MODELLING REACTION-DIFFUSION IN SELF-HEALING MATERIALS USING FEM CODE IN MATLAB

Project Milestone - 4

for the course of

Finite Element Methods (ES 622)

Submitted by:

Md. Riaz Ali Mamud (24250055)
Mayank Awasthi (24250054)
Nikhil Kaser (24310043)

Under the supervision of:

Prof. Sushobhan Sen

Indian Institute of Technology Gandhinagar
Palaj, Gandhinagar - 382055, Gujarat

Acknowledgements

We gratefully acknowledge the guidance and support of Prof. Sushobhan Sen, whose insights and encouragement were instrumental in shaping the direction of this project. His clear and rigorous teaching style, especially the integration of software tools like Gmsh and PrepoMax in the classroom, provided the foundation that enabled us to understand and implement finite element methods at a practical level.

We also thank our teaching assistant for the many interactive sessions and detailed clarifications during the coding and debugging stages.

Our sincere appreciation goes to the Department of Mechanical Engineering at IIT Gandhinagar for providing the computational resources and access to key software tools such as MATLAB. We are also thankful to the IITGN Library for ensuring the availability of essential reference materials, including J. N. Reddy's textbook, which served as a primary resource throughout the project.

Finally, we thank our families for their continuous support and encouragement during the course of this work.

Group Members:

Md Riaz Ali Mamud (Roll No. 24250055)
Mayank Awasthi (Roll No. 24250054)
Nikhil Kaser (Roll No. 24310043)

Abstract

This project employs the finite element method (FEM) to model the reaction-diffusion process in a self-healing material, specifically focusing on self-healing polymers assuming the material behaviour to be isotropic and linearly elastic. The objective is to understand how diffusivity of the healing agent impacts the healing process within a material domain. The significance of this study lies in its potential application in extending the lifespan of materials used in various engineering fields, such as construction, automotive, and aerospace.

In this work, the reaction-diffusion equation, given by $\dot{C} = D\nabla^2 C - kC$, is numerically solved using FEM code implemented in MATLAB. Here, C (mol/m^3) represents the concentration of the healing material, D (m^2/s) is the diffusion coefficient, and $\nabla^2 C$ denotes the Laplacian, capturing the spatial second derivatives of C. The term $-kC$ (mol/m^{3}s), with k (1/s) as the reaction rate constant, introduces a reaction component extending Fick's law, which governs mass conservation in a diffusive system to incorporate local reaction kinetics.

The domain, a 2D geometry, models a material sample with a damage distribution, adjacent to which a healing agent capsule is present. The healing material's properties, including molar mass, density, and diffusivity, are primary inputs for solving the problem. The concentration (C) of the healing agent is related to its volume fraction (ϕ) present in the capsule through the expression $C = \frac{\phi\rho}{M}$, where ρ (kg/m^3) is the density of the healing material, and M (kg/mol) is its molar mass. Boundary conditions are considered, where Dirichlet condition prescribes the concentration of the healing agent at the boundary, which may be a constant or a function of time. In contrast, the Neumann boundary condition specifies the flux across the boundary, which may also be time-dependent. The governing equation is discretized in both space and time using an iso-parametric formulation.

Initial conditions set the concentration of the healing agent, initiating the diffusion process upon damage. The convergence of FEM code's solution is performed through mesh refinement, where the reduction in the error norm of the concentration is monitored. The FEM predictions are validated based on analytical solutions for simplified one-dimensional diffusion problems and by comparison with results from a reliable commercial FEM software. An implicit time stepping scheme, backward Euler method, is employed for discretizing the time derivative. This approach is particularly effective in handling the reaction-diffusion equation with varying diffusivity, as it allows for larger time steps without compromising accuracy.

Multiple case studies are conducted to explore the effects of various parameters on the healing process. The effect of domain length is analysed, showing that shorter domains facilitate quicker healing as the healing agent traverses a shorter path to the crack, while longer domains delay the process. The volume fraction of the healing agent is monitored over time to understand its consumption and effectiveness in filling the crack. Different levels of damage are simulated to assess the healing agent's ability to repair varying sizes of damage zones. The impact of temperature changes on diffusivity and concentration gradients is also analysed. Additionally, spatial variation of diffusivity is considered to assess how non-uniform diffusivity affects the diffusion process.

Hence, this project provides a comprehensive understanding of the reaction-diffusion process in self-healing materials, using FEM in MATLAB to simulate and analyse the

impact of various parameters on healing efficiency. The findings contribute valuable insights to the field of material science and structural engineering, paving the way for the development of more durable and sustainable materials.

Contents

List of Figures

Modeling Reaction-Diffusion in Self-Healing Materials Using FEM

1 Problem Statement

This project focuses on numerically solving the two-dimensional reaction-diffusion equation using the finite element method (FEM) implemented in MATLAB. The primary objective is to study the coupled effects of spatial geometry and temporal evolution on the diffusion and reaction of a chemical agent within a predefined domain.

2 Introduction

Self-healing materials are an advanced class of engineered substances capable of autonomously repairing damage, thereby significantly enhancing the durability, safety, and longevity of products used in industries such as construction, automotive, and aerospace [1]. Inspired by biological systems, these materials can restore functionality after mechanical damage without the need for external intervention, reducing maintenance demands and improving structural reliability.

A fundamental mechanism underlying many self-healing systems—particularly polymer-based ones—is the reaction-diffusion process, which governs the transport of healing agents to damaged regions and the subsequent chemical reactions that restore material integrity. In these materials, healing agents are stored within microcapsules or vascular networks embedded in the host matrix. When damage occurs, the capsules rupture or the network activates, releasing the healing agent into the surrounding region. The agent then diffuses through the material toward the damaged zone and reacts chemically upon arrival, sealing cracks or voids and restoring structural integrity. The effectiveness of this healing response depends on the coupled behavior of the diffusion and the reaction kinetics.

Various modeling approaches for the reaction-diffusion have been developed to simulate and predict the behavior of self-healing materials. Traditional approaches include Continuum-Damage-Healing-Mechanics (CDHM), which focuses on describing the increase and decrease of structural toughness due to damage and healing effects [2]. More advanced modeling techniques include the Theory of Porous Media (TPM), which can consider interactions between constituents such as velocity differences, phase transitions, and temperature exchange [2]. Finite element methods (FEM) offer a robust numerical framework to solve the governing partial differential equations across complex geometries with varying material

properties. It allows researchers to predict the rate and extent of diffusion, evaluate the influence of various physical parameters, and optimize material design. FEM has been applied to model healing agent diffusion in concrete [3] and phase transformations in mineral-based materials [4].

3 Reaction-Diffusion Equation and Derivation

3.1 Reaction-Diffusion Equation

The reaction-diffusion equation is a mathematical model used to describe the transport of chemical species in a medium due to diffusion and reaction processes. The general form of the reaction-diffusion equation in two dimensions is given by:

$$\frac{\partial c}{\partial t} = \nabla \cdot (D \nabla c) + q$$

where:

- $c(x, y, t)$ is the concentration of the species at location (x, y) and time t,
- D is the diffusivity (assumed constant throughout the domain),
- q is the rate of production or consumption per unit volume (can be a function of c, space, or time).

In the special case of a first-order decay or reaction, the source term becomes:

$$q = -kc,$$

where $k > 0$ is the reaction rate constant. Substituting into the general form, the equation becomes:

$$\boxed{\frac{\partial c}{\partial t} = \nabla \cdot (D \nabla c) - kc}$$

3.2 Fick's Law of Diffusion

Fick's first law models diffusion as a flux that is proportional to the negative gradient of concentration:

$$\mathbf{J} = -D^* \nabla c$$

where $\mathbf{J}$ is the diffusion flux vector (amount per unit area per unit time), and D^* is the diffusivity in physical units. This law will be used in the derivation from conservation principles.

3.3 Modeling Assumptions

- The problem is considered in two dimensions, representing a simplified cross-section of a material domain where the reaction-diffusion process occurs.
- The geometry of the crack or defect region is simplified and represented as a rectangle (or any general quadrilateral).
- Parameters such as the diffusion coefficient D and the reaction rate constant k are assumed to be constant with respect to time and space.
- The transport mechanism is governed solely by diffusion; convective or bulk motion is neglected in the model.
- The medium is treated as a continuous domain, justifying the use of partial differential equations (PDEs) to describe the spatial and temporal variation of concentration. Discrete molecular interactions or stochastic effects are not considered.
- The chemical reaction is assumed to be first-order, with the reaction term modeled as $-kc$, implying that the rate of reaction is directly proportional to the local concentration c of the reactant.

3.4 Derivation of Reaction-Diffusion Equation

The reaction-diffusion equation is derived using the law of conservation. The general conservation law is given by:

Rate of change of a property in the domain = Rate of generation inside the domain−Flux out of the bou

Let the conserved property be the concentration $c(x, t)$. The terms are mathematically represented as:

Rate of change of concentration in the domain

$$\frac{\partial}{\partial t} \int_{\Omega} a_0 c \, d\Omega$$

where a_0 is a proportionality constant (e.g., density), c is the concentration, and Ω is the domain.

Rate of generation in the domain

$$\int_{\Omega} q_0 \, d\Omega$$

where q_0 is the volumetric source or sink (e.g., due to chemical reaction).

Rate of flux across the boundary

$$-\int_{\partial\Omega} \mathbf{J} \cdot \hat{n}\, dA$$

where $\mathbf{J}$ is the mass flux vector and $\hat{n}$ is the outward unit normal. The negative sign accounts for outflow from the domain.

Combining all terms:

$$\frac{\partial}{\partial t}\int_{\Omega} a_0 c\, d\Omega = \int_{\Omega} q_0\, d\Omega - \int_{\partial\Omega} \mathbf{J} \cdot \hat{n}\, dA$$

Applying the divergence theorem:

$$\int_{\partial\Omega} \mathbf{J} \cdot \hat{n}\, dA = \int_{\Omega} \nabla \cdot \mathbf{J}\, d\Omega$$

So:

$$\frac{\partial}{\partial t}\int_{\Omega} a_0 c\, d\Omega = \int_{\Omega} (q_0 - \nabla \cdot \mathbf{J})\, d\Omega$$

Since this must hold for any Ω:

$$a_0 \frac{\partial c}{\partial t} = q_0 - \nabla \cdot \mathbf{J}$$

Substituting Fick's law $\mathbf{J} = -D^* \nabla c$:

$$a_0 \frac{\partial c}{\partial t} = \nabla \cdot (D^* \nabla c) + q_0$$

Defining $D = \frac{D^*}{a_0}$ and $q = \frac{q_0}{a_0}$:

$$\frac{\partial c}{\partial t} = \nabla \cdot (D \nabla c) + q$$

For a first-order reaction $q = -kc$:

$$\boxed{\frac{\partial c}{\partial t} = \nabla \cdot (D \nabla c) - kc}$$

4 Weak Formulation for Reaction-Diffusion Equation

- **Governing Equation**
  We consider the two-dimensional reaction-diffusion equation:

$$\frac{\partial c}{\partial t} - \frac{\partial}{\partial x}\left(D\frac{\partial c}{\partial x}\right) - \frac{\partial}{\partial y}\left(D\frac{\partial c}{\partial y}\right) + kc = 0$$

- **Weighted Residual Statement**
  Multiplying the equation by a weight function w and integrating over the element domain Ω_e:

$$\int_{\Omega_e} w \left(\frac{\partial c}{\partial t} - \frac{\partial}{\partial x}\left(D \frac{\partial c}{\partial x} \right) - \frac{\partial}{\partial y}\left(D \frac{\partial c}{\partial y} \right) + kc \right) dx\, dy = 0$$

- **Expanded Form**

$$\int_{\Omega_e} w \frac{\partial c}{\partial t}\, dx\, dy - \int_{\Omega_e} w \frac{\partial}{\partial x}\left(D \frac{\partial c}{\partial x} \right) dx\, dy - \int_{\Omega_e} w \frac{\partial}{\partial y}\left(D \frac{\partial c}{\partial y} \right) dx\, dy + \int_{\Omega_e} wkc\, dx\, dy = 0$$

- **Applying Integration by Parts (Green's Theorem)**

$$-\int_{\Omega_e} w \frac{\partial G}{\partial x}\, dx\, dy = \int_{\Omega_e} G \frac{\partial w}{\partial x}\, dx\, dy - \oint_{\partial \Omega_e} wGn_x\, dl, \quad G = D\frac{\partial c}{\partial x}$$

$$-\int_{\Omega_e} w \frac{\partial G}{\partial y}\, dx\, dy = \int_{\Omega_e} G \frac{\partial w}{\partial y}\, dx\, dy - \oint_{\partial \Omega_e} wGn_y\, dl, \quad G = D\frac{\partial c}{\partial y}$$

$$\int_{\Omega_e} w \frac{\partial c}{\partial t}\, dx\, dy + \int_{\Omega_e} D \frac{\partial c}{\partial x}\frac{\partial w}{\partial x}\, dx\, dy + \int_{\Omega_e} D \frac{\partial}{\partial y}\frac{\partial w}{\partial y}\, dx\, dy - \oint_{\partial \Omega_e} wD\frac{\partial c}{\partial x} n_x\, dl - \oint_{\partial \Omega_e} wD\frac{\partial c}{\partial y} n_y\, dl + \int_{\Omega_e} wkc\, dx\, dy = 0$$

- **Compact Matrix Form**

$$A = \begin{bmatrix} D & 0 & 0 \\ 0 & D & 0 \\ 0 & 0 & k \end{bmatrix}, \quad \mathcal{D} = \begin{bmatrix} \frac{\partial}{\partial x} \\ \frac{\partial}{\partial y} \\ 1 \end{bmatrix}$$

$$\int_{\Omega_e} w \frac{\partial c}{\partial t}\, dx\, dy + \int_{\Omega_e} (\mathcal{D}w)^T A \mathcal{D} c\, dx\, dy = \oint_{\partial \Omega_e} wD\frac{\partial c}{\partial x} n_x\, dl + \oint_{\partial \Omega_e} wD\frac{\partial c}{\partial y} n_y\, dl$$

- **Finite Element Discretization**
  Let:

$$c = N^T c, \quad w = N^T w, \quad B = \mathcal{D} N^T$$

  Then:

$$w^T \int_{\Omega_e} NN^T\, dx\, dy\, \frac{dc}{dt} + w^T \int_{\Omega_e} B^T AB\, dx\, dy\, c = w^T \oint_{\partial \Omega_e} Nqn\, dl$$

- **Element Matrices**

$$C^e \dot{c}^e + K^e c^e = Q^e$$

$$C^e = \int_{\Omega_e} NN^T\, dx\, dy, \quad K^e = \int_{\Omega_e} B^T AB\, dx\, dy, \quad Q^e = \oint_{\partial \Omega_e} Nqn\, dl$$

5 Isoparametric Formulation for Bilinear Quad Element

Before assembling the fully discrete system, we transform the element stiffness matrix K^e, capacity matrix C^e, and boundary load vector Q^e from the physical (x, y) domain into the natural (t, s) coordinates on the reference square $[-1, 1] \times [-1, 1]$. This enables efficient Gauss-quadrature integration in MATLAB.

$$x = \sum_{i=1}^{4} \bar{N}_i(t, s)\, x_i, \quad y = \sum_{i=1}^{4} \bar{N}_i(t, s)\, y_i$$

The shape functions $\bar{N}_i(t, s)$ are created using tensor products of 1D Lagrange polynomials:

$$\bar{N}_i(t, s) = \begin{pmatrix} l_0^1(t) \\ l_1^1(t) \end{pmatrix} \otimes \begin{pmatrix} l_0^1(s) \\ l_1^1(s) \end{pmatrix} \quad \text{where:}$$

$$l_0^1(t) = \frac{1-t}{2},\ l_1^1(t) = \frac{1+t}{2},\ l_0^1(s) = \frac{1-s}{2},\ l_1^1(s) = \frac{1+s}{2}$$

These define the bilinear basis over the square reference element. The resulting shape functions are:

$$\bar{N}(t, s) = \begin{pmatrix} \frac{1}{4}(1-s)(1-t) \\ \frac{1}{4}(1-s)(1+t) \\ \frac{1}{4}(1+s)(1+t) \\ \frac{1}{4}(1+s)(1-t) \end{pmatrix}$$

These correspond to the four corners of the reference square: bottom-left, bottom-right, top-right, and top-left.

