Introduction to 1D Time-Dependent Heat Conduction

Yazhuo Liu

September 19, 2024

Contents

1 Introduction

Heat conduction is a fundamental physical process describing the transfer of thermal energy within a material without the movement of the material itself. In one dimension (1D), heat conduction can be modeled to understand temperature distribution along a rod or similar structures. This document covers the derivation of the time-dependent 1D heat conduction equation incorporating energy balance, boundary conditions (Neumann and Dirichlet), the weak form suitable for finite element analysis, and an implementation using FEniCS with detailed explanations.

2 Derivation of the Time-Dependent 1D Heat Conduction Equation

2.1 Physical Principles

The 1D heat conduction process is governed by two main principles:

1. **Fourier's Law**: Relates the heat flux $q(x)$ to the temperature gradient.

$$q(x) = -k\frac{dT}{dx}$$

   - $q(x)$: Heat flux ($\mathrm{W/m^2}$)
   - k: Thermal conductivity (W/m·K)
   - T: Temperature (K)
   - x: Spatial coordinate (m)

2. **Conservation of Energy**: The rate of change of thermal energy within a differential volume equals the net heat flux into the volume plus any internal heat generation Q.

$$\rho c_p \frac{\partial T}{\partial t}\Delta x = q_{\text{in}} - q_{\text{out}} + Q\Delta x$$

   - ρ: Density ($\mathrm{kg/m^3}$)
   - c_p: Specific heat capacity (J/kg·K)
   - $\frac{\partial T}{\partial t}$: Temporal temperature change (K/s)
   - Δx: Length of the differential element (m)
   - Q: Internal heat generation per unit volume ($\mathrm{W/m^3}$)

2.2 Energy Balance Derivation

Consider a small segment of the rod between x and $x + \Delta x$. The energy balance can be expressed as:

2.2.1 Heat Entering at x

$$q(x) = -k\frac{dT}{dx}\bigg|_x$$

2.2.2 Heat Leaving at $x + \Delta x$

$$q(x + \Delta x) = -k \frac{dT}{dx}\bigg|_{x+\Delta x}$$

2.2.3 Net Heat Flux into the Element

$$q_{\text{in}} - q_{\text{out}} = -k \left(\frac{dT}{dx}\bigg|_{x+\Delta x} - \frac{dT}{dx}\bigg|_{x} \right)$$

Using a Taylor series expansion for small Δx:

$$\frac{dT}{dx}\bigg|_{x+\Delta x} \approx \frac{dT}{dx}\bigg|_{x} + \Delta x \frac{d^2T}{dx^2}\bigg|_{x}$$

Thus,

$$q_{\text{in}} - q_{\text{out}} \approx -k \left(\frac{d^2T}{dx^2} \Delta x \right)$$

2.2.4 Accumulation of Heat

The rate of accumulation of heat within the element is given by:

$$\rho c_p \frac{\partial T}{\partial t} \Delta x$$

2.3 Combining Heat Flux and Heat Accumulation

From the energy balance:

$$\rho c_p \frac{\partial T}{\partial t} \Delta x = q_{\text{in}} - q_{\text{out}} + Q \Delta x$$

Substituting the net heat flux:

$$\rho c_p \frac{\partial T}{\partial t} \Delta x = -k \frac{d^2T}{dx^2} \Delta x + Q \Delta x$$

Dividing both sides by Δx (assuming $\Delta x \neq 0$):

$$\rho c_p \frac{\partial T}{\partial t} = -k \frac{d^2T}{dx^2} + Q$$

Rearranging:

$$\frac{\partial T}{\partial t} = \alpha \frac{\partial^2 T}{\partial x^2} + \frac{Q}{\rho c_p}$$

where $\alpha = \frac{k}{\rho c_p}$ is the thermal diffusivity.

This is the **Time-Dependent 1D Heat Conduction Equation**:

$$\frac{\partial T}{\partial t} - \alpha \frac{\partial^2 T}{\partial x^2} = \frac{Q}{\rho c_p}$$

3 Boundary Conditions

To solve the heat conduction equation, appropriate boundary conditions must be specified. In this context, we consider:

- **Neumann Boundary Condition** at $x = 0$: Specifies the heat flux.
- **Dirichlet Boundary Condition** at $x = L$: Specifies the temperature.

