNodes = CN(elemIdx, :);
32
33     % Heat source values at the current element's nodes
34     elemHeatSource = fNOD(elemNodes);
35
36     % Coordinates of the nodes of the current element
37     elemCoords = COOR(elemNodes, :)';
38
39     % Compute the local heat source contribution for the element
40     localFs = ComputeFseVector(elemHeatSource, weig, shapef, dershapef,
       elemCoords);
41
42     % Assemble the local contributions into the global flux vector
43     for localNodeIdx = 1:nnodeE
44         Fs(elemIdx) = Fs(elemIdx) + localFs(localNodeIdx);
45     end
46 end
```

A.4 Code for section 1.2.4

```
1     function [d qheatGLO posgp] = SolveHE(K,Fs,Fbnd,dR,rnod,COOR,CN,TypeElement,
      ConductMglo) ;
2 % This function returns  n returns the (nnode x 1) vector of nodal temperatures (
      d),
3 % as well as the vector formed by the heat flux vector at all gauss
4 %%% points  (qheatGLO)
5 % Input data
6 % K = Global conductance matrix   (nnode x nnode)
7 % Fs = Global source flux vector  (nnode x 1)
8 % Fbnd = Global boundary flux vector  (nnode x 1)
9 % rnod = Set of nodes at which temperature is prescribed
10 % dR = Vector of prescribed displacements
11 % ----------------------
12 nnode = size(COOR,1); ndim = size(COOR,2); nelem = size(CN,1); nnodeE = size(CN
      ,2) ;     posgp=[] ;
13 % Solution of the system of FE equation
14 % Right-hand side
15 F = Fs + Fbnd ;
16 % Set of nodes at which temperature is unknown
17 lnod = 1:nnode ;
18 lnod(rnod) = [] ;
19 % dL =  Kll^{-1}*(Fl .Klr*dR)
20  dL=K(lnod,lnod)\(F(lnod)-K(lnod,rnod)*dR);
21
22 % Vector of   temperatures
23 d = zeros(nnode,1) ;
24 d(lnod)= dL ;
25 d(rnod) = dR ;
26
27 %%%% COmputation of heat flux vector at each gauss point
28 disp('Computation of heat flux vector at each gauss point and elements')
```

```
ngaus = size(posgp,2) ; qheatGLO = zeros(ngaus*ndim,nelem);
% Computing this array is not mandatory....
[qheatGLO posgp]= HeatFlux_2(COOR,CN,TypeElement,ConductMglo,d) ;
```

A.5 Code for section 1.2.5

```
    function [qFluxGlobal, gaussPositions] = HeatFlux(COOR, CN, ElementType,
    GlobalConductivity, temperatureVector)
% This function computes the heat flux vector at each Gauss point for all
    elements.
%
% Input:
% COOR - Coordinates of the nodes (nnode x ndim)
% CN - Connectivity matrix (nelem x nnodeE)
% ElementType - Type of finite element
% GlobalConductivity - Conductivity matrices for all elements
% temperatureVector - Nodal temperature vector (nnode x 1)
%
% Output:
% qFluxGlobal - Heat flux vector at Gauss points (ngauss*ndim x nelem)
% gaussPositions - Positions of the Gauss points

% Problem dimensions
numNodes = size(COOR, 1);
numDims = size(COOR, 2);
numElements = size(CN, 1);
nodesPerElement = size(CN, 2);

% Initialize global heat flux vector
qFluxGlobal = zeros(8, numElements);

% Define the integrand type and compute shape functions
integrandType = 'K';
[weights, gaussPositions, shapeFunctions, shapeFunctionDerivatives] =
    ComputeElementShapeFun(ElementType, nodesPerElement, integrandType);

% Loop over each element to compute heat fluxes
for elemIdx = 1:numElements
    % Nodes and temperatures of the current element
    elementNodes = CN(elemIdx, :);
    localTemperatures = temperatureVector(elementNodes);
    localConductivity = GlobalConductivity(:, :, elemIdx);
    elementCoords = COOR(elementNodes, :)';

    % Element-specific dimensions
    numDims = size(elementCoords, 1);
    nodesPerElement = size(elementCoords, 2);
    numGaussPoints = length(weights);

    % Loop over Gauss points to compute local heat flux
    for gaussIdx = 1:numGaussPoints
        % Derivative matrix for current Gauss point
        localDerivatives = shapeFunctionDerivatives(:, :, gaussIdx);

        % Jacobian matrix and determinant
        jacobianMatrix = elementCoords * localDerivatives';
        detJacobian = det(jacobianMatrix);

        % Derivatives in physical space
        physicalDerivatives = inv(jacobianMatrix)' * localDerivatives;

        % Compute heat flux
        localHeatFlux = -localConductivity * physicalDerivatives *
```

```
    localTemperatures;

        % Store in global array
        startIdx = (gaussIdx - 1) * numDims + 1;
        endIdx = gaussIdx * numDims;
        qFluxGlobal(startIdx:endIdx, elemIdx) = localHeatFlux;
    end
end
end
```