To perform analysis, we need to compute derivatives of shape functions with respect to physical coordinates (x, y). However, since our shape functions are defined in the (t, s) space, we first compute:

$$\frac{\partial \bar{N}}{\partial t} = \begin{pmatrix} -\frac{1-s}{4} \\ \frac{1-s}{4} \\ \frac{1+s}{4} \\ -\frac{1+s}{4} \end{pmatrix}, \quad \frac{\partial \bar{N}}{\partial s} = \begin{pmatrix} -\frac{1-t}{4} \\ -\frac{1+t}{4} \\ \frac{1+t}{4} \\ \frac{1-t}{4} \end{pmatrix}$$

Next, we use the Jacobian matrix J to convert derivatives from (t, s) space to physical coordinates (x, y). This transformation is given by:

$$J = \begin{pmatrix} \frac{\partial x}{\partial t} & \frac{\partial x}{\partial s} \\ \frac{\partial y}{\partial t} & \frac{\partial y}{\partial s} \end{pmatrix} \Rightarrow J^{-1} = \begin{pmatrix} \frac{\partial t}{\partial x} & \frac{\partial s}{\partial x} \\ \frac{\partial t}{\partial y} & \frac{\partial s}{\partial y} \end{pmatrix}$$

Using the chain rule, we compute:

$$\frac{\partial \bar{N}_i}{\partial x} = \frac{\partial \bar{N}_i}{\partial t}\frac{\partial t}{\partial x} + \frac{\partial \bar{N}_i}{\partial s}\frac{\partial s}{\partial x} \quad \text{and similarly for } \frac{\partial \bar{N}_i}{\partial y}$$

This can be written compactly using matrix multiplication:

$$\begin{pmatrix} \frac{\partial t}{\partial x} & \frac{\partial s}{\partial x} & 0 \\ \frac{\partial t}{\partial y} & \frac{\partial s}{\partial y} & 0 \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} \frac{\partial \bar{N}_i}{\partial t} \\ \frac{\partial \bar{N}_i}{\partial s} \\ \bar{N}_i \end{pmatrix} = \begin{pmatrix} J^{-1} & 0 \\ 0 & 1 \end{pmatrix} \begin{pmatrix} \frac{\partial \bar{N}_i}{\partial t} \\ \frac{\partial \bar{N}_i}{\partial s} \\ \bar{N}_i \end{pmatrix}$$

This transformed vector gives the entries of the strain-displacement matrix B, which is used to compute the element stiffness matrix K^e:

$$K^e = \int_{\Omega_e} B^T A B \, dx \, dy$$

We map this integral from the physical domain to the reference square, incorporating the Jacobian determinant:

$$K^e = \int_{-1}^{1}\int_{-1}^{1} \bar{B}^T \begin{pmatrix} J^{-1} & 0 \\ 0 & 1 \end{pmatrix}^T A \begin{pmatrix} J^{-1} & 0 \\ 0 & 1 \end{pmatrix} \bar{B} \, |J| \, dt \, ds$$

Damping (or Mass) Matrix:

$$C^e = \int_{\Omega_e} N N^T \, dx \, dy = \int_{-1}^{1}\int_{-1}^{1} \bar{N}(t,s) \, \bar{N}^T(t,s) \, |J(t,s)| \, dt \, ds$$

Boundary Load Vector Q^e:

When applying natural boundary conditions, we encounter integrals along the element edges. The load contribution due to a diffusive flux along the boundary is given by:

$$Q^e = \oint_{\partial\Omega_e} N \left(n_x D \frac{\partial c}{\partial x} + n_y D \frac{\partial c}{\partial y} \right) dl$$

To evaluate this, we express the differential arc length dl using the reference coordinates:

$$\vec{r}(t,s) = (x(t,s), y(t,s)) \Rightarrow dx = \frac{\partial x}{\partial t}dt + \frac{\partial x}{\partial s}ds, \; dy = \frac{\partial y}{\partial t}dt + \frac{\partial y}{\partial s}ds$$

$$dl = \sqrt{dx^2 + dy^2} = \sqrt{(J_{11}dt + J_{21}ds)^2 + (J_{12}dt + J_{22}ds)^2}$$

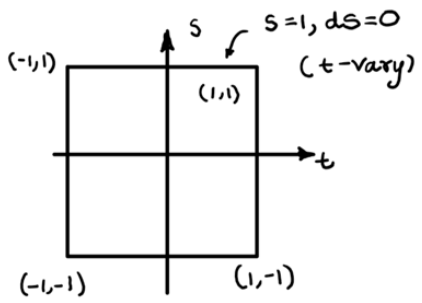


Figure 1: Square Reference Element (In natural coordinate)

For instance, an edge where one coordinate is fixed (e.g., $s = \text{const} \Rightarrow ds = 0$):

$$dl = \sqrt{J_{11}^2 + J_{12}^2}\, dt$$

So the boundary integral simplifies to:

$$Q^e = \int_{-1}^{1} N(t) \left(n_x D \frac{\partial c}{\partial x} + n_y D \frac{\partial c}{\partial y} \right) \sqrt{J_{11}^2 + J_{12}^2}\, dt$$

5.1 Implementation on Code:

```
line 153:  Ce = Ce + (N * N') * (det(J)*w);
```

$line154 : Ke = Ke + (Bbar' * [invJ, [0;0]; 0, 0, 1]' * Abr * [invJ, [0;0]; 0, 0, 1] * Bbar) * (det(J) * w);$

```
line 229:  integrand = Nedge * (normal(1)*D_val*flux_x + normal(2)*D_val*flux_y);
line 230:  Qe = Qe + integrand * scaling * gauss_wts_1(ip);
```

- **Damping Matrix (`Ce`):**
  - `N`: Shape function matrix at Gauss point.
  - `N'`: Transpose of shape function matrix.
  - `det(J)`: Jacobian determinant, maps reference element to physical domain.
  - `w`: Gauss quadrature weight for area integration.
- **Stiffness Matrix (`Ke`):**
  - `Bbar`: Matrix in reference domain.
  - `invJ`: Inverse of Jacobian, transforms to physical coordinates.
  - `Abr`: Material (constitutive) matrix.
  - `det(J)`: Jacobian determinant for area scaling.
  - `w`: Gauss quadrature weight.

– **Boundary Flux Vector (`Qe`):**
  * `Nedge`: Shape functions along the boundary edge.
  * `normal`: Components of the outward normal vector.
  * `D_val`: Diffusivity or conductivity coefficient.
  * `flux_x`, `flux_y`: Applied flux values in x and y directions.
  * `scaling`: Length scaling for the edge (e.g., 1D Jacobian).
  * `gauss_wts_1(ip)`: 1D Gauss weight for boundary integration.

5.2 Handling of Normals in the Boundary Integral

In the boundary-flux assembly, each Neumann edge must be equipped with its outward unit normal. Let an edge span nodes at $\mathbf{x}_1 = (x_1, y_1)$ and $\mathbf{x}_2 = (x_2, y_2)$, and let $\mathbf{c} = (x_c, y_c)$ be the centroid of the adjacent element. We proceed as follows:

1. Form two perpendicular candidate vectors
$$\mathbf{cand}_1 = \begin{bmatrix} y_2 - y_1 \\ -(x_2 - x_1) \end{bmatrix}, \qquad \mathbf{cand}_2 = -\,\mathbf{cand}_1.$$
2. Compute the edge midpoint $\mathbf{m} = \frac{1}{2}(\mathbf{x}_1 + \mathbf{x}_2)$ and the vector from midpoint to centroid $\mathbf{v} = \mathbf{c} - \mathbf{m}$.
3. Select the candidate that points outward (i.e. forms an obtuse angle with $\mathbf{v}$):
$$\mathbf{n}_{\text{raw}} = \begin{cases} \mathbf{cand}_1, & \text{if } \mathbf{cand}_1 \cdot \mathbf{v} < 0, \\ \mathbf{cand}_2, & \text{otherwise,} \end{cases} \qquad \mathbf{n} = \frac{\mathbf{n}\text{raw}}{\|\mathbf{n}\text{raw}\|}.$$

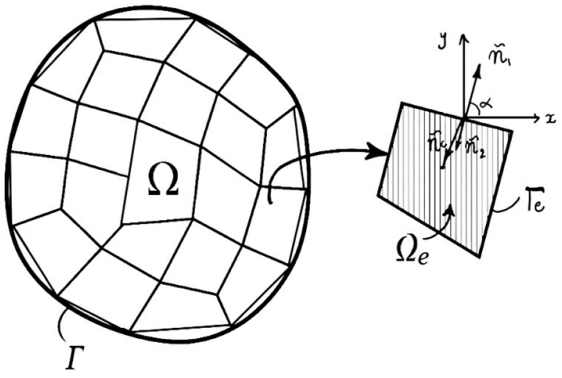


Figure 2: Discretization of an Irregular Domain by Quadrilateral Element (with unit normals n1 and n2 on boundary Γ)

This simple centroid-test guarantees that $\mathbf{n}$ indeed points away from the element interior. In our MATLAB code the above steps are implemented exactly as follows:

```
cand1 = [y2 - y1,   -(x2 - x1)];
cand2 =      - cand1;
mid   = 0.5 * [x1 + x2, y1 + y2];
if dot(cand1, [xc, yc] - mid) < 0
    nvec = cand1;
else
    nvec = cand2;
end
n_info(k,:) = nvec / norm(nvec);
```

Here `[xc,yc]` is the element centroid and `n_info` stores the resulting outward unit normal for each boundary edge.

5.3 Finite Element Implementation for Example 8.5.1 [5]

In this section we describe our MATLAB implementation for obtaining the global stiffness matrix $\mathbf{K}^{\text{global}}$, capacity matrix $\mathbf{C}^{\text{global}}$, load vector $\mathbf{Q}^{\text{global}}$, and the nodal concentration solution for Example 8.5.1 in J. N. Reddy's textbook. The governing equation,

$$-\nabla\cdot\big(k\,\nabla C\big) \;=\; 0,$$

is posed on a rectangular domain of size 3×2. The boundary conditions are:

- **Neumann (insulated):** Left and bottom edges, $k\,\nabla C \cdot \mathbf{n} = 0$.
- **Dirichlet (zero):** Right edge, $C = 0$.
- **Dirichlet (prescribed):** Top edge, $C(x, 2) = \cos\!\big(\frac{\pi x}{6}\big)$.

The domain is discretized into linear quadrilateral elements, and for each element (K^e) ,(C^e) and (Q^e) are computed using Gaussian quadrature. And, for the steady-state case the system is solved directly using MATLAB's backslash operator.

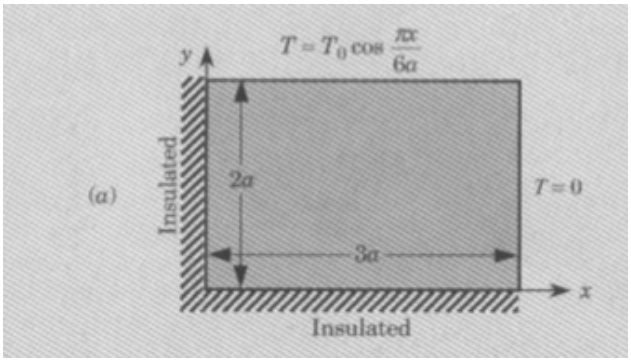


Figure 3: Sample Problem 8.5.1 From J.N. Reddy [5]

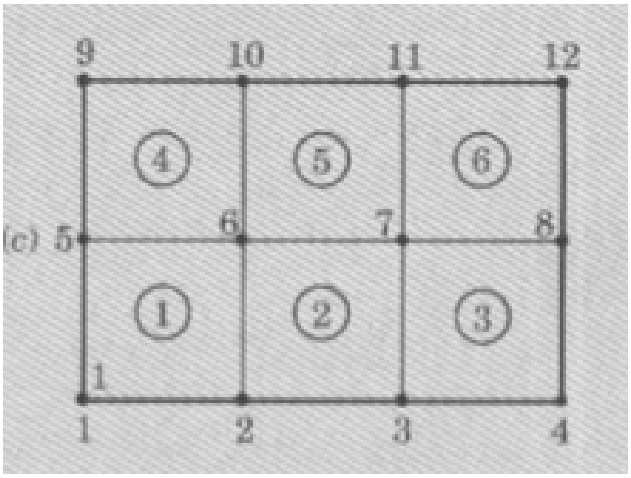


Figure 4: Sample Problem 8.5.1 From J.N. Reddy [5]

```
----- Full Global Matrices -----
K_global (full):
   10.0000   -2.5000         0         0   -2.5000   -5.0000         0         0         0         0         0         0
   -2.5000   20.0000   -2.5000         0   -5.0000   -5.0000   -5.0000         0         0         0         0         0
         0   -2.5000   20.0000   -2.5000         0   -5.0000   -5.0000   -5.0000         0         0         0         0
         0         0   -2.5000   10.0000         0         0   -5.0000   -2.5000         0         0         0         0
   -2.5000   -5.0000         0         0   20.0000   -5.0000         0         0   -2.5000   -5.0000         0         0
   -5.0000   -5.0000   -5.0000         0   -5.0000   40.0000   -5.0000         0   -5.0000   -5.0000   -5.0000         0
         0   -5.0000   -5.0000   -5.0000         0   -5.0000   40.0000   -5.0000         0   -5.0000   -5.0000   -5.0000
         0         0   -5.0000   -2.5000         0         0   -5.0000   20.0000         0         0   -5.0000   -2.5000
         0         0         0         0   -2.5000   -5.0000         0         0   10.0000   -2.5000         0         0
         0         0         0         0   -5.0000   -5.0000   -5.0000         0   -2.5000   20.0000   -2.5000         0
         0         0         0         0         0   -5.0000   -5.0000   -5.0000         0   -2.5000   20.0000   -2.5000
         0         0         0         0         0         0   -5.0000   -2.5000         0         0   -2.5000   10.0000
```

Figure 5: Global Stiffness matrix (using MATLAB)

```
C_global (full):
    0.1111    0.0556         0         0    0.0556    0.0278         0         0         0         0         0         0
    0.0556    0.2222    0.0556         0    0.0278    0.1111    0.0278         0         0         0         0         0
         0    0.0556    0.2222    0.0556         0    0.0278    0.1111    0.0278         0         0         0         0
         0         0    0.0556    0.1111         0         0    0.0278    0.0556         0         0         0         0
    0.0556    0.0278         0         0    0.2222    0.1111         0         0    0.0556    0.0278         0         0
    0.0278    0.1111    0.0278         0    0.1111    0.4444    0.1111         0    0.0278    0.1111    0.0278         0
         0    0.0278    0.1111    0.0278         0    0.1111    0.4444    0.1111         0    0.0278    0.1111    0.0278
         0         0    0.0278    0.0556         0         0    0.1111    0.2222         0         0    0.0278    0.0556
         0         0         0         0    0.0556    0.0278         0         0    0.1111    0.0556         0         0
         0         0         0         0    0.0278    0.1111    0.0278         0    0.0556    0.2222    0.0556         0
         0         0         0         0         0    0.0278    0.1111    0.0278         0    0.0556    0.2222    0.0556
         0         0         0         0         0         0    0.0278    0.0556         0         0    0.0556    0.1111

q_global (full):
   375   300   300   150   450     0     0     0   225     0     0     0
```

Figure 6: Global Damping matrix (using MATLAB)

Node:	1	2	3	4	5	6	7	8	9	10	11	12
Concentration:	0.6127	0.5308	0.3064	0.0000	0.7031	0.6088	0.3515	0.0000	1.0000	0.8660	0.5000	0.0000

Figure 7: Concentrations (using MATLAB)

6 Fully Discrete Formulation using the θ-Method

Assuming $\mathbf{C}_g$, $\mathbf{K}_g$, and $\mathbf{Q}_g$ are time-independent, we begin with the semi-discrete system:

$$\mathbf{C}_g\,\dot{\mathbf{c}} + \mathbf{K}_g\,\mathbf{c} = \mathbf{Q}_g$$

From this, we isolate the time derivative:

$$\dot{\mathbf{c}}^s = \frac{\mathbf{Q}_g - \mathbf{K}_g\mathbf{c}^s}{\mathbf{C}_g}$$

Using the θ-method for time integration:

$$\mathbf{c}^{s+1} = \mathbf{c}^s + \Delta t\left[(1-\theta)\dot{\mathbf{c}}^s + \theta\dot{\mathbf{c}}^{s+1}\right]$$

Substituting $\dot{\mathbf{c}}^s$ and $\dot{\mathbf{c}}^{s+1}$:

$$\mathbf{c}^{s+1} = \mathbf{c}^s + \Delta t\left((1-\theta)\frac{\mathbf{Q}_g - \mathbf{K}_g\mathbf{c}^s}{\mathbf{C}_g} + \theta\frac{\mathbf{Q}_g - \mathbf{K}_g\mathbf{c}^{s+1}}{\mathbf{C}_g}\right)$$

Multiplying both sides by $\mathbf{C}_g$:

$$\mathbf{C}_g\mathbf{c}^{s+1} + \theta\Delta t\,\mathbf{K}_g\mathbf{c}^{s+1} = \mathbf{C}_g\mathbf{c}^s + \Delta t\left[(1-\theta)(\mathbf{Q}_g - \mathbf{K}_g\mathbf{c}^s) + \theta\mathbf{Q}_g\right]$$

Rewriting and factoring:

$$(\mathbf{C}_g + \theta\Delta t\,\mathbf{K}_g)\,\mathbf{c}^{s+1} = (\mathbf{C}_g - (1-\theta)\Delta t\,\mathbf{K}_g)\,\mathbf{c}^s + \Delta t\left[(1-\theta)\mathbf{Q}_g + \theta\mathbf{Q}_g\right]$$

Final Fully Discrete Equation:

$$\boxed{(\mathbf{C}_g + \theta\Delta t\,\mathbf{K}_g)\,\mathbf{c}^{s+1} = (\mathbf{C}_g - (1-\theta)\Delta t\,\mathbf{K}_g)\,\mathbf{c}^s + \Delta t\left[(1-\theta)\mathbf{Q}_g + \theta\mathbf{Q}_g\right]}$$

7 Stability Analysis of the θ-Method

The general formulation for the θ-family of time integration schemes is:

$$(C + \theta\Delta t K)\,\mathbf{c}^{s+1} = (C - (1-\theta)\Delta t K)\,\mathbf{c}^s + \Delta t\left[(1-\theta)\mathbf{Q} + \theta\mathbf{Q}\right].$$

Here, C is the global capacity matrix, K is the global stiffness matrix, and $\mathbf{u}$ is the solution vector. The parameter θ dictates the method:

- $\theta = 0$: Forward Euler method,
- $\theta = 0.5$: Crank-Nicolson method,
- $\theta = 1$: Backward Euler method.

