Derivation and Numerical Solution of the Heat Conduction Equation with Detailed Code Explanation

Contents

1 Introduction

This document presents a comprehensive derivation of the heat conduction equation starting from the integral forms of the conservation laws, formation of the weak form of the partial differential equation (PDE) including boundary conditions, and implementation of the numerical solution using FEniCS with detailed line-by-line explanations of the code.

2 Derivation of the Heat Conduction Equation

2.1 Integral Conservation Laws

The conservation laws of mass, momentum, and energy are fundamental principles governing the behavior of physical systems. They can be expressed in integral form over a control volume V bounded by a closed surface S.

Material derivative :

$$\frac{D}{Dt}(*) = \frac{\partial}{\partial t}(*) + \vec{v} \cdot \nabla(*) \tag{1}$$

The material derivative is used to describe time rates of change for a given particle. The first term on RHS represents the self change, and the second term represents the flow/flux.

2.1.1 Conservation of Mass

The integral form of the conservation of mass is:

$$\frac{D}{Dt}(\int_V \rho \, dV) = 0 \tag{2}$$

$$\implies \frac{\partial}{\partial t} \int_V \rho \, dV + \int_S \rho \vec{v} \cdot \vec{n} \, dS = 0 \tag{3}$$

where:

- ρ is the density.
- $\vec{v}$ is the velocity vector.
- $\vec{n}$ is the outward-pointing unit normal vector on the surface S.

2.1.2 Conservation of Momentum

The integral form of the conservation of momentum is:

$$\frac{D}{Dt}(\int_V \rho \vec{v} \, dV) = \sum F_{\text{ext}} \tag{4}$$

$$\implies \frac{\partial}{\partial t} \int_V \rho \vec{v} \, dV + \int_S \rho \vec{v} (\vec{v} \cdot \vec{n}) \, dS = \int_S \vec{n} \cdot \vec{\sigma} \, dS + \int_V \rho \vec{g} \, dV \tag{5}$$

where:

- $\vec{\sigma}$ is the stress tensor.
- $\vec{g}$ is the body force per unit mass (e.g., gravity).

2.1.3 Conservation of Energy

The integral form of the conservation of energy is:

$$\frac{D}{Dt}(\int_V \rho e \, dV) = \dot{Q}_{\text{in}} + \dot{W}_{\text{in}} \tag{6}$$

$$\implies \frac{\partial}{\partial t}\int_V \rho e \, dV + \int_S \rho e \vec{v} \cdot \vec{n} \, dS = -\int_S \vec{q} \cdot \vec{n} \, dS + \int_V \rho \dot{q} \, dV + \int_S (\vec{\sigma} \cdot \vec{v}) \cdot \vec{n} \, dS \tag{7}$$

where:

- e is the internal energy per unit mass.
- $\vec{q}$ is the heat flux vector.
- $\rho\dot{q}$ represents volumetric heat sources.

2.2 Derivation of Differential Forms

To obtain the differential forms of the conservation equations, we apply the divergence theorem and consider the control volume to be fixed in space.

2.2.1 Conservation of Mass

Starting with the integral form:

$$\frac{\partial}{\partial t}\int_V \rho \, dV + \int_S \rho \vec{v} \cdot \vec{n} \, dS = 0 \tag{8}$$

Applying the divergence theorem:

$$\int_V \frac{\partial \rho}{\partial t} \, dV + \int_V \nabla \cdot (\rho \vec{v}) \, dV = 0 \tag{9}$$

Since the control volume V is arbitrary, the integrand must be zero:

$$\frac{\partial \rho}{\partial t} + \nabla \cdot (\rho \vec{v}) = 0 \tag{10}$$

2.2.2 Conservation of Momentum

Starting with the integral form:

$$\frac{\partial}{\partial t}\int_V \rho \vec{v} \, dV + \int_S \rho \vec{v} (\vec{v} \cdot \vec{n}) \, dS = \int_S \vec{\sigma} \cdot \vec{n} \, dS + \int_V \rho \vec{g} \, dV \tag{11}$$