3.1 Neumann Boundary Condition at $x = 0$

A Neumann boundary condition specifies the heat flux at the boundary.
Example:

$$-q\bigg|_{x=0} = q_0$$

where:

$$-k\frac{dT}{dx}\bigg|_{x=0} = q_0$$

q_0 is the prescribed heat flux at $x = 0$.

3.2 Dirichlet Boundary Condition at $x = L$

A Dirichlet boundary condition specifies the temperature at the boundary.
Example:

$$T(L,t) = T_0$$

where: T_0 is the prescribed temperature at $x = L$.

4 Weak Form Derivation for the Time-Dependent Case

To apply the Finite Element Method (FEM) using FEniCS for the time-dependent heat conduction problem with specified boundary conditions, we derive the weak (variational) form of the equation incorporating the energy balance.

4.1 Starting with the Strong Form

The time-dependent 1D heat conduction equation is:

$$\frac{\partial T}{\partial t} - \alpha\frac{\partial^2 T}{\partial x^2} = \frac{Q}{\rho c_p} \quad \text{in } \Omega \times (0, T]$$

where $\Omega = [0, L]$ is the spatial domain and T is the final time.

4.2 Energy Balance Integration

Multiply the strong form by a test function v (which vanishes on Dirichlet boundaries) and integrate over the domain:

$$\int_\Omega \left(\frac{\partial T}{\partial t} - \alpha\frac{\partial^2 T}{\partial x^2}\right) v\,dx = \int_\Omega \frac{Q}{\rho c_p} v\,dx$$

4.3 Applying Integration by Parts

Focus on the diffusion term:

$$\int_\Omega -\alpha \frac{\partial^2 T}{\partial x^2} v \, dx$$

Apply integration by parts to lower the derivative order:

$$\int_\Omega -\alpha \frac{\partial^2 T}{\partial x^2} v \, dx = -\alpha \frac{\partial T}{\partial x} v \Big|_0^L + \int_\Omega \alpha \frac{\partial T}{\partial x} \frac{\partial v}{\partial x} \, dx$$

4.3.1 Boundary Terms

- **Neumann Boundary Condition at** $x = 0$:

$$-\alpha \frac{\partial T}{\partial x} \Big|_{x=0} = q_0$$

- **Dirichlet Boundary Condition at** $x = L$:

$$T(L, t) = T_0$$

  Since $v(L) = 0$ (test function vanishes on Dirichlet boundaries), the boundary term at $x = L$ disappears.

Thus, the boundary terms reduce to:

$$-\alpha \frac{\partial T}{\partial x} \Big|_{x=0} v(0) = q_0 v(0)$$

4.4 Incorporating Boundary Conditions into the Weak Form

Substituting the boundary condition at $x = 0$ into the weak form:

$$\int_\Omega \frac{\partial T}{\partial t} v \, dx + \int_\Omega \alpha \frac{\partial T}{\partial x} \frac{\partial v}{\partial x} \, dx = \int_\Omega \frac{Q}{\rho c_p} v \, dx + q_0 v(0)$$

4.5 Time Discretization

Using the **Backward Euler Method**, an implicit time-stepping scheme:

Let T^n denote the temperature at the previous time step and T^{n+1} at the current time step. The time derivative is approximated as:

$$\frac{\partial T}{\partial t} \approx \frac{T^{n+1} - T^n}{\Delta t}$$

where Δt is the time step size.

Substituting into the weak form:

$$\int_\Omega \frac{T^{n+1} - T^n}{\Delta t} v \, dx + \int_\Omega \alpha \frac{\partial T^{n+1}}{\partial x} \frac{\partial v}{\partial x} \, dx = \int_\Omega \frac{Q}{\rho c_p} v \, dx + q_0 v(0)$$

4.6 Rearranging Terms

Bring all known terms to the right-hand side:

$$\int_\Omega \frac{T^{n+1}}{\Delta t} v\, dx + \int_\Omega \alpha \frac{\partial T^{n+1}}{\partial x}\frac{\partial v}{\partial x}\, dx = \int_\Omega \frac{T^n}{\Delta t} v\, dx + \int_\Omega \frac{Q}{\rho c_p} v\, dx + q_0 v(0)$$

This is the **Weak Form** of the time-dependent 1D heat conduction equation using the Backward Euler time discretization with a Neumann boundary condition at $x = 0$ and a Dirichlet boundary condition at $x = L$.