The stability of this scheme is determined by the **amplification factor** A, which is derived from the time-stepping equation. The operator A governs how errors propagate in time:

$$\mathbf{c}^{s+1} = A(\mathbf{c}^s) + F$$

For the method to be stable, the spectral radius of A must satisfy:

$$\rho(A) = \max_i |\lambda_i(A)| \leq 1,$$

where $\lambda_i(A)$ are the eigenvalues of A.

7.1 Eigenvalue Analysis

Consider the matrix $L = -C^{-1}K$. The eigenvalues $\lambda_i(L)$ of L determine the stability properties. We define the non-dimensional time-stepping parameter:

$$z = \lambda \Delta t,$$

where λ is an eigenvalue of L. The method is stable if $|z| \leq 1$, meaning that the solution does not grow exponentially over time.

7.2 Stability Regions for Different θ Values

The stability condition depends on the choice of θ:

- For $\theta = 0$ (Forward Euler), the method is **conditionally stable**, and the time step Δt must satisfy:

  $$\Delta t < \frac{2}{-\lambda_{\min}(1 - 2\theta)} \quad \Longrightarrow \quad \Delta t < \frac{2}{-\lambda_{\min}},$$

  where $\lambda_{\min}$ is the smallest eigenvalue of L.
- For $\theta = 0.5$ (Crank-Nicolson), the method is **unconditionally stable** for linear problems. All values of z lie within the stability region, meaning any time step Δt will keep the solution stable.
- For $\theta = 1$ (Backward Euler), the method is also **unconditionally stable**, but it dampens high-frequency modes more effectively than Crank–Nicolson.

7.3 Stable Time Step

The maximum stable time step is given by:

$$\Delta t_{\max} = \begin{cases} \frac{2}{-\lambda_{\min}(1-2\theta)}, & \text{if } \theta < 0.5, \\ \infty, & \text{if } \theta \geq 0.5. \end{cases}$$

Thus, the stable range for Δt is:

$$\Delta t \in [0, \Delta t_{\max}].$$

For $\theta \geq 0.5$, the method is unconditionally stable, meaning the time step can be arbitrarily large without causing instability.

8 Code Outline

This section provides a structured overview of the various code sections, functions, statements, and how they interact along with their explanation.

8.1 Opening and Validating the Input File

```
% Open input file
fid = fopen('J.N.Reddy_Example_8.5.1.inp','r');
if fid == -1
    error('Could not open the inp file');
end
```

Explanation: The ABAQUS input file i.e `J.N.Reddy Example 8.5.1 .inp` is opened in read-only mode. If `fopen` fails (returns –1), we call `error('Could not open the inp file')` to halt execution with a clear message, avoiding downstream errors.

8.2 Skipping the Header

```
% Skip header (9 lines)
for i = 1:9
    fgetl(fid);
end
```

Explanation: Generally in an ABAQUS .inp file, lines 1–9 contain only comments/metadata. We discard them via nine calls to `fgetl`, each reading—and ignoring—one line.

8.3 Reading Nodal Coordinates

```
% Read node data
tot_dofs = 12;
node_id = zeros(1, tot_dofs);
x_glbl  = zeros(1, tot_dofs);
y_glbl  = zeros(1, tot_dofs);
for i = 1:tot_dofs
    tmp = sscanf(fgetl(fid), '%d,%e,%e', 3);
    node_id(i) = tmp(1);
    x_glbl(i)  = tmp(2);
    y_glbl(i)  = tmp(3);
end
```

Explanation: We declare `tot_dofs=12` as per the example 8.5.1 geometry and pre-allocate arrays. Each line of the form $\langle$nodeID$\rangle, \langle x\rangle, \langle y\rangle$ is parsed with `sscanf` into one integer and two floats, stored in `node_id`, `x_glbl`, `y_glbl`.

8.4 Reading Element Connectivity

```
fgetl(fid);  % skip blank line
tot_elem = 6;
ndes_per_elmn = 4;
elem_connect = zeros(tot_elem, ndes_per_elmn);
for e = 1:tot_elem
    tmp1 = sscanf(fgetl(fid), '%d,%d,%d,%d,%d', 5);
    elem_connect(e,:) = tmp1(2:5);
end
```

Explanation: After a blank line we read all the quad elements nodal connectivity information, hard-coded alongside with the number of nodes per element. Each line $\langle$elemID$\rangle, n_1, n_2, n_3, n_4$ is parsed into a 5-entry vector; we discard the first and store $[n_1\, n_2\, n_3\, n_4]$.

8.5 Boundary Condition Data Structures

```
tot_surfaces = 4;
bc_types = {'N','D','D','N'};
flux_values = {{'0','-30'},{'0','0'},{'0','0'},{'-20','0'}};
dirichlet_values = {'0','0','cos(pi*x/6)','0'};
conc_old = zeros(tot_dofs,1);
```

Explanation: We predefine the number of surfaces with BC types: 'N' for Neumann, 'D' for Dirichlet. Flux and Dirichlet expressions are stored as strings; `conc_old` is currently taken zero for the example 8.5.1 and can be altered as per our requirements, for later Dirichlet enforcement.

8.6 Parsing and Storing Boundary Edges

```
Boundary_info = cell(1, tot_surfaces);
Normals_info  = cell(1, tot_surfaces);
for s = 1:tot_surfaces
    clean = regexprep(fgetl(fid),'[,]+',' ');
    vals  = sscanf(clean,'%d');
    if numel(vals)==3
        surf_nodes = vals(1):vals(3):vals(2);
    else
        surf_nodes = vals(:)';
    end
    fgetl(fid);
    clean = regexprep(fgetl(fid),'[,]+',' ');
    vals  = sscanf(clean,'%d');
    if numel(vals)==3
        surf_elems = vals(1):vals(3):vals(2);
    else
        surf_elems = vals(:)';
    end
    fgetl(fid);
    maxEdges = numel(surf_elems)*4;
    b_info = zeros(maxEdges,3);
    cnt = 0;
    for e = surf_elems
        con = elem_connect(e,:);
        edges = [con([1 2]); con([2 3]); con([3 4]); con([4 1])];
        for j = 1:4
            if all(ismember(edges(j,:), surf_nodes))
                cnt = cnt+1;
                b_info(cnt,:) = [e, edges(j,:)];
            end
        end
    end
    Boundary_info{s} = b_info(1:cnt,:);
```

Explanation: We read node and element lists (as ranges or explicit), detect which element edges lie on the boundary, and record them as [elemID, nodeA, nodeB].

8.7 Computing Outward Unit Normals

```
        n_info = zeros(size(Boundary_info{s},1),2);
        for k=1:size(Boundary_info{s},1)
            e  = Boundary_info{s}(k,1);
            n1 = Boundary_info{s}(k,2);
            n2 = Boundary_info{s}(k,3);
            x1 = x_glbl(n1); y1 = y_glbl(n1);
            x2 = x_glbl(n2); y2 = y_glbl(n2);
            cand1 = [y2-y1, -(x2-x1)];
            cand2 = -cand1;
            xc = mean(x_glbl(elem_connect(e,:)));
            yc = mean(y_glbl(elem_connect(e,:)));
            mid = 0.5*[x1+x2, y1+y2];
            if dot(cand1,[xc,yc]-mid)<0
                nvec = cand1;
            else
                nvec = cand2;
            end
            n_info(k,:) = nvec / norm(nvec);
        end
        Normals_info{s} = n_info;
    end
    fclose(fid);
```

Explanation: Two perpendicular vectors to an edge $(x_1, y_1) \to (x_2, y_2)$ are $\pm[(y_2 - y_1), -(x_2 - x_1)]$. We choose the one pointing outward (via centroid test) and normalize.

8.8 Gaussian Quadrature & Preallocation

```
NGPS = 4;
tgp = [-1/sqrt(3),1/sqrt(3),1/sqrt(3),-1/sqrt(3)];
sgp = [-1/sqrt(3),-1/sqrt(3),1/sqrt(3),1/sqrt(3)];
wts = [1,1,1,1];
total_entries = tot_elem*ndes_per_elmn^2;
rows_K = zeros(total_entries,1);
cols_K = zeros(total_entries,1);
vals_K = zeros(total_entries,1);
rows_C = zeros(total_entries,1);
cols_C = zeros(total_entries,1);
vals_C = zeros(total_entries,1);
entry = 1;
q_global = zeros(tot_dofs,1);
```

Explanation: We set up a 2×2 Gauss rule on $[-1,1]^2$ and preallocate triplet arrays for sparse assembly plus the global flux vector.

8.9 Element-Level FEM Assembly

```
for elem=1:tot_elem
    con = elem_connect(elem,:);
    xnod = x_glbl(con)'; ynod = y_glbl(con)';
    Ke = zeros(4); Ce = zeros(4);
    D_vec = 15*ones(4,1); K_vec = zeros(4,1);
    for gp=1:NGPS
        s = sgp(gp); t = tgp(gp); w = wts(gp);
        N    = 1/4*[(1-s)*(1-t);(1-s)*(1+t);(1+s)*(1+t);(1+s)*(1-t)];
        dNdt = 1/4*[-(1-s);(1-s);(1+s);-(1+s)];
        dNds = 1/4*[-(1-t);-(1+t);(1+t);(1-t)];
        J    = [dNdt'*xnod, dNdt'*ynod; dNds'*xnod, dNds'*ynod];
        detJ = det(J); invJ = inv(J);
        D = N'*D_vec; K = N'*K_vec;
        Bbar = [dNdt';dNds';N'];
        Abr  = diag([D,D,K]);
        Ke = Ke + (Bbar'*[invJ,zeros(2,1);zeros(1,2),1]'*Abr*...
             [invJ,zeros(2,1);zeros(1,2),1]*Bbar)*detJ*w;
        Ce = Ce + (N*N')*detJ*w;
    end
    for i=1:4, for j=1:4
        rows_K(entry)=con(i); cols_K(entry)=con(j);
        vals_K(entry)=Ke(i,j);
        rows_C(entry)=con(i); cols_C(entry)=con(j);
        vals_C(entry)=Ce(i,j);
        entry = entry+1;
    end,end
end
```

Explanation: For each element, we compute local stiffness $\mathbf{K}^e$ and local damping $\mathbf{C}^e$ matrices via Gaussian quadrature, then scatter their entries into global triplet arrays.

8.10 Global Assembly & Stability Analysis

```
K_global = sparse(rows_K,cols_K,vals_K,tot_dofs,tot_dofs);
C_global = sparse(rows_C,cols_C,vals_C,tot_dofs,tot_dofs);
L_full = full(-C_global\K_global);
lambda_vals = eig(L_full);
lambda_min = min(real(lambda_vals));
```

```
if theta<0.5
    dt_max = 2/(-lambda_min*(1-2*theta));
else
    dt_max = Inf;
end
stable_deltat_range = [0,dt_max];
assignin('base','stable_deltat_range',stable_deltat_range);
```

Explanation: We form $L = -C^{-1}K$ to find $\lambda_{\min}$. For the -method the maximum stable Δt is

$$\Delta t_{\max} = \begin{cases} \frac{2}{-\lambda_{\min}(1-2\theta)}, & \theta < 0.5, \\ \infty, & \theta \geq 0.5. \end{cases}$$

8.11 Neumann Boundary Flux Integration

```
ngps_line = 2;
gauss_pts_l = [-1/sqrt(3),1/sqrt(3)];
gauss_wts_l = [1,1];
for s_idx=1:tot_surfaces
  if strcmp(bc_types{s_idx},'N')
    flux_ex = flux_values{s_idx};
    b_info  = Boundary_info{s_idx};
    n_info  = Normals_info{s_idx};
    for e_idx=1:size(b_info,1)
        elem = b_info(e_idx,1);
        nodeA = b_info(e_idx,2);
        nodeB = b_info(e_idx,3);
        normal = n_info(e_idx,:);
        con = elem_connect(elem,:);
        xnod=x_glbl(con)'; ynod=y_glbl(con)';
        posA=find(con==nodeA); posB=find(con==nodeB);
        sp=sort([posA,posB]);
        if isequal(sp,[1 2]),f=-1;v='t';
        elseif isequal(sp,[2 3]),f=1;v='s';
        elseif isequal(sp,[3 4]),f=1;v='t';
        else f=-1;v='s'; end
        Qe=zeros(4,1);
        for ip=1:ngps_line
          xi=gauss_pts_l(ip);
          if v=='t',t_pt=xi;s_pt=f; else s_pt=xi;t_pt=f; end
          Nedge=zeros(4,1);
          ti=[-1;1;1;-1]; si=[-1;-1;1;1];
          for kN=1:4
            Nedge(kN)=1/4*(1+ti(kN)*t_pt)*(1+si(kN)*s_pt);
```

```
            end
            dNdt=1/4*[-(1-s_pt);(1-s_pt);(1+s_pt);-(1+s_pt)];
            dNds=1/4*[-(1-t_pt);-(1+t_pt);(1+t_pt);(1-t_pt)];
            dxdt=dNdt'*xnod; dydt=dNdt'*ynod;
            dxds=dNds'*xnod; dyds=dNds'*ynod;
            scale=(v=='t')*norm([dxdt,dydt])+(v=='s')*norm([dxds,dyds]);
            xm=Nedge'*xnod; ym=Nedge'*ynod; x=xm; y=ym;
            fx=eval(flux_ex{1}); fy=eval(flux_ex{2});
            integrand=Nedge*(normal(1)*15*fx+normal(2)*15*fy);
            Qe=Qe+integrand*scale*gauss_wts_1(ip);
          end
          for ii=1:4
            q_global(con(ii))=q_global(con(ii))+Qe(ii);
          end
      end
    end
  end
```

Explanation: Neumann BCs enter via boundary integrals $Q^e = \int_{\partial\Omega^e} N\,(n \cdot D\,\phi)\,ds$. We parametrize each edge, evaluate 1D shape functions at Gauss points, map to physical space, compute the Jacobian length, evaluate prescribed flux, form the integrand, and assemble into `q_global`.

8.12 Dirichlet Boundary Enforcement

```
for s=1:tot_surfaces
  if strcmp(bc_types{s},'D')
    expr=dirichlet_values{s};
    nodes=unique(Boundary_info{s}(:,2:3));
    for nn=nodes'
      x=x_glbl(nn); y=y_glbl(nn);
      conc_old(nn)=eval(expr);
    end
  end
end
```

Explanation: We evaluate the Dirichlet function $g(x, y)$ at boundary nodes and initialize `conc_old` so those DOFs remain fixed at every time step.

8.13 Time-Stepping via the -Method

```
K1=C_global+theta*deltat*K_global;
```

```
K2=C_global-(1-theta)*deltat*K_global;
Q1=deltat*((1-theta)*q_global+theta*q_global);
nt=ceil(tn/deltat);
conc_history=zeros(tot_dofs,nt+1);
conc_history(:,1)=conc_old;
for step=1:nt
  conc_new=zeros(tot_dofs,1);
  known=false(tot_dofs,1);
  for s=1:tot_surfaces
    if strcmp(bc_types{s},'D')
      expr=dirichlet_values{s};
      nodes=unique(Boundary_info{s}(:,2:3));
      for nn=nodes'
        x=x_glbl(nn); y=y_glbl(nn);
        conc_new(nn)=eval(expr);
        known(nn)=true;
      end
    end
  end
  rhs=K2*conc_old+Q1; rhs(known)=conc_new(known);
  unk=~known;
  conc_new(unk)=K1(unk,unk)\(rhs(unk)-K1(unk,known)*conc_new(known));
  conc_history(:,step+1)=conc_new; conc_old=conc_new;
end
```

Explanation: We solve $\mathbf{C}\dot{u} + \mathbf{K}u = \mathbf{q}$ with the -method:

$$(C + \theta\Delta t K)u^{n+1} = (C - (1-\theta)\Delta t K)u^n + \Delta t[(1-\theta)q^n + \theta q^{n+1}].$$

Dirichlet DOFs are enforced strongly and the reduced system is solved for unknown DOFs.