Applying the divergence theorem:

$$\int_V \frac{\partial}{\partial t}(\rho \vec{v}) \, dV + \int_V \nabla \cdot (\rho \vec{v} \otimes \vec{v}) \, dV = \int_V \nabla \cdot \vec{\sigma} \, dV + \int_V \rho \vec{g} \, dV \tag{12}$$

Simplifying:

$$\frac{\partial}{\partial t}(\rho \vec{v}) + \nabla \cdot (\rho \vec{v} \otimes \vec{v}) = \nabla \cdot \vec{\sigma} + \rho \vec{g} \tag{13}$$

2.2.3 Conservation of Energy

Starting with the integral form:

$$\frac{\partial}{\partial t}\int_V \rho e\, dV + \int_S \rho e\vec{v}\cdot\vec{n}\, dS = -\int_S \vec{q}\cdot\vec{n}\, dS + \int_V \rho\dot{q}\, dV + \int_S \vec{n}\cdot(\vec{\sigma}\cdot\vec{v})\, dS \tag{14}$$

Applying the divergence theorem:

$$\int_V \frac{\partial}{\partial t}(\rho e)\, dV + \int_V \nabla\cdot(\rho e\vec{v})\, dV = -\int_V \nabla\cdot\vec{q}\, dV + \int_V \rho\dot{q}\, dV + \int_V \nabla\cdot(\vec{\sigma}\cdot\vec{v})\, dV \tag{15}$$

Simplifying:

$$\frac{\partial}{\partial t}(\rho e) + \nabla\cdot(\rho e\vec{v}) = -\nabla\cdot\vec{q} + \rho\dot{q} + \nabla\cdot(\vec{\sigma}\cdot\vec{v}) \tag{16}$$

2.3 Simplifying Under Assumptions

We make the following assumptions to simplify the equations:

1. **Stationary Medium:** $\vec{v} = 0$
2. **Constant Properties:** ρ, c, k are constants
3. **No Internal Heat Generation:** $\dot{q} = 0$
4. **Negligible Viscous Dissipation:** $\nabla\cdot(\vec{\sigma}\cdot\vec{v}) = 0$

2.3.1 Simplified Conservation of Mass

With $\vec{v} = 0$:

$$\frac{\partial\rho}{\partial t} = 0 \tag{17}$$

Since ρ is constant, this equation is satisfied.

2.3.2 Simplified Conservation of Momentum

With $\vec{v} = 0$:

$$\nabla\cdot\sigma = \rho\vec{g} \tag{18}$$

This is the equilibrium equation. Typically, we assume the body forces is negligible ($\rho\vec{g} = 0$).

2.3.3 Simplified Conservation of Energy

With $\vec{v} = 0$ and $\dot{q} = 0$:

$$\frac{\partial}{\partial t}(\rho e) = -\nabla\cdot\vec{q} \tag{19}$$

2.4 Relating Internal Energy to Temperature

Assuming the internal energy per unit mass e is related to temperature T by:

$$e = cT \tag{20}$$

where c is the specific heat capacity at constant volume.
Differentiating with respect to time:

$$\frac{\partial}{\partial t}(\rho e) = \rho c \frac{\partial T}{\partial t} \tag{21}$$

2.5 Fourier's Law of Heat Conduction

Fourier's law relates the heat flux $\vec{q}$ to the temperature gradient:

$$\vec{q} = -k\nabla T \tag{22}$$

where k is the thermal conductivity.
Substituting Fourier's law into the simplified energy equation:

$$\rho c \frac{\partial T}{\partial t} = -\nabla \cdot (-k\nabla T) \tag{23}$$

Simplifying:

$$\rho c \frac{\partial T}{\partial t} = \nabla \cdot (k\nabla T) \tag{24}$$

Assuming k is constant:

$$\rho c \frac{\partial T}{\partial t} = k\nabla^2 T \tag{25}$$

2.6 Defining Thermal Diffusivity

Thermal diffusivity α is defined as:

$$\alpha = \frac{k}{\rho c} \tag{26}$$

Substituting α into the equation:

$$\frac{\partial T}{\partial t} = \alpha \nabla^2 T \tag{27}$$

This is the **heat conduction equation**.