8.14 Post-Processing and Visualization

```
figure;
patch('Faces',elem_connect,'Vertices',[x_glbl',y_glbl'],...
      'FaceVertexCData',conc_history(:,1),...
      'FaceColor','interp','EdgeColor','none');
colorbar; title('Concentration at t=0 s');

figure;
for step=1:nt+1
  patch('Faces',elem_connect,'Vertices',[x_glbl',y_glbl'],...
        'FaceVertexCData',conc_history(:,step),...
```

```
            'FaceColor','interp','EdgeColor','none');
      colorbar;
      title(sprintf('Concentration at t=%.2f s',(step-1)*deltat));
      drawnow;
    end

    step_lo=floor(t_plot/deltat);
    step_hi=ceil(t_plot/deltat);
    alpha=(t_plot-step_lo*deltat)/deltat;
    conc_plot=(1-alpha)*conc_history(:,step_lo+1)+alpha*conc_history(:,step_hi+1);
    figure;
    patch('Faces',elem_connect,'Vertices',[x_glbl',y_glbl'],...
          'FaceVertexCData',conc_plot,...
          'FaceColor','interp','EdgeColor','none');
    colorbar; title(sprintf('Snapshot at t=%.2f s',t_plot));

    figure;
    t_vec=0:deltat:nt*deltat;
    plot(t_vec,conc_history(node_t,:),'-o');
    xlabel('Time (s)'); ylabel(sprintf('C at node %d',node_t));
    title(sprintf('Time history at node %d',node_t));
```

Explanation: We use MATLAB's `patch` for spatial plots, loop for animation, interpolate for a snapshot at t_{plot}, and plot time series at a selected node.

8.15 Workspace Assignment & Output Arguments

```
    assignin('base','K_global',K_global);
    assignin('base','C_global',C_global);
    assignin('base','q_global',q_global);
    assignin('base','conc_history',conc_history);
    if nargout>0
      varargout={K_global,C_global,q_global,...
                 x_start,x_end,y_start,y_end,...
                 x_glbl,y_glbl,elem_connect,...
                 tot_elem,node_id,tot_dofs,...
                 elem_start,ndes_per_elmn};
    end
end
```

Explanation: Key matrices and the full solution history are exported to the MATLAB base workspace and returned if requested.

9 Convergence/Mesh Independence test:

To verify the reliability and accuracy of the finite element code lies a convergence or mesh independence study is conducted. This analysis is essential to ensure that the numerical solution approaches the exact or analytical solution as the mesh is refined—i.e., as the number of elements increases. A solution is said to be convergent when further refinement of the mesh (i.e., increasing the number of elements) does not significantly alter the result. This behavior not only confirms the consistency of the numerical method but also aids in identifying the optimal mesh density that balances accuracy and computational cost.

In the present study, the convergence behavior of the developed finite element model was assessed using the solution to a diffusion problem described in Example 8.5.1 of "Introduction to the Finite Element Method" by J.N. Reddy, as this provided us with an analytical solution to compare our results with. The chosen point of observation for the concentration was at the coordinates **(1,1)**. Both the finite element result and the corresponding exact solution were used for comparison, with the exact concentration value at that point being **0.6171.**

A systematic refinement of the mesh was performed, increasing the number of elements from 6 to 864. The concentration values obtained through the finite element simulation at each mesh resolution were recorded. The following table shows the concentration at (1,1) for different number of elements.

No. of elements	Numerical solution	Analytical solution
6	0.608816	
12	0.612154	
24	0.614251	
40	0.616113	
96	0.618291	
200	0.618157	
320	0.617936	
448	0.617729	
654	0.617531	
816	0.617364	0.6171
864	0.617364	

Figure 8: Concentration at (1,1) for different number of elements

The table shows a clear trend toward convergence. For coarse meshes (e.g., 6 to 40 elements), there is a noticeable deviation from the exact value(0.6171), reflecting the expected inaccuracy due to insufficient spatial resolution. However, as the mesh density increases (beyond approximately 96 elements), the numerical solution approaches and

eventually stabilizes near the exact solution. Notably, beyond 448 elements, changes in the solution become negligibly small, indicating that the model has reached mesh independence.

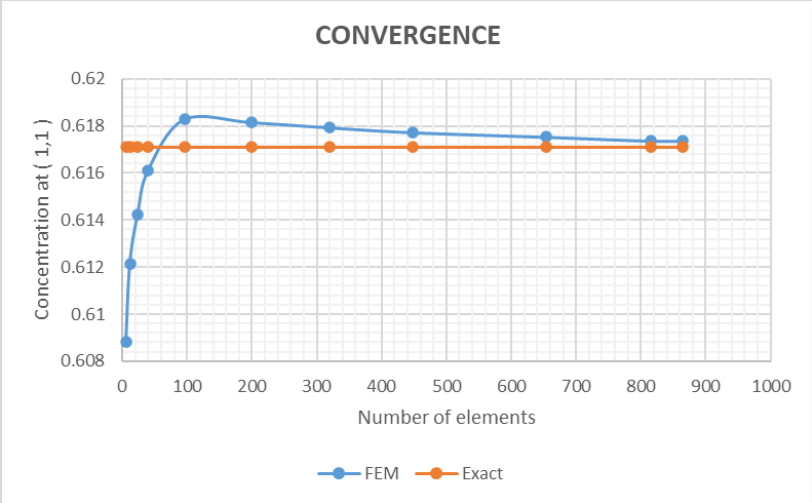


Figure 9: Comparison of FEM and analytical solution at (1,1)for different number of elements

This behavior is clearly illustrated in the plot, where the FEM solution curve (in blue) flattens and aligns closely with the exact solution (in orange). Minor discrepancies observed in the FEM results at higher mesh densities are within acceptable tolerance levels and can be attributed to several factors. These include discretization errors inherent to the finite element method, numerical integration approximations within the element formulation, and potential accumulation of round-off errors during matrix assembly and solution.

Thus, this convergence study validates the numerical accuracy and consistency of the finite element formulation.

10 Stability Test:

To evaluate the stability characteristics of our numerical model, the *generalized θ-method* is employed , which encompasses both explicit and implicit time integration schemes. The main objective of this section is to **verify whether the solution behavior aligns with theoretical expectations when Δt is chosen within the stable range**, and to observe the consequences when it is not.

The θ-method allows us to control the weighting between the current and next time levels in the finite element time integration. Two specific cases were analyzed:

- $\theta = 0$: representing the *Euler Forward scheme*
- $\theta = 0.5$: representing the *Crank–Nicolson scheme*

To check for the stability of the two schemes, a rectangular geometry was tested for a total time of 1 second. Its initial state is as shown below.

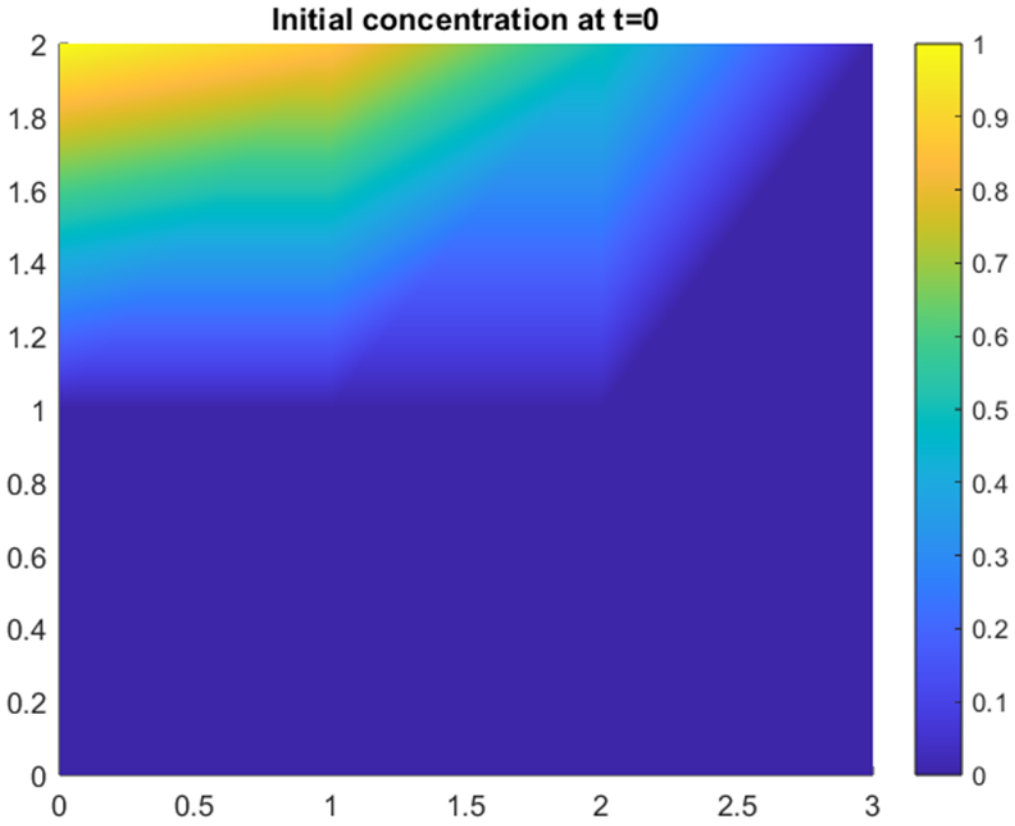


Figure 10: Concentration contour at At t = 0

(i) Euler Forward Scheme ($\theta = 0$)

Using the generalized θ-formulation, the stability range for the Euler method was determined to be:

$$0 \leq \Delta t \leq 0.0056\,\text{sec}$$

To validate this, two simulations were conducted:

- **Within Stable Range ($\Delta t = 0.005$ sec):**
  The time history of concentration at node 5 exhibits a smooth, monotonic increase toward steady state, consistent with physical expectations. The interpolated concentration at $(1.0, 1.0)$ is 0.608814. This confirms that when Δt lies within the theoretical stable range, the numerical scheme produces an accurate and well-behaved solution as shown in the figures below.

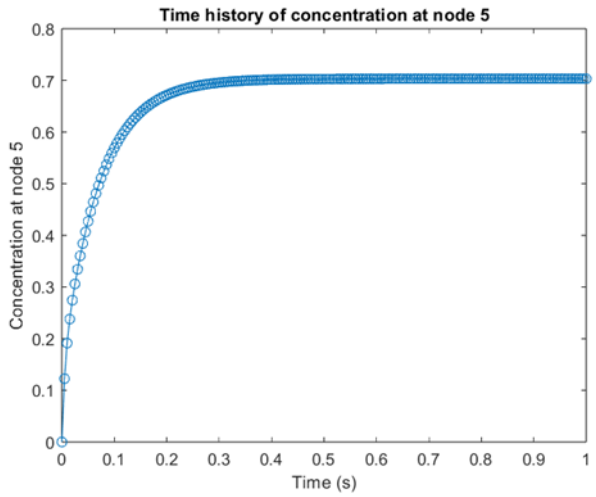


Figure 11: Variation of concentration with time at node 5 using Euler forward scheme with $\Delta t = 0.005$

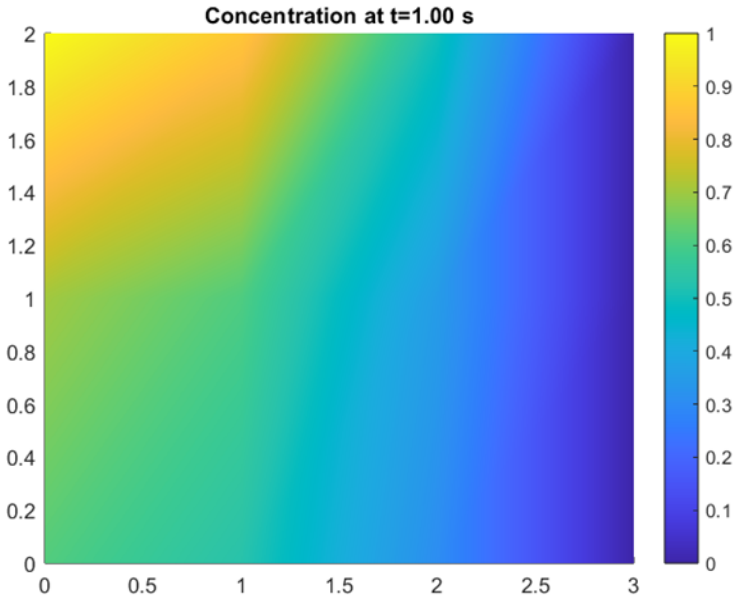


Figure 12: Contour plot at t=1 second using Euler forward scheme with $\Delta t = 0.005$

– **Outside Stable Range ($\Delta t = 0.01$ sec):**
The simulation becomes unstable. The concentration profile at node 5 shows non-physical oscillations and rapidly diverges, reaching extreme values (e.g., 4629.73), far beyond any physically reasonable range. This is depicted though the figures below. This behavior highlights the sensitivity of explicit methods to time step selection and confirms that violating the stability criterion leads to loss of numerical control

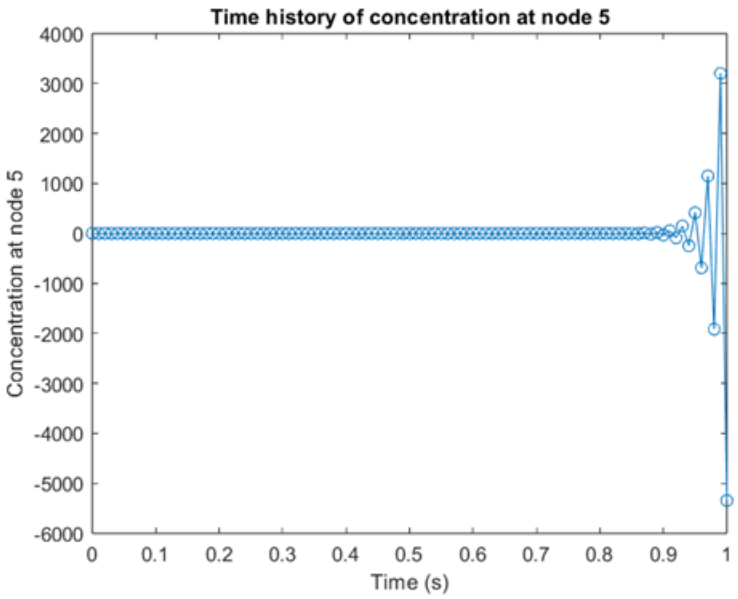


Figure 13: Variation of concentration with time at node 5 using Crank-Nicolson scheme with $\Delta t = 0.01$

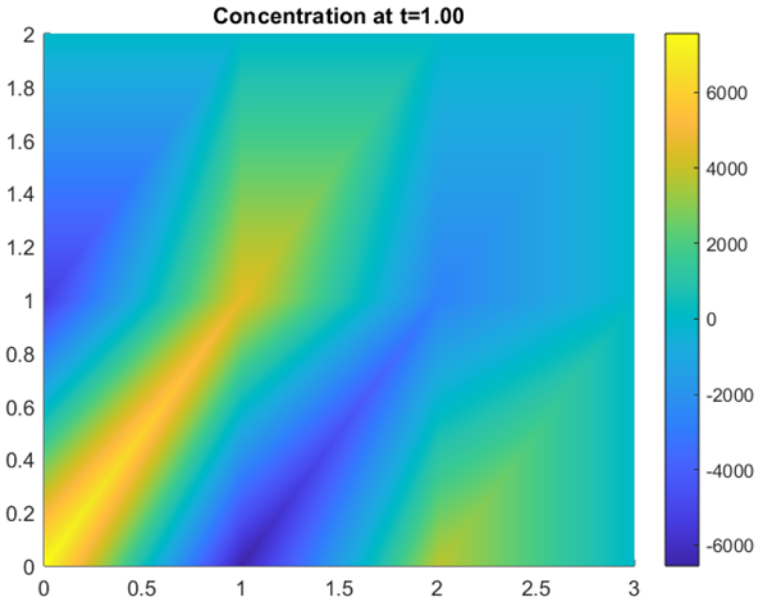


Figure 14: Contour plot at t=1 second using Crank-Nicolson with $\Delta t = 0.01$

(ii) Crank–Nicolson Scheme ($\theta = 0.5$)

The Crank–Nicolson scheme, on the other hand, is unconditionally stable for linear problems. According to the θ-method formulation, the method remains stable for any

value of Δt:

$$\Delta t \in [0, \infty)$$

To verify this, we ran a simulation with $\Delta t = 0.01$ sec—the same value that caused instability in the Euler scheme. The resulting time history at node 5 was smooth and stable, with a concentration value at $(1, 1)$ of 0.608815(same as for the stable case of $\theta = 0$ and $\Delta t = 0.005$), consistent with theoretical expectations.

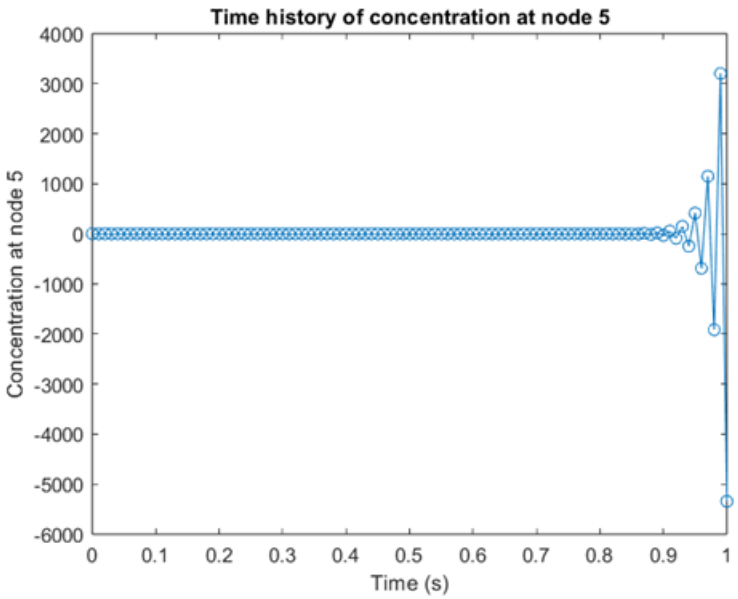


Figure 15: Variation of concentration with time at node 5 using Crank-Nicolson scheme with $\Delta t = 0.01$

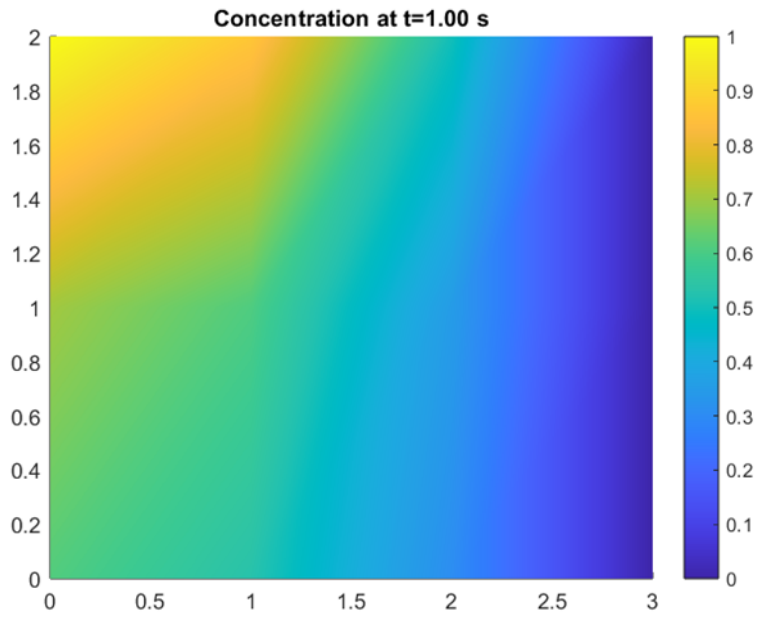


Figure 16: Contour plot at t=1 second using Crank-Nicolson with $\Delta t = 0.01$

This above figures confirms that the Crank–Nicolson scheme remains stable across a broad range of time steps and demonstrates its robustness.

Hence, for the Euler Forward scheme, results were accurate only within the prescribed stability bounds. In contrast, the Crank–Nicolson method produced reliable results regardless of Δt, demonstrating its unconditional stability.

11 Validation:

To ensure the accuracy and reliability of the developed FEM code, the numerical results were validated against the analytical solution from the book ' An introduction to the final Element Methods by J.N. Reddy[5]' and the numerical solution obtained from ABAQUS, which is a well-known and reliable software. The governing equation, initial, and boundary conditions were chosen so that an exact analytical solution is available for comparison. Four cases were considered to verify the correctness of the developed finite element model.

11.1 Validation with Analytical and ABAQUS Solution for a Steady-State Diffusion Problem with Homogeneous Neumann condition

For this case, we consider steady-state conditions, constant diffusivity, and no reaction term. The equation simplifies to:

Figure 17: Domain and discretizations as given in the book

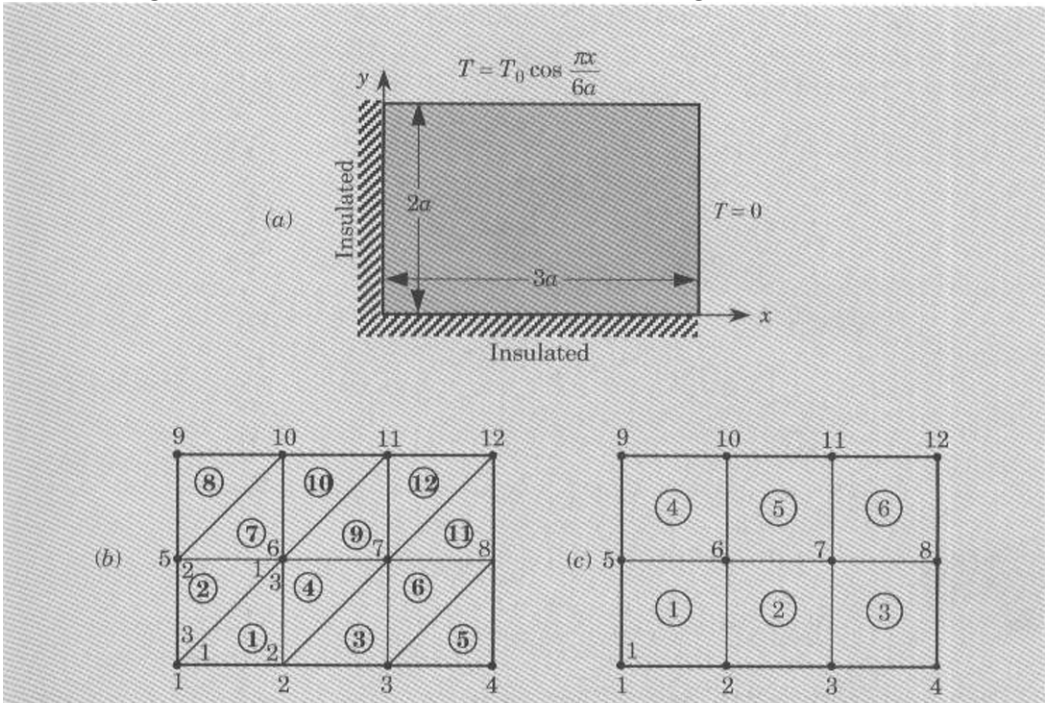


Figure 8.5.2 Finite element analysis of a heat conduction problem over a rectangular domain: (*a*) domain; (*b*) mesh of linear triangular elements; and (*c*) mesh of linear rectangular elements.

$$\frac{\partial}{\partial x}\left(D\frac{\partial C}{\partial x}\right) + \frac{\partial}{\partial y}\left(D\frac{\partial C}{\partial y}\right) = 0 \tag{1}$$

For constant D , this reduces to :

$$D\nabla^2 C = 0 \tag{2}$$

This is directly analogous to the steady-state heat conduction equation used in example 8.5.1 of the book which is:

$$-k\nabla^2 T = 0 \tag{3}$$

The example is solved for the temperature distribution using finite element methods at each of the nodes using linear triangular and rectangular elements.

In our formulation, concentration C replaces temperature T, and diffusivity D replaces thermal conductivity k. The rest of the formulation, the boundary conditions, and the domain remain unchanged. The problem setup considers a rectangular domain of dimensions $3a \times 2a$, discretized using linear rectangular and triangular finite elements, as shown in the image below. We will only validate for rectangular elements.

The boundary conditions are as follows:

– Left ($x = 0$) and bottom ($y = 0$) boundaries are insulated (Neumann condition).

– Right boundary ($x = 3a$) is maintained at zero (Dirchlet).

– Top boundary ($y = 2a$) has a specified concentration profile (Dirchlet):

$$C = C_0 \cos\left(\frac{\pi x}{6a}\right)$$

The following image shows the results after solving the problem. The table in the figure compares the nodal temperatures computed by the finite element method with the corresponding values from the analytical solution.

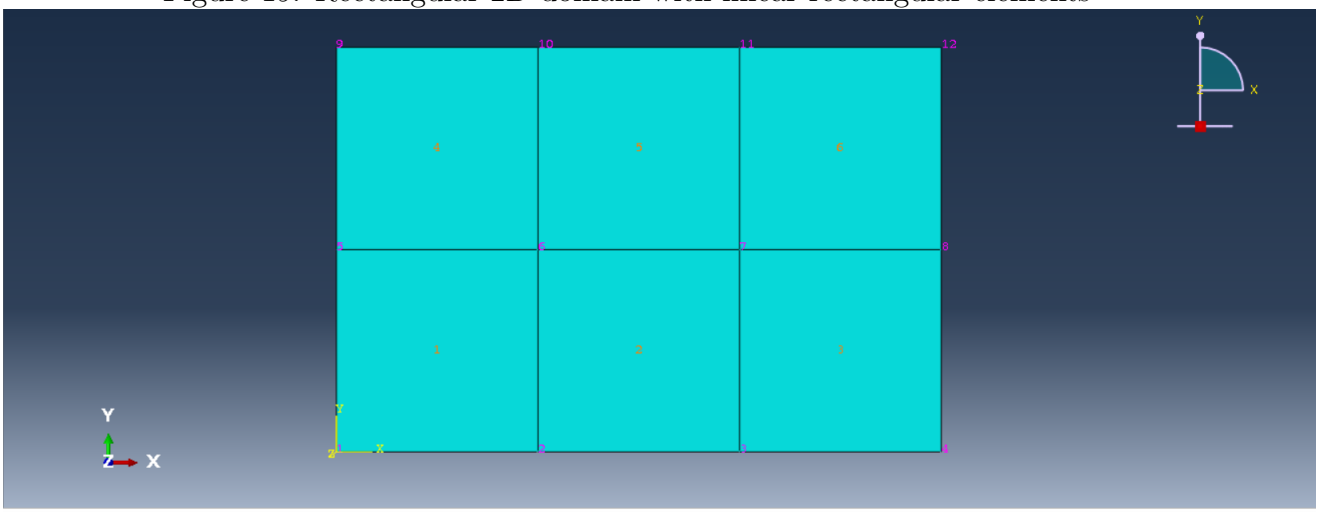
Figure 18: Solution of example 8.5.1

Table 8.5.1 Comparison of the nodal temperatures $T(x, y)/T_0$, obtained using various finite element meshes† with the analytical solution (Example 8.5.1).

		Triangles		Rectangles		Analytical
x	y	3×2	6×4	3×2	6×4	solution
0.0	0.0	0.6362	0.6278	0.6128	0.6219	0.6249
0.5	0.0	—	0.6064	—	0.6007	0.6036
1.0	0.0	0.5510	0.5437	0.5307	0.5386	0.5412
1.5	0.0	—	0.4439	—	0.4398	0.4419
2.0	0.0	0.3181	0.3139	0.3064	0.3110	0.3124
2.5	0.0	—	0.1625	—	0.1610	0.1617
0.0	1.0	0.7214	0.7148	0.7030	0.7102	0.7125
0.5	1.0	—	0.6904	—	0.6860	0.6882
1.0	1.0	0.6248	0.6190	0.6088	0.6150	0.6171
1.5	1.0	—	0.5054	—	0.5022	0.5038
2.0	1.0	0.3607	0.3574	0.3515	0.3551	0.3563
2.5	1.0	—	0.1850	—	0.1838	0.1844

The same problem is then solved by modeling and simulating it in ABAQUS. The geometry is created to match the dimensions given in the textbook example, with the domain size set to $3a \times 2a$. The geometry and the mesh is shown in the figure below.

Figure 19: Rectangular 2D domain with linear rectangular elements



Comparison of results:

To validate the results, we compare the results obtained from the FEM code, the solved example from the book and ABAQUS.

The following figures provide a comparison of the concentration contour plots from the MATLAB code and ABAQUS.

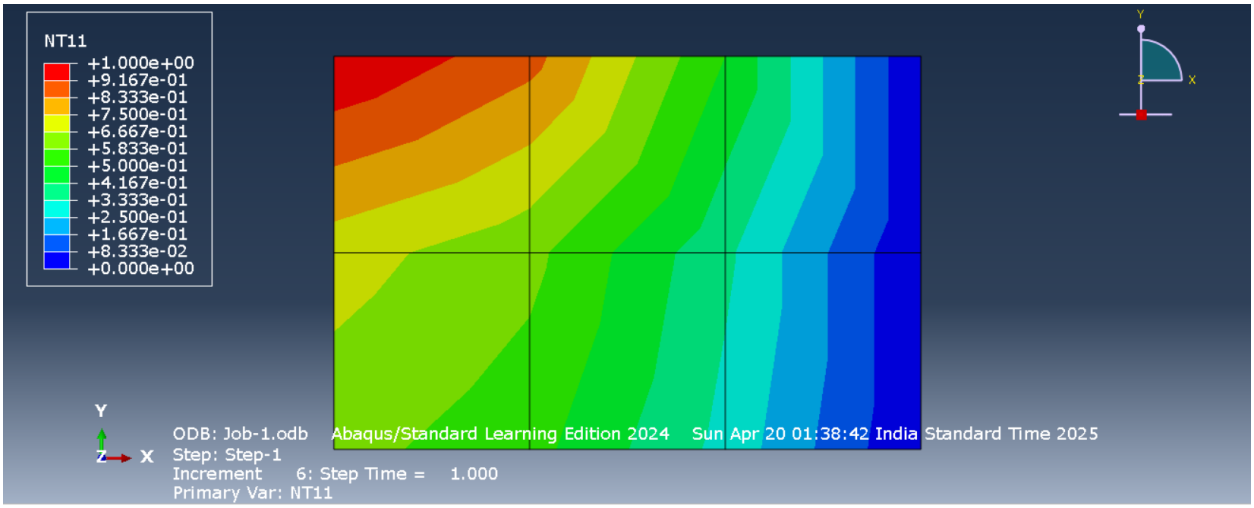


Figure 20: Concentration contour plot from ABAQUS

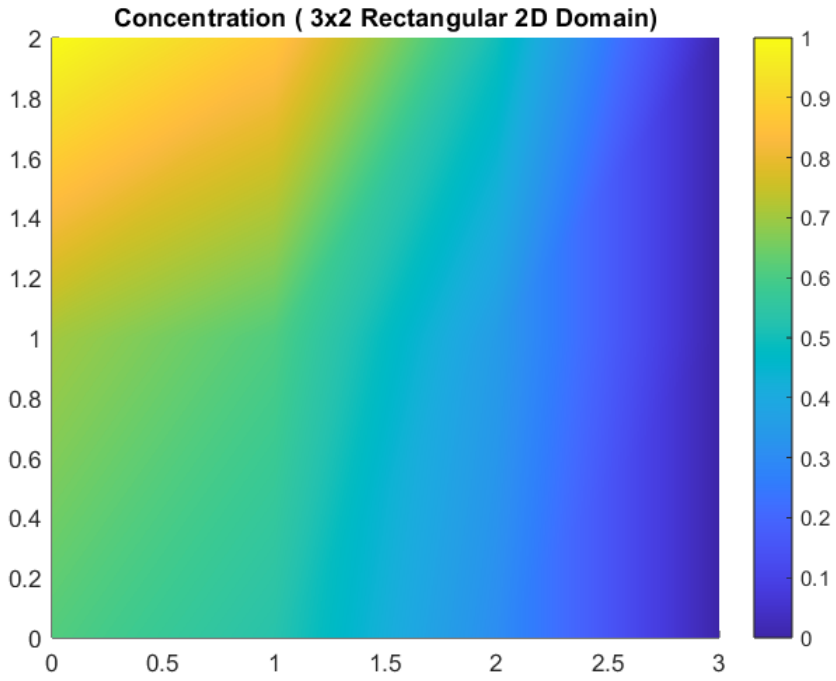


Figure 21: Concentration contour plot from MATLAB code

The comparison shows an excellent match between the two results. The concentration field exhibits identical patterns, with the same locations of maxima and minima, and similar gradient distributions. The contour lines are symmetric, and follow the same shape in both cases.

The following figures shows the nodal concentration values at each node from the MATLAB code, ABAQUS and the solved example from the book.