2.7 Final Heat Conduction Equation

The heat conduction equation in terms of temperature T is:

$$\frac{\partial T}{\partial t} = \alpha \nabla^2 T \tag{28}$$

This partial differential equation describes how temperature changes with time due to heat conduction in a stationary medium with constant properties.

3 Weak Formulation of the Heat Conduction Equation

3.1 Strong Form of the PDE

The heat conduction equation (strong form) is:

$$\frac{\partial T}{\partial t} = \alpha \nabla^2 T \quad \text{in} \quad \Omega \tag{29}$$

with boundary conditions:

$$T = \bar{T} \quad \text{on} \quad \partial\Omega_T \quad \text{(Dirichlet condition)} \tag{30}$$

$$-k\frac{\partial T}{\partial n} = \bar{q} \quad \text{on} \quad \partial\Omega_q \quad \text{(Neumann condition)} \tag{31}$$

3.2 Formation of the Weak Form

To derive the weak form, we multiply both sides of the PDE by a test function $w \in V_0$ (space of admissible test functions) and integrate over the domain Ω:

$$\int_\Omega w\frac{\partial T}{\partial t}\, dV = \alpha \int_\Omega w\nabla^2 T\, dV \tag{32}$$

3.2.1 Integration by Parts

We apply integration by parts to the right-hand side to reduce the order of differentiation on T:

$$\int_\Omega w\nabla^2 T\, dV = -\int_\Omega \nabla w \cdot \nabla T\, dV + \int_{\partial\Omega} w\frac{\partial T}{\partial n}\, dS \tag{33}$$

Substituting back:

$$\int_\Omega w\frac{\partial T}{\partial t}\, dV = -\alpha \int_\Omega \nabla w \cdot \nabla T\, dV + \alpha \int_{\partial\Omega} w\frac{\partial T}{\partial n}\, dS \tag{34}$$

3.2.2 Incorporating Boundary Conditions

Since $w = 0$ on $\partial\Omega_T$ (Dirichlet boundary), the boundary integral reduces to:

$$\alpha \int_{\partial\Omega_q} w\frac{\partial T}{\partial n}\, dS \tag{35}$$

Using the Neumann boundary condition $-k\frac{\partial T}{\partial n} = \bar{q}$:

$$\frac{\partial T}{\partial n} = -\frac{\bar{q}}{k} \tag{36}$$

Substituting back:

$$\alpha \int_{\partial\Omega_q} w\left(-\frac{\bar{q}}{k}\right) dS = -\frac{\alpha}{k}\int_{\partial\Omega_q} w\bar{q}\, dS \tag{37}$$

Since $\alpha = \frac{k}{\rho c}$, we have:

$$-\frac{\alpha}{k} = -\frac{1}{\rho c} \tag{38}$$

Therefore, the boundary term becomes:

$$-\frac{1}{\rho c} \int_{\partial\Omega_q} w\bar{q}\, dS \tag{39}$$

3.2.3 Final Weak Formulation

Collecting terms, the weak form is:

$$\int_\Omega w\rho c \frac{\partial T}{\partial t}\, dV + \int_\Omega k\nabla w \cdot \nabla T\, dV = \int_{\partial\Omega_q} w\bar{q}\, dS \tag{40}$$

3.3 Function Spaces

- **Trial Function Space** V:

$$V = \{T \in H^1(\Omega) \,|\, T = \bar{T} \text{ on } \partial\Omega_T\} \tag{41}$$

- **Test Function Space** V_0:

$$V_0 = \{w \in H^1(\Omega) \,|\, w = 0 \text{ on } \partial\Omega_T\} \tag{42}$$

3.4 Summary of the Weak Form

Find $T \in V$ such that:

$$\int_\Omega w\rho c \frac{\partial T}{\partial t}\, dV + \int_\Omega k\nabla w \cdot \nabla T\, dV = \int_{\partial\Omega_q} w\bar{q}\, dS \quad \forall w \in V_0 \tag{43}$$

This weak form incorporates the boundary conditions and is suitable for numerical methods such as the finite element method.