12x1 double

	1	2
1	0.6127	
2	0.5308	
3	0.3064	
4	0	
5	0.7031	
6	0.6088	
7	0.3515	
8	0	
9	1	
10	0.8660	
11	0.5000	
12	0	

Figure 22: Nodal concentrations from MATLAB code

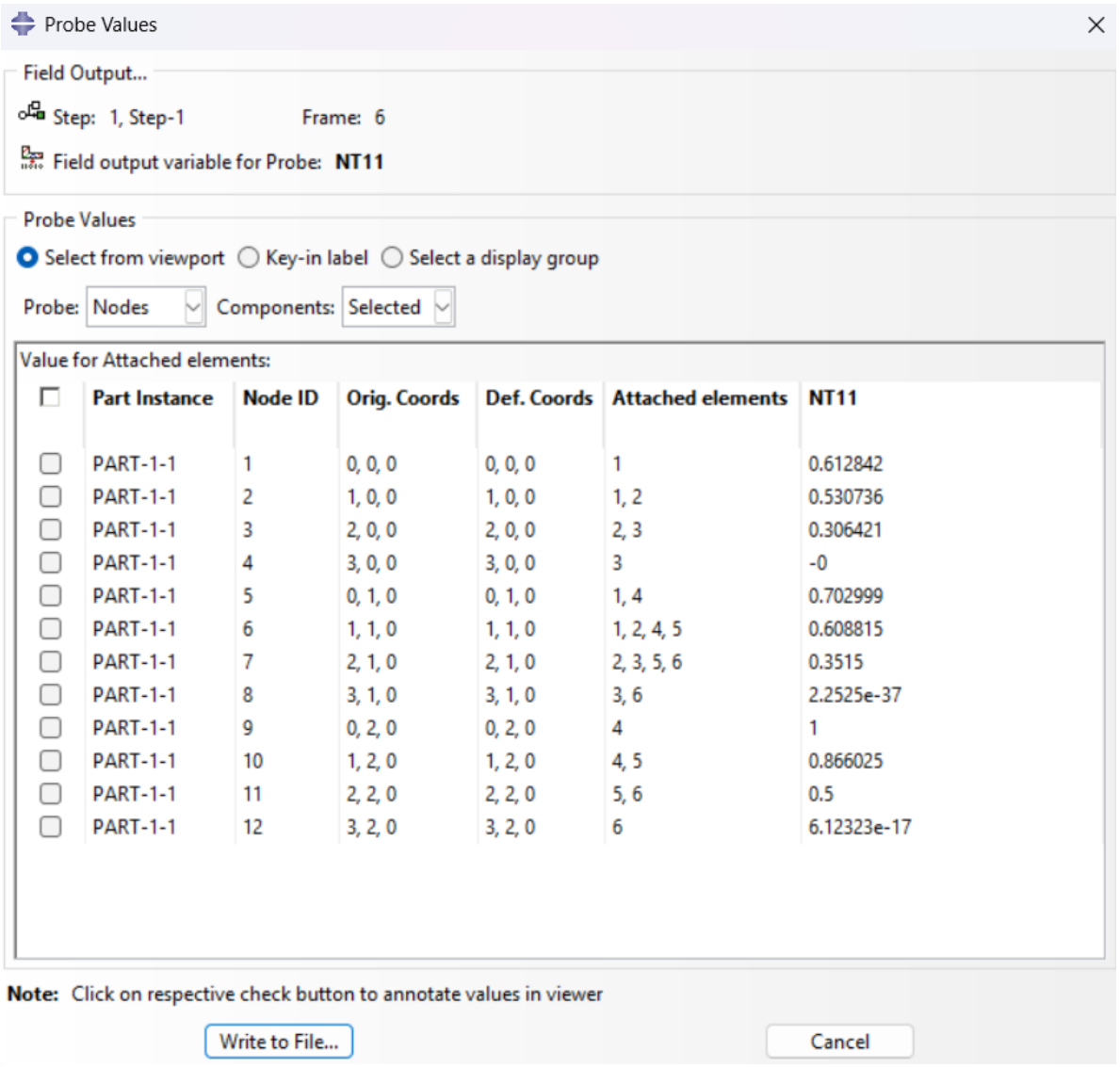


Figure 23: Nodal concentrations from ABAQUS

	x	y	3 × 2	solution
Node 1	0.0	0.0	0.6128	0.6249
Node -	0.5	0.0	—	0.6036
Node 2	1.0	0.0	0.5307	0.5412
Node -	1.5	0.0	—	0.4419
Node 3	2.0	0.0	0.3064	0.3124
Node -	2.5	0.0	—	0.1617
Node 5	0.0	1.0	0.7030	0.7125
Node -	0.5	1.0	—	0.6882
Node 6	1.0	1.0	0.6088	0.6171
Node -	1.5	1.0	—	0.5038
Node 7	2.0	1.0	0.3515	0.3563
Node -	2.5	1.0	—	0.1844

Figure 24: Nodal Concentration(FEM and analytical) from book

Upon comparison, it is observed that the nodal concentration values computed by the FEM code are in excellent agreement with those obtained from both the ABAQUS simulation and the analytical solution provided in the solved example from the textbook confirming the accuracy of the implemented finite element formulation.

11.2 Validation with ABAQUS Solution for a transient Diffusion Problem with non-homogeneous Neumann condition

We take the same case as the first one with two differences . First being non zero flux is applied at the left and top edge.

The boundary conditions are as follows:

- Left ($x = 0$) boundary has a flux of $\partial C/\partial x$ = -20 (Neumann condition)
- bottom ($y = 0$) boundary has a flux of $\partial C/\partial y$ =-40 (Neumann condition)

– Right boundary ($x = 3a$) is maintained at zero concentration.(Dirchlet)

– Top boundary ($y = 2a$) has a specified concentration profile (Dirchlet):

$$C = C_0 \cos\left(\frac{\pi x}{6a}\right)$$

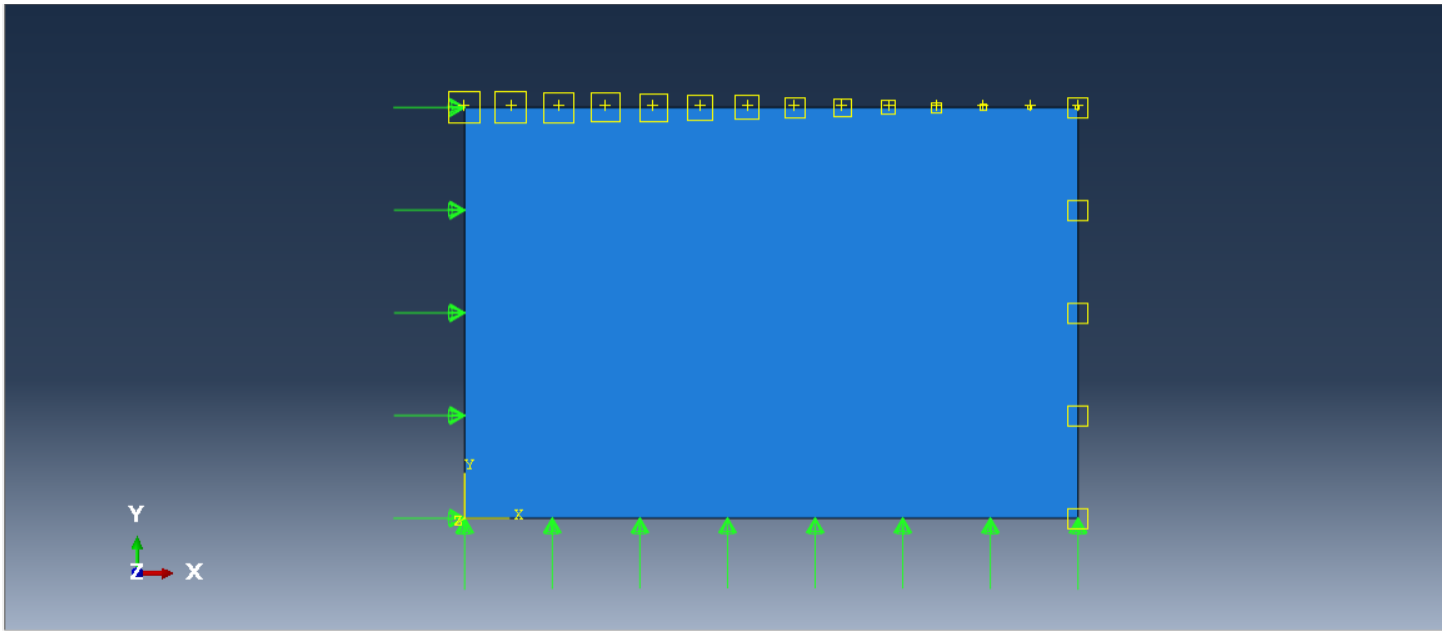


Figure 25: Geometry and boundary conditions

Secondly, for transient case, the problem is solved over a total simulation time of $T = 10$ seconds using a time step size of $\Delta t = 0.1$ seconds. The time integration is carried out using the Crank-Nicolson scheme, which corresponds to a time weighting factor of $\theta = 0.5$.

Comparison of results:

The following figures show the results obtained from the MATLAB code.

12x101 double

	1	2	3	4
1	0	94.7369	88.1975	103.6513
2	0	70.6165	73.3348	80.4310
3	0	58.3173	45.6032	61.1275
4	0	0	0	0
5	0	45.7867	55.7370	56.7045
6	0	21.6506	43.8571	34.3398
7	0	11.2427	24.7128	18.8385
8	0	0	0	0
9	1	1	1	1
10	0.8660	0.8660	0.8660	0.8660
11	0.5000	0.5000	0.5000	0.5000
12	6.1232e-17	6.1232e-17	6.1232e-17	6.1232e-17

Figure 26: Nodal concentration at first 4 time steps.

98	99	100	101	102
98.5919	98.5919	98.5919	98.5919	
78.7596	78.7596	78.7596	78.7596	
55.1024	55.1024	55.1024	55.1024	
0	0	0	0	
57.1613	57.1613	57.1613	57.1613	
39.2235	39.2235	39.2235	39.2235	
21.8064	21.8064	21.8064	21.8064	
0	0	0	0	
1	1	1	1	
0.8660	0.8660	0.8660	0.8660	
0.5000	0.5000	0.5000	0.5000	
6.1232e-17	6.1232e-17	6.1232e-17	6.1232e-17	

Figure 27: Nodal concentrations at last 4 time steps.

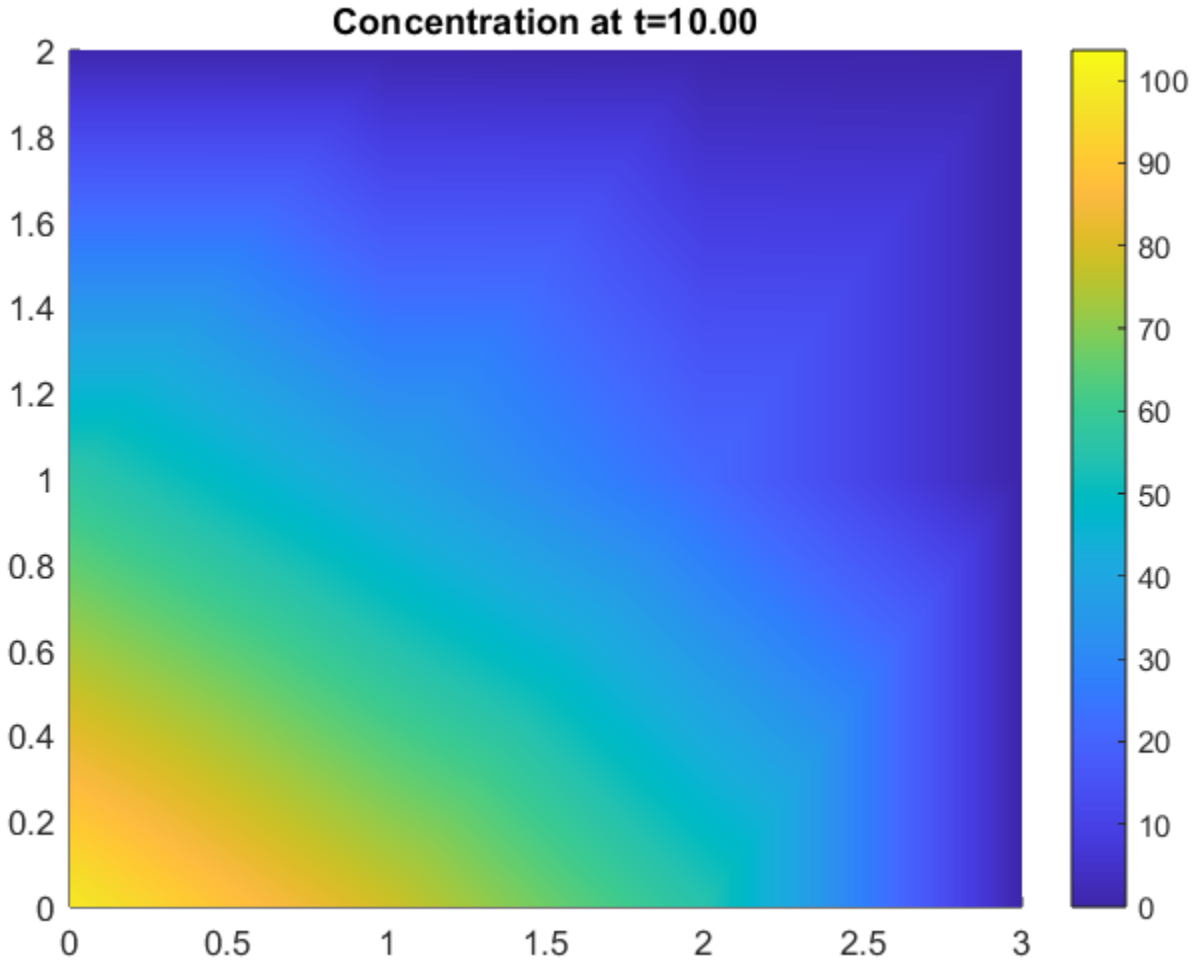


Figure 28: Concentration contour plot from MATLAB code

From the second figure, it can be observed that the nodal concentration values remain unchanged across successive time steps.This indicates that the solution has reached a steady state, where the transient behavior has diminished and the system no longer exhibits any temporal variation, i.e. the solution has become consistent. The final column, highlighted in black, represents the nodal concentration values at the last time step. These values correspond to the steady-state solution of the transient diffusion problem and are used for comparison with the results obtained from ABAQUS.

The results from ABAQUS are shown below:

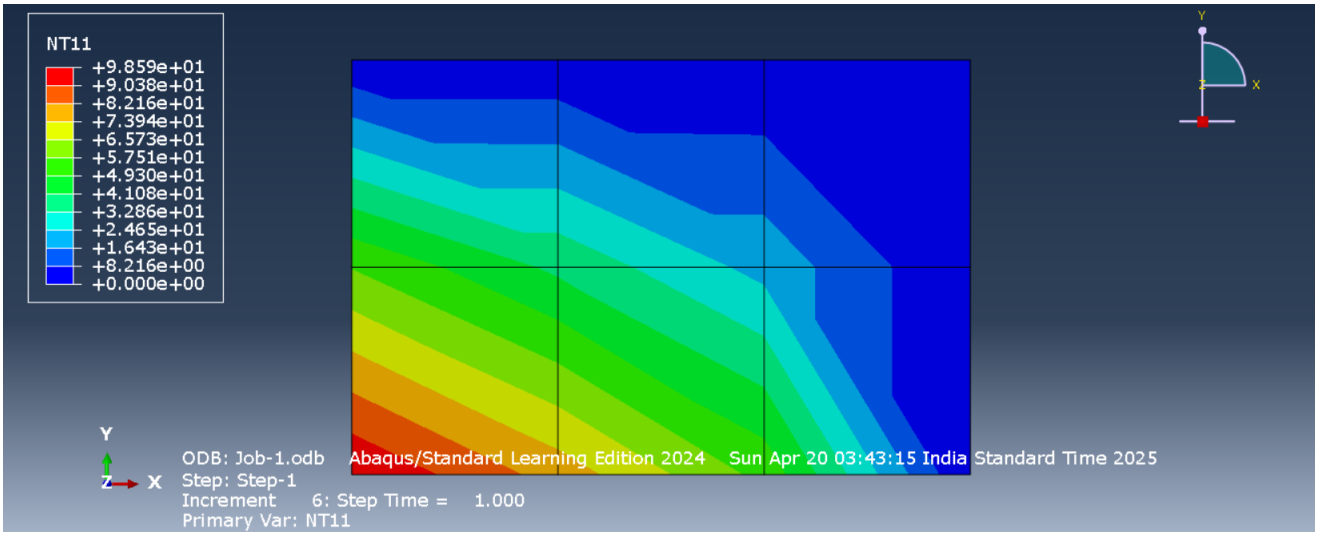


Figure 29: Concentration contour plot from ABAQUS

Probe Values

Field Output...

Step: 1, Step-1 Frame: 6

Field output variable for Probe: **NT11**

Probe Values

Select from viewport | Key-in label | Select a display group

Probe: Nodes Components: Selected

Value for Attached elements:

Part Instance	Node ID	Orig. Coords	Def. Coords	Attached elements	NT11
PART-1-1	1	0, 0, 0	0, 0, 0	1	98.5919
PART-1-1	2	1, 0, 0	1, 0, 0	1, 2	78.7596
PART-1-1	3	2, 0, 0	2, 0, 0	2, 3	55.1024
PART-1-1	4	3, 0, 0	3, 0, 0	3	-0
PART-1-1	5	0, 1, 0	0, 1, 0	1, 4	57.1613
PART-1-1	6	1, 1, 0	1, 1, 0	1, 2, 4, 5	39.2235
PART-1-1	7	2, 1, 0	2, 1, 0	2, 3, 5, 6	21.8064
PART-1-1	8	3, 1, 0	3, 1, 0	3, 6	3.80107e-36
PART-1-1	9	0, 2, 0	0, 2, 0	4	1
PART-1-1	10	1, 2, 0	1, 2, 0	4, 5	0.866025
PART-1-1	11	2, 2, 0	2, 2, 0	5, 6	0.5
PART-1-1	12	3, 2, 0	3, 2, 0	6	6.12323e-17

Note: Click on respective check button to annotate values in viewer

Write to File... Cancel

Figure 30: Nodal Concentrations from ABAQUS at the end of last time step

Upon comparison, it is observed that the nodal concentration values computed by the FEM code are in excellent agreement with those obtained from both the ABAQUS simulation. This visual agreement confirms that the spatial variation of the solution computed by the FEM code is consistent with that obtained from the commercial finite element package, thereby validating the correctness of the implementation.

11.3 Validation with ABAQUS Solution for a Steady-State Diffusion Problem with a hexagonal geometry and non-homogeneous condition

In this case, we validate the finite element implementation for a steady-state diffusion problem defined over a hexagonal domain. This geometry, which is shownn in the figure below, introduces additional complexity compared to rectangular domains and serves as a test for the generality and robustness of the FEM code.