4 Numerical Solution Using FEniCS

We will solve the heat conduction equation using FEniCS for a cubic domain of size $10 \times 10 \times 10$ units with the following boundary conditions:

- **Bottom Face (z = 0):** $T = 300\,\mathrm{K}$
- **Top Face (z = 10):** A moving Gaussian heat source modeled as a Neumann boundary condition.
- **Other Boundaries:** Insulated (zero heat flux)

4.1 FEniCS Code with Line-by-Line Explanations

Below is the FEniCS code with detailed explanations for each part.

```
from dolfin import *
import numpy as np

# Material properties
rho = 7800.0        # Density (kg/m^3)
c = 500.0           # Specific heat capacity (J/(kg*K))
k = 45.0            # Thermal conductivity (W/(m*K))
alpha = k / (rho * c)  # Thermal diffusivity

# Create a mesh of the domain (a cube from (0,0,0) to (10,10,10))
mesh = BoxMesh(Point(0, 0, 0), Point(10, 10, 10), 20, 20, 20)

# Define the function space (continuous Lagrange elements of degree 1)
V = FunctionSpace(mesh, 'P', 1)

# Define boundary condition for the bottom face (Dirichlet condition)
def bottom(x, on_boundary):
    return near(x[2], 0.0) and on_boundary

T_bottom = Constant(300.0)             # Fixed temperature at the bottom
bc_bottom = DirichletBC(V, T_bottom, bottom)  # Apply the boundary
   condition

# Parameters for the moving Gaussian heat source
q0 = 1e5            # Peak heat flux (W/m^2)
v_x = 0.5           # Velocity in x-direction (m/s)
v_y = 0.0           # Velocity in y-direction (m/s)
sigma = 1.0         # Standard deviation (controls the spread)

# Initial position of the heat source
x0_0 = 0.0          # Initial x-position
y0_0 = 5.0          # Initial y-position

# Expressions for the moving source position
x0 = Expression('x0_0 + v_x*t', degree=1, x0_0=x0_0, v_x=v_x, t=0.0)
y0 = Expression('y0_0 + v_y*t', degree=1, y0_0=y0_0, v_y=v_y, t=0.0)

# Heat flux expression as a user-defined function
class HeatFluxExpression(UserExpression):
    def __init__(self, x0, y0, sigma, t, **kwargs):
        super().__init__(**kwargs)
        self.x0 = x0        # x-position expression of the heat source
        self.y0 = y0        # y-position expression of the heat source
        self.sigma = sigma  # Standard deviation of the Gaussian
        self.t = t          # Current time

    def eval(self, values, x):
        x0_val = self.x0(self.t)  # Evaluate x-position at current time
        y0_val = self.y0(self.t)  # Evaluate y-position at current time
        exponent = -((x[0] - x0_val)**2 + (x[1] - y0_val)**2) / (2 * self.
   sigma**2)
```

```
        values[0] = q0 * np.exp(exponent)  # Compute the heat flux value

    def value_shape(self):
        return ()

q_flux = HeatFluxExpression(x0, y0, sigma, t=0.0, degree=2)  # Instantiate the heat flux expression

# Initial temperature distribution
T_initial = Constant(300.0)     # Initial temperature of the domain
T_n = interpolate(T_initial, V) # Project initial temperature onto the function space

# Define the unknown and test functions
T = Function(V)          # Temperature at the current time step (unknown)
v = TestFunction(V)      # Test function used in variational formulation
dt = 0.1                 # Time step size

# Define boundary measure for the top surface where the heat flux is applied
boundary_markers = MeshFunction('size_t', mesh, mesh.topology().dim() - 1, 0)
top = AutoSubDomain(lambda x, on_boundary: near(x[2], 10.0) and on_boundary)
top.mark(boundary_markers, 1)  # Mark the top boundary with marker '1'
ds = Measure('ds', domain=mesh, subdomain_data=boundary_markers)  # Redefine the measure 'ds' with subdomains

# Weak form of the heat conduction equation
F = (T - T_n) / dt * v * dx + k * dot(grad(T), grad(v)) * dx  - q_flux * v * ds(1)
J = derivative(F, T)
# Time-stepping parameters
t = 0.0                          # Initial time
T_end = 10.0                     # Final time
num_steps = int(T_end / dt)      # Number of time steps

# Create VTK file for saving the solution (for visualization)
vtkfile = File('heat_conduction/temperature.pvd')

# Time-stepping loop
for n in range(num_steps):
    # Update current time
    t += dt
    print(f'Time step {n+1}/{num_steps}: t = {t:.2f}')

    # Update moving heat source position and heat flux expression
    x0.t = t           # Update time in x-position expression
    y0.t = t           # Update time in y-position expression
    q_flux.t = t       # Update time in heat flux expression

    # Solve the variational problem
    solve(F == 0, T, bc_bottom, J=J)