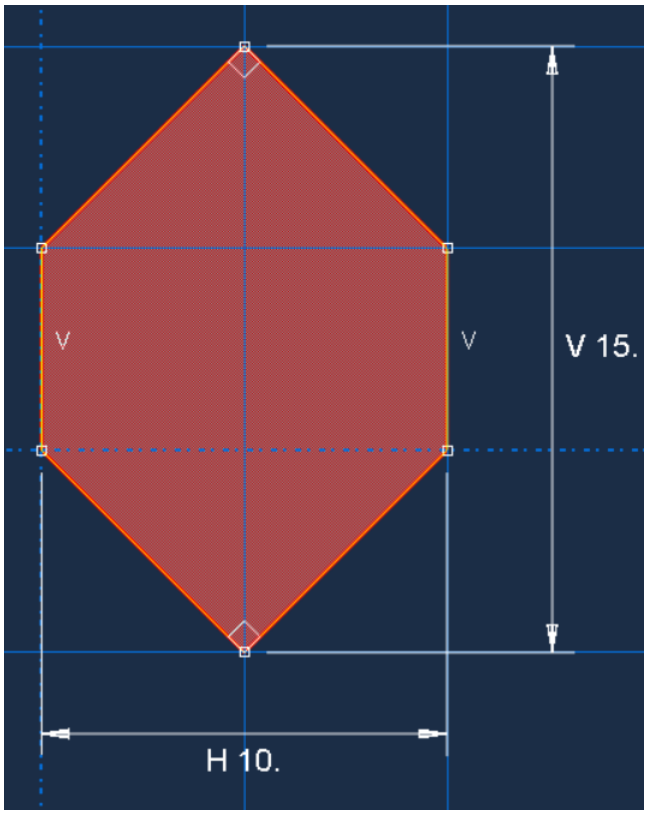


Figure 31: Hexagonal domain

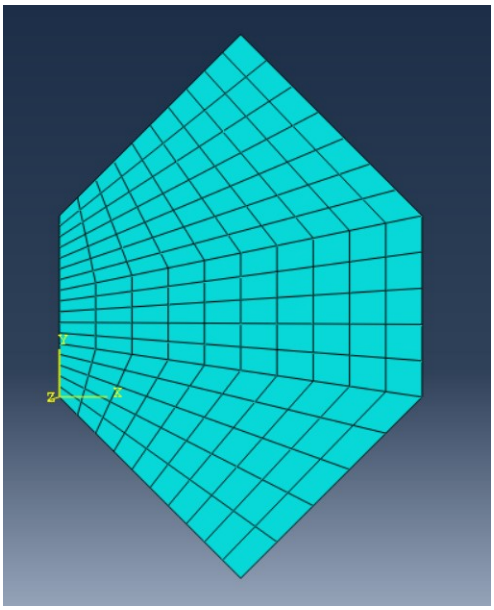


Figure 32: Meshed into 198 elements

The boundary conditions applied are non-homogeneous Dirichlet and Neumann conditions on different edges of the hexagon as shown in the figure.

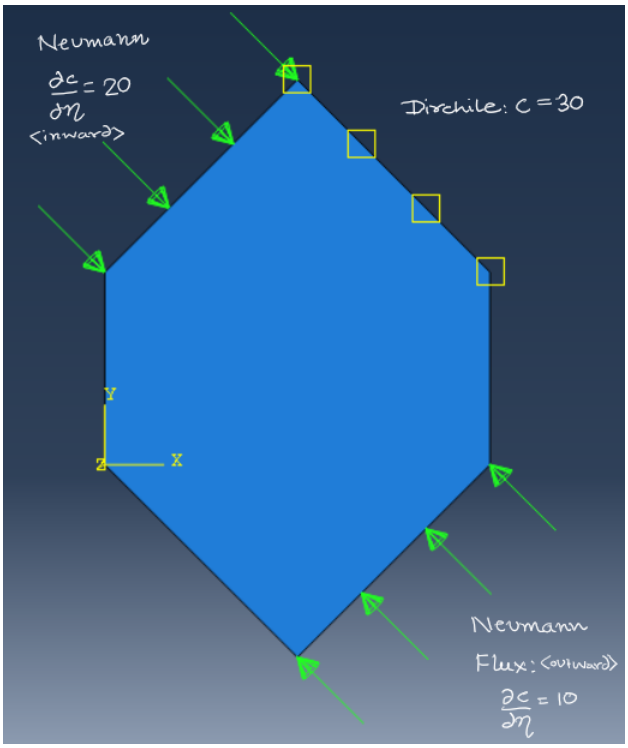


Figure 33: Direchlet and Neumann boundary conditions applied on all the edges

the problem is solved over a total simulation time of $T = 500$ seconds using a time step

size of $\Delta t = 0.1$ seconds. The time integration is carried out using the Crank-Nicolson scheme, which corresponds to a time weighting factor of $\theta = 0.5$.

Comaparison of results:

The following figures show the results obtained from the MATLAB. To validate the nodal concentration values, nodal values were extracted at three locations: The first 4, middle 4 and last four nodes.

	1
Node 1:	157.2930
Node 2:	152.2574
Node 3:	148.4895
Node 4:	145.5627
Node 99:	110.6630
Node 100:	115.1833
Node 101:	119.7050
Node 102:	124.2358
Node 195:	155.5710
Node 196:	163.3356
Node 197:	168.0310
Node 198:	169.6072

Figure 34: Concentrations at 12 selected nodes at the end of 500 seconds

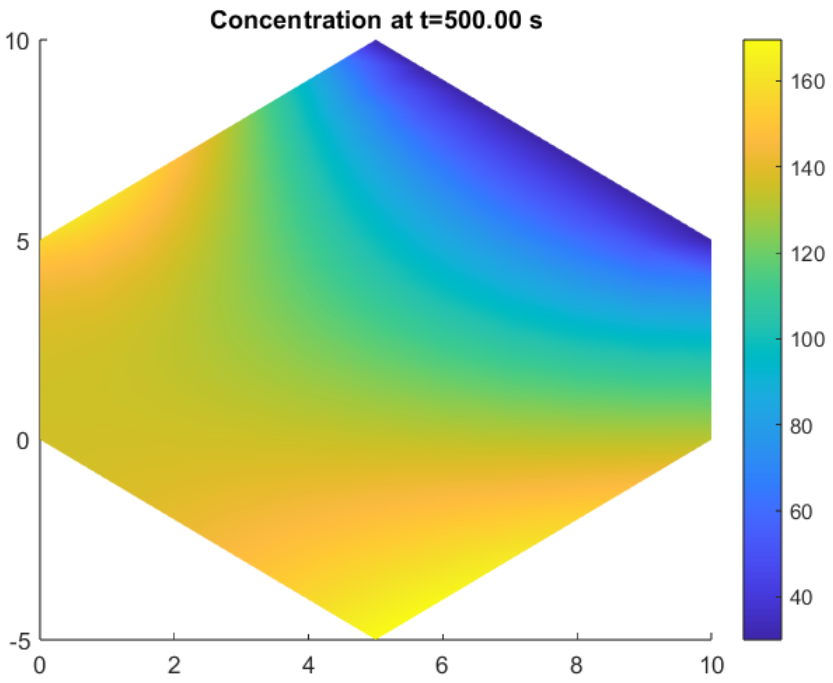


Figure 35: Concentration contour plot at the end of 500 seconds from MATLAB

The results from ABAQUS are shown below. The nodal concentrations are checked at the same nodes i.e., first 4, middle 4, last 4 nodes for the sake of comparison.

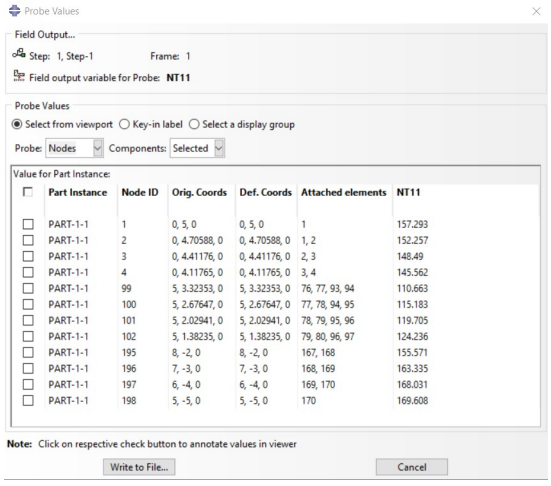


Figure 36: Concentrations at the same 12 selected nodes at the end of 500 seconds

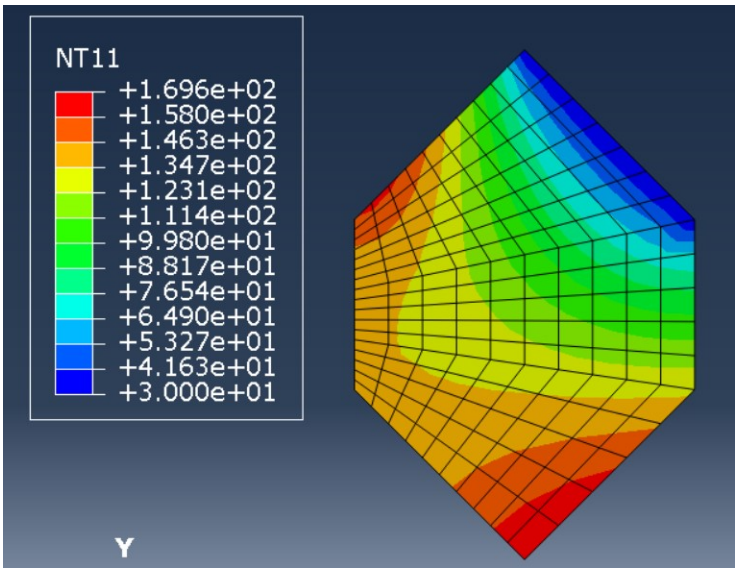


Figure 37: Concentration contour plot at the end of 500 seconds from ABAQUS

The concentration values obtained from ABAQUS and those computed by the FEM code were found to be in perfect agreement at all selected nodes. Additionally, the contour plots generated from both simulations were visually identical, with matching gradients, symmetry, and boundary behavior. This strong agreement in both numerical values and visual representation confirms the correctness of the FEM implementation, as well as the reliability of its interpolation and visualization capabilities, even when applied to complex geometries.

12 Test Cases

To establish the validity and robustness of the developed diffusion-based finite element formulation, a series of benchmark test cases were employed. Each test case was carefully selected to isolate and investigate specific aspects of the model's predictive capabilities under varying boundary conditions and material configurations.

12.1 Effect of Domain Length on Steady-State Convergence:

In the first test case, the finite element model is evaluated for its response to variations in domain length, which effectively simulates different crack lengths in a diffusive medium. The primary objective of this analysis is to assess the time required for the concentration field to attain a steady state as the domain length increases. To this end, simulations were conducted across multiple domain lengths while tracking the concentration evolution at nodes positioned at the farthest end of the domain. The corresponding plots, as illustrated in Figures demonstrate the variation of concentration with respect to time at those farthest nodes.

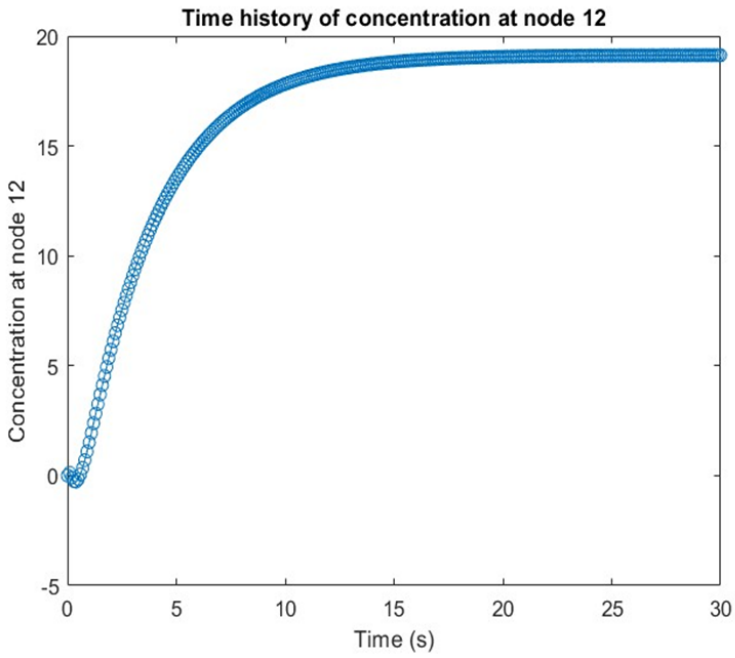


Figure 38: Variation of concentration with time at node 12

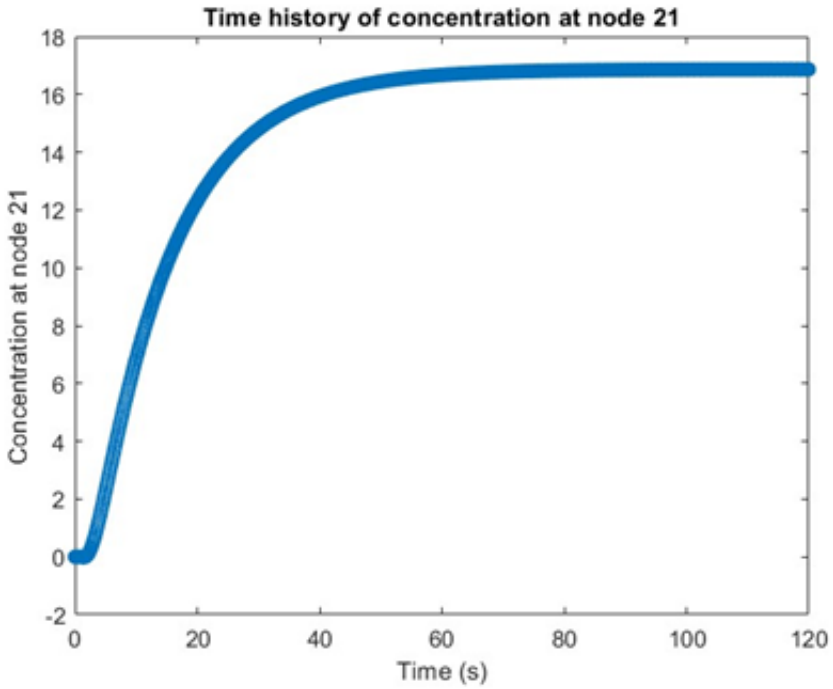


Figure 39: Variation of concentration with time at node 21

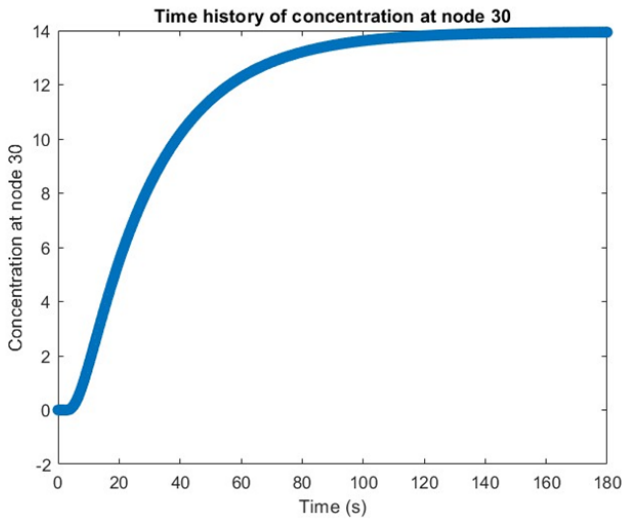


Figure 40: Variation of concentration with time at node 30

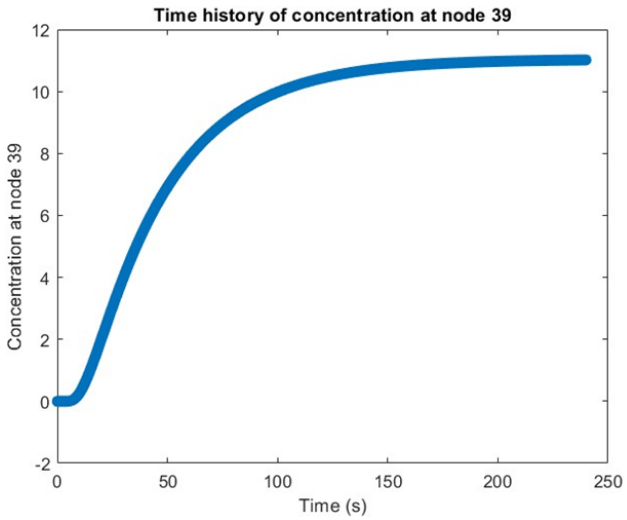


Figure 41: Variation of concentration with time at node 31

The below table shows the approximate time required by the corresponding concentrations to reach steady state.

LENGTH OF THE DOMAIN	NODE	APPROX. TIME(s)
3	12	21.3492
6	21	79.2567
9	30	151.7785
12	31	212.6788

Figure 42: Time required to reach steady state at farthest node of each domain length

From the table we can see a clear trend: the time required to achieve steady-state conditions increases with increasing domain length. This behavior aligns with physical expectations, as a longer domain permits a larger spatial extent for diffusion, thereby prolonging the equilibration process. The results serve to validate the temporal accuracy of the model while providing insights into the influence of geometric scaling on diffusion dynamics.