```

```
97      # Save the solution to file
98      vtkfile << (T, t)
99
100     # Update previous solution for next time step
101     T_n.assign(T)
```

Listing 1: FEniCS Code for Heat Conduction with Line-by-Line Explanation

4.2 Explanation of the Code

4.2.1 Importing Modules

Lines 1-2:

- `from dolfin import *`: Imports all necessary classes and functions from the FEniCS library.
- `import numpy as np`: Imports NumPy for numerical operations, such as exponentials.

4.2.2 Defining Material Properties

Lines 5-8:

- `rho = 7800.0`: Sets the density ρ to 7800 kg/m^3 (typical for steel).
- `c = 500.0`: Sets the specific heat capacity c to 500 J/(kg·K).
- `k = 45.0`: Sets the thermal conductivity k to 45 W/(m·K).
- `alpha = k / (rho * c)`: Calculates the thermal diffusivity α.

4.2.3 Creating the Mesh and Function Space

Lines 11-14:

- `mesh = BoxMesh(Point(0, 0, 0), Point(10, 10, 10), 20, 20, 20)`: Creates a 3D mesh of the cube domain from (0,0,0) to (10,10,10) with 20 divisions along each axis.
- `V = FunctionSpace(mesh, 'P', 1)`: Defines the function space V using continuous Lagrange (polynomial) elements of degree 1.

4.2.4 Defining Boundary Conditions

Lines 16-22:

- `def bottom(x, on_boundary)`: Defines a function to identify points on the bottom boundary.
- `return near(x[2], 0.0) and on_boundary`: Returns `True` if the point is on the bottom face ($z = 0$) and on the boundary.
- `T_bottom = Constant(300.0)`: Sets the fixed temperature at the bottom to 300 K.
- `bc_bottom = DirichletBC(V, T_bottom, bottom)`: Applies the Dirichlet boundary condition on the bottom face.

4.2.5 Defining the Moving Gaussian Heat Source

Lines 24-31:

- `q0 = 1e5`: Sets the peak heat flux q_0 to 1×10^6 W/m^2.
- `v_x = 0.5`, `v_y = 0.0`: Sets the velocities of the heat source in the x and y directions.
- `sigma = 1.0`: Sets the standard deviation of the Gaussian function.
- `x0_0 = 0.0`, `y0_0 = 5.0`: Sets the initial position of the heat source.
- `x0` and `y0`: Define expressions for the time-dependent positions of the heat source in x and y directions.

4.2.6 Defining the Heat Flux Expression

Lines 33-55:

- `class HeatFluxExpression(UserExpression)`: Defines a custom user expression for the heat flux.
- `__init__`: Initializes the expression with the position expressions, standard deviation, and time.
- `def eval(self, values, x)`: Evaluates the heat flux at a given point x.
  - `x0_val = self.x0(self.t)`: Computes the current x-position of the heat source.
  - `y0_val = self.y0(self.t)`: Computes the current y-position of the heat source.
  - `exponent`: Calculates the exponent of the Gaussian function based on the distance from the heat source center.
  - `values[0] = q0 * np.exp(exponent)`: Computes the heat flux value at point x.
- `def value_shape(self)`: Returns an empty tuple indicating a scalar output.
- `q_flux = HeatFluxExpression(x0, y0, sigma, t=0.0, degree=2)`: Instantiates the heat flux expression.

4.2.7 Setting the Initial Condition

Lines 57-59:

- `T_initial = Constant(300.0)`: Sets the initial temperature of the domain to 300 K.
- `T_n = interpolate(T_initial, V)`: Projects the initial temperature onto the function space V.

4.2.8 Defining the Variational Problem

Lines 61-64:

- `T = Function(V)`: Defines the unknown temperature function at the current time step.
- `v = TestFunction(V)`: Defines the test function for the variational formulation.
- `dt = 0.1`: Sets the time step size.

4.2.9 Boundary Measure for the Top Surface

Lines 66-70:

- `boundary_markers = MeshFunction('size_t', mesh, mesh.topology().dim() - 1, 0)`: Creates a mesh function to mark boundaries.
- `top = AutoSubDomain(lambda x, on_boundary: near(x[2], 10.0) and on_boundary)`: Defines the top boundary where $z = 10$.
- `top.mark(boundary_markers, 1)`: Marks the top boundary with the marker '1'.
- `ds = Measure('ds', domain=mesh, subdomain_data=boundary_markers)`: Redefines the measure 'ds' to include subdomain data for boundary integration.

4.2.10 Formulating the Weak Form

Lines 72-73:

- The weak form F is defined as:

$$F = \left(\frac{T - T_n}{dt} \right) v \, dx + k \nabla T \cdot \nabla v \, dx - q_{\text{flux}} \, v \, ds(1)$$

- `(T - T_n) / dt * v * dx`: Represents the time derivative term.
- `k * dot(grad(T), grad(v)) * dx`: Represents the diffusion term.
- `- q_flux * v * ds(1)`: Represents the heat flux term on the top boundary (with marker '1').

4.2.11 Time-Stepping Parameters

Lines 75-78:

- `t = 0.0`: Initializes the time variable.
- `T_end = 10.0`: Sets the end time for the simulation.
- `num_steps = int(T_end / dt)`: Calculates the number of time steps.

4.2.12 Output File for Results

Line 80-81:

- `vtkfile = File('heat_conduction/temperature.pvd')`: Creates a VTK file for saving the temperature results, which can be visualized using ParaView.

4.2.13 Time-Stepping Loop

Lines 83-101:

- `for n in range(num_steps)`: Starts the time-stepping loop.
- `t += dt`: Updates the current time.
- `print(f'Time step {n+1}/{num_steps}:  t = {t:.2f}')`: Prints the current time step information.
- `x0.t = t`, `y0.t = t`, `q_flux.t = t`: Updates the time parameter in the position expressions and heat flux expression.
- `solve(F == 0, T, bc_bottom)`: Solves the variational problem $F = 0$ for T with the boundary condition.
- `vtkfile << (T, t)`: Saves the current solution T and time t to the VTK file.
- `T_n.assign(T)`: Updates the previous solution T_n for the next time step.

5 Conclusion

Starting from the integral forms of the conservation laws, we derived the heat conduction equation and formulated its weak form suitable for finite element analysis. We then implemented a numerical solution using FEniCS, incorporating boundary conditions and a moving Gaussian heat source. The detailed derivations and line-by-line explanation of the code provide insights into setting up and solving PDEs using FEniCS.

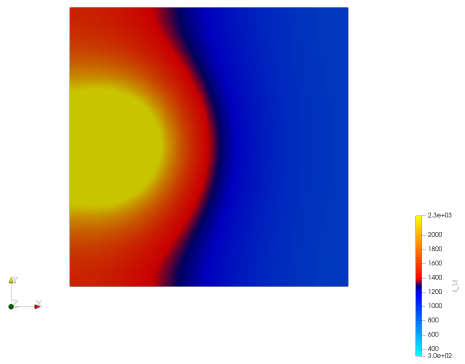

Figure 1: Temperature at t=3

6 References

- *Fundamentals of Fluid Mechanics* by BRUCE et al.
- *Fundamentals of Heat and Mass Transfer* by Bergman et al.
- *FEniCS Project Documentation*: `https://fenicsproject.org/`