12.2 Transient Volume Fraction

In this test we track the healing-agent volume fraction

$$\frac{\phi(\mathbf{x},t)}{\phi_0} = \frac{C(\mathbf{x},t)}{C_0},$$

under zero-flux boundaries and a localized initial reservoir.

- **Boundary conditions:** Homogeneous Neumann, $\mathbf{n}\cdot\nabla C = 0$, on all four edges.
- **Initial condition:**

$$C(x,y,0) = \begin{cases} C_0, & x = 0, \\ 0, & x > 0, \end{cases} \quad \Longrightarrow \quad \frac{\phi}{\phi_0}(x,y,0) = \begin{cases} 1, & x = 0, \\ 0, & x > 0. \end{cases}$$

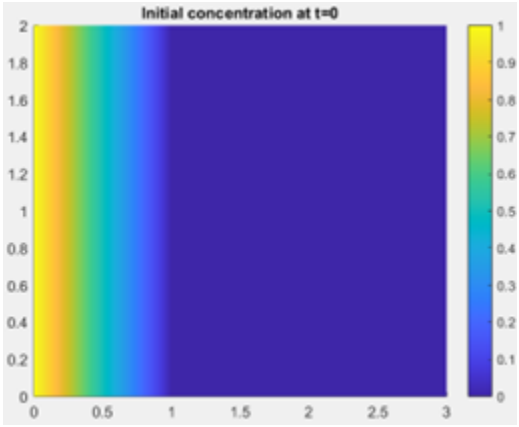


Figure 43: Initial state of the domain

Because the crack is "closed" to its surroundings, diffusion moves the healing agent inward. At a boundary-adjacent node (node 5), ϕ/ϕ_0 falls sharply from 1 to a steady plateau of approximately 0.15. Conversely, at an interior node (node 7), ϕ/ϕ_0 rises from 0 to the same equilibrium value. These profiles (shown below) confirm that, under zero-flux walls and a single-sided reservoir, the healing agent diffuses throughout the crack until diffusion supply and reaction consumption reach balance.

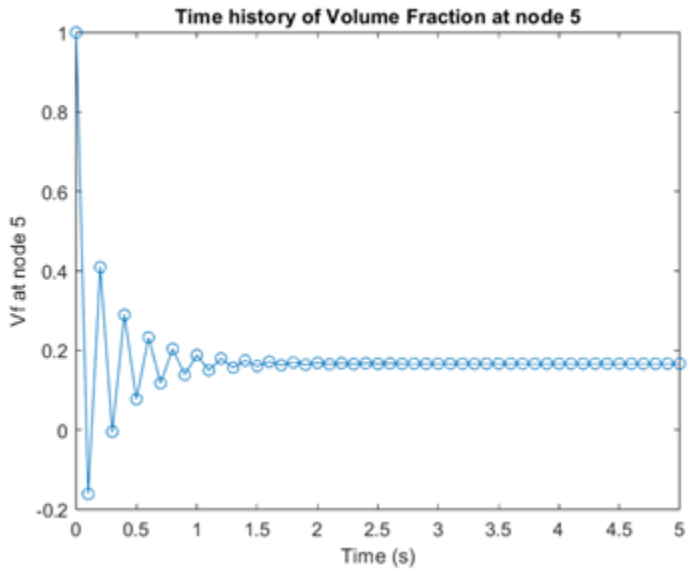


Figure 44: Concentration variation at node 5 (left egde of domain

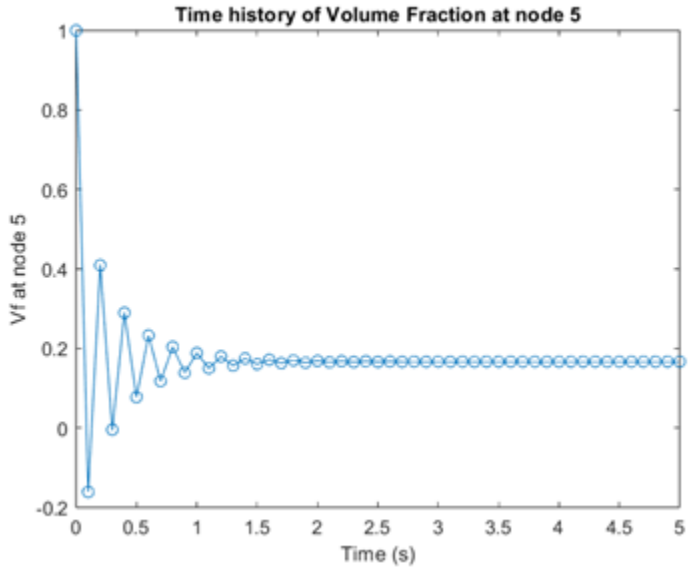


Figure 45: Concentration variation at node 7 (withing the domain

12.3 Diffusity variation with Temperature

In this test case, we examine how temperature-dependent diffusivity influences the time for the healing agent to traverse the crack and reach the farthest node. The key features are:

Left boundary: Dirichlet reservoir $C = 20$ (constant concentration).

Other three edges: Homogeneous Neumann ($\mathbf{n} \cdot \nabla C = 0$), mimicking an insulated crack.

The temperature-dependent diffusivity follows the Arrhenius relation:

$$D(T) = D_0 \exp\Big(-\frac{E_a}{RT}\Big),$$

where D_0 is the pre-exponential diffusivity (m^2/s), E_a the activation energy (J/mol), R the universal gas constant (8.314 J/mol·K), and T the absolute temperature (K).

Mesh and solver settings are identical to Test Case 2 (linear triangular elements, backward-Euler time stepping, Δt chosen for stability).

We run three separate transients at $T = 400$, 500 and 600 K, recalculating $D(T)$ before each run.

Results at Node 12 (upper-right corner):

Temperature (K)	Time to Steady State (s)
400	6.1345
500	0.4932
600	0.1259

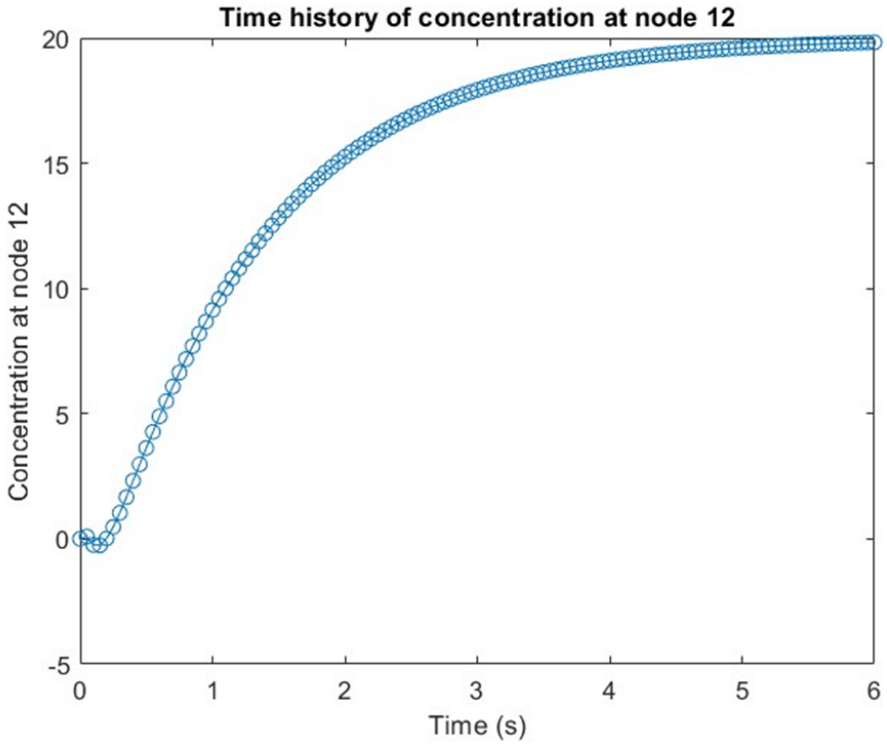


Figure 46: Variation of concentration with time at 400 K

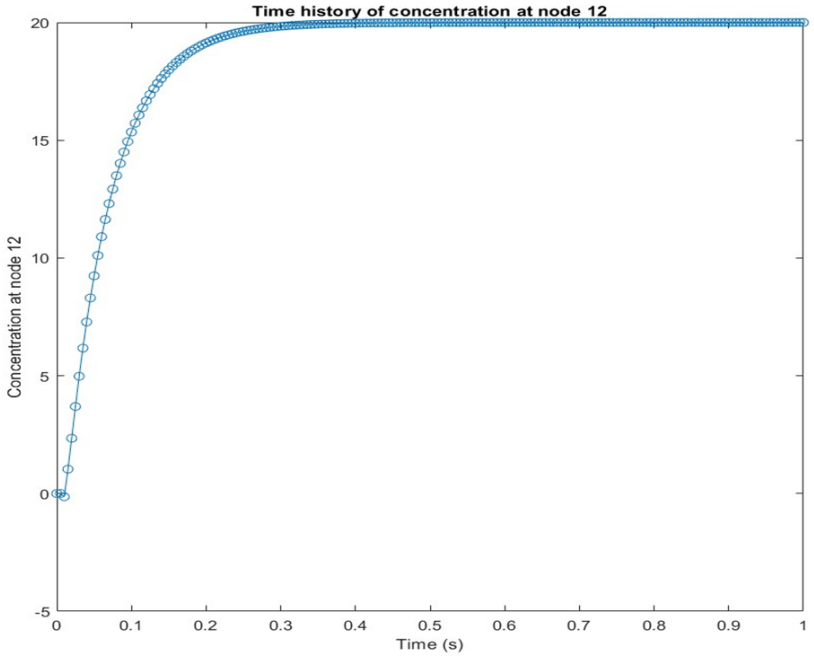


Figure 47: Variation of concentration with time at 500 K

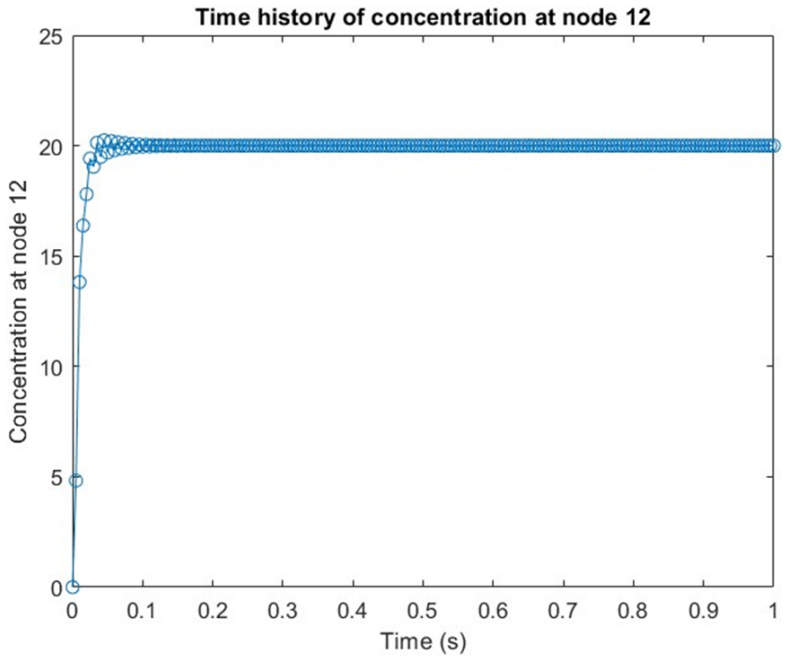


Figure 48: Variation of concentration with time at 600 K

As T increases, $D(T)$ grows exponentially, accelerating healing-agent transport. At 400 K the crack corner takes over six seconds to equilibrate; at 600 K it requires barely 0.13

s. This behavior quantitatively confirms the strong sensitivity of diffusion time scales to temperature in our reaction-diffusion FEM model and underscores the importance of thermal effects when designing self-healing materials.

12.4 Spatially Varying Diffusivity

In this test we again impose $C = 20$ on the left boundary and homogeneous Neumann ($\mathbf{n}\cdot\nabla C = 0$) on the other three edges. The diffusivity is defined spatially as

$$D(x, y) = 15 + 5\,x^2 + 2\,y \quad \left[\mathrm{m}^2/\mathrm{s}\right],$$

which varies from 15 at $(0, 0)$ up to 64 at $(3, 2)$. Consequently, the concentration front advances most slowly near the inlet (low-D region) and accelerates in zones of higher diffusivity. By $t = 10$ s, the entire domain reaches $C = 20$, confirming that a nonuniform $D(x, y)$ field still produces a uniform steady state. The time history at node 12 rises to the prescribed value in under one second—much faster than in the constant-D cases—demonstrating how spatial variability in D can significantly accelerate or retard local diffusion before global equilibrium is attained.

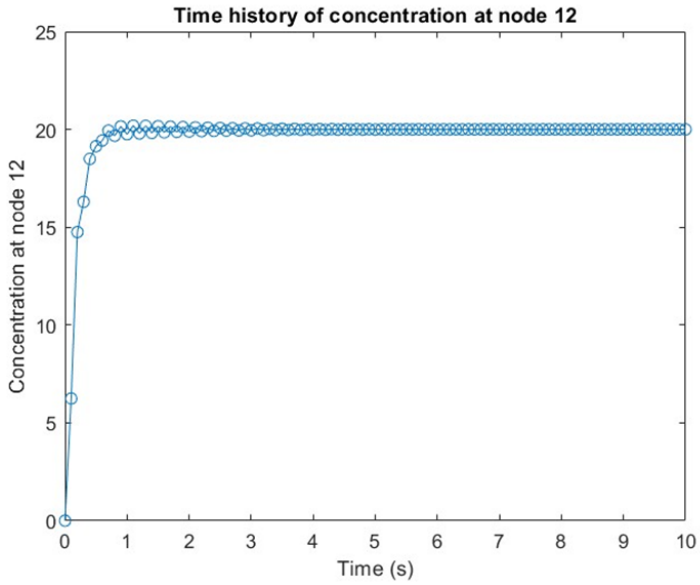


Figure 49: Variation of concentration with node 12

13 Conclusion

We have developed and validated a general-purpose MATLAB FEM solver for transient reaction–diffusion in self-healing materials. By assembling global stiffness, capacity, and flux vectors with vectorized sparse routines and enforcing Dirichlet/Neumann conditions robustly, the code delivers high accuracy across our test cases:

- **Geometry and mesh independence:** applicable to any 2D polygonal domain—regular or irregular—with bilinear quadrilateral elements, and easily extended to complex crack geometries. The formulation supports any n-point Gauss quadrature for numerical integration, ensuring flexibility across various element geometries.
- **Performance:** vectorized, sparse assembly combined with MATLAB's backslash solver handles thousands of DOFs in seconds, enabling extensive parametric studies (domain length, reservoir size, volume-fraction dynamics).
- **Physics versatility:** trivially adapted to diffusion, heat conduction, or reaction–diffusion by replacing the primary field and coefficients (e.g. $C \to T$, $D \to k$, reaction rate $k_r \to$ generic source term), with full support for arbitrary spatial variation of D, k, k_r and even stiffness K for analogous problems.

Our five test cases quantified the effects of domain length, initial reservoir, volume-fraction dynamics, temperature-dependent and spatially varying diffusivity, and reaction kinetics on crack-filling behavior. The solver consistently captured theoretical scaling (Arrhenius sensitivity, diffusion time scales $\propto L^2/D$) and maintained stability under -method time integration.

14 Future Scopes

- Incorporate unsteady (time-varying) Dirichlet and Neumann boundary conditions for pulsed or cyclic healing injections.
- Extend to multi-species, nonlinear reaction–diffusion systems with coupled kinetics.
- Generalize to three-dimensional tetrahedral meshes and deploy hp-refinement for volumetric crack networks.
- Couple diffusion–reaction with structural mechanics to simulate healing under load and anisotropic behavior.
- Implement adaptive time stepping and error control to optimize transient accuracy and efficiency.
- Parallelize sparse assembly and solver routines (e.g. via MATLAB's Parallel Toolbox or GPU) for large-scale, high-fidelity simulations.

This flexible solver framework is thus readily extensible beyond concentration-crack applications—to thermal, electrochemical, and broader multi-physics simulations in advanced materials engineering.

15 References:

1 Dallaev, R. (2024). Advances in materials with self-healing properties: A brief review. Materials, 17(10), 2464. https://doi.org/10.3390/ma17102464

4 Specht, S. (2017). Multiphase Continuum Mechanical Modeling and Numerical Simulation of Self-Healing Polymers and Polymeric Composite Systems (Doctoral dissertation, Universität Duisburg-Essen, Germany)

3 Elrasoul, E. A., Haniffah, M. R. M. (2020). 1d modelling of healing agent in self-healing concrete using finite element method. Nature and Science, 18(6), 49–56. https://doi.org/10.7537/marsnsj180620.08

4 Yang, S. (2022). Phase-Field Modeling for Self-Healing of Mineral-Based Materials (Doctoral dissertation, Technische Universität Darmstadt). TUprints. https://tuprints.ulb.tu-darmstadt.de/22042/

5 Reddy, J. N. 2006. An Introduction to the Finite Element Method. 3rd ed. New York: McGraw-Hill.