5 FEniCS Implementation for Time-Dependent 1D Heat Conduction

FEniCS is an open-source computing platform for solving partial differential equations (PDEs) using the Finite Element Method (FEM). Below is a FEniCS code example for the time-dependent 1D heat conduction problem with a Neumann boundary condition at $x = 0$ and a Dirichlet boundary condition at $x = L$, along with a detailed line-by-line explanation.

5.1 FEniCS Code

```
from fenics import *
import numpy as np
import matplotlib.pyplot as plt

# Suppress FEniCS log output by setting the log level to 'ERROR'
set_log_level(40)  # 40=ERROR

# Alternatively, to completely deactivate logging, uncomment the following line:
# set_log_active(False)

# Define physical parameters
alpha = 1.0                 # Thermal diffusivity (W/m*K)
rho = 1.0                   # Density (kg/m^3)
cp = 1.0                    # Specific heat capacity (J/kg*K)
Q = Constant(0.0)           # Internal heat generation (W/m^3)

# Define domain parameters
L = 1.0                     # Length of the rod (m)
nx = 50                     # Number of finite elements

# Define boundary conditions
q0 = Constant(100.0)        # Prescribed heat flux at x=0 (W/m^2)
T0 = Constant(300.0)        # Prescribed temperature at x=L (K)

# Time-stepping parameters
T_final = 1.0               # Final time (s)
```

```
num_steps = 50               # Number of time steps
dt = T_final / num_steps    # Time step size (s)

# Create mesh and define function space
mesh = IntervalMesh(nx, 0, L)
V = FunctionSpace(mesh, 'P', 1)

# Define boundary identification functions
def boundary_neumann(x, on_boundary):
    return on_boundary and near(x[0], 0.0)

def boundary_dirichlet(x, on_boundary):
    return on_boundary and near(x[0], L)

# Define boundary condition for Dirichlet at x=L
bc_dirichlet = DirichletBC(V, T0, boundary_dirichlet)

# Define initial condition
T_n = Function(V)
T_n.assign(Constant(300.0))  # Initial temperature distribution

# Define trial and test functions
T = TrialFunction(V)
v = TestFunction(V)

# Define measures for boundary integration
boundary_markers = MeshFunction('size_t', mesh, mesh.topology().dim()-1,
    0)
neumann_marker = 1
dirichlet_marker = 2

class NeumannBoundary(SubDomain):
    def inside(self, x, on_boundary):
        return on_boundary and near(x[0], 0.0)

class DirichletBoundary(SubDomain):
    def inside(self, x, on_boundary):
        return on_boundary and near(x[0], L)

# Mark boundaries
NeumannBoundary().mark(boundary_markers, neumann_marker)
DirichletBoundary().mark(boundary_markers, dirichlet_marker)

# Define measures with boundary markers
ds = Measure('ds', domain=mesh, subdomain_data=boundary_markers)

# Define the variational problem
a = (rho * cp / dt) * T * v * dx + alpha * dot(grad(T), grad(v)) * dx
L_form = (rho * cp / dt) * T_n * v * dx + Q * v * dx + q0 * v * ds(
   neumann_marker)
```

```

# Create function to hold the solution
T_sol = Function(V)

# Time-stepping loop using high-level solve
time = 0.0
for n_step in range(num_steps):
    # Update current time
    time += dt

    # Solve the weak form directly
    solve(a == L_form, T_sol, bc_dirichlet)

    # Update previous solution
    T_n.assign(T_sol)

    # Plot solution at certain intervals
    if n_step % 10 == 0 or n_step == num_steps - 1:
        plt.plot(mesh.coordinates(), T_sol.compute_vertex_values(mesh),
           label=f'Time = {time:.2f}s')

# Finalize and display the plot
plt.xlabel('Position x (m)')
plt.ylabel('Temperature T (K)')
plt.title('Time-Dependent 1D Heat Conduction')
plt.legend()
plt.grid(True)
plt.show()
```

Listing 1: High-Level FEniCS Code for 1D Time-Dependent Heat Conduction with Log Suppression

5.2 Line-by-Line Explanation

5.2.1 Imports

```
from fenics import *
import numpy as np
import matplotlib.pyplot as plt
```

- **FEniCS**: Imports all necessary classes and functions from the FEniCS library for finite element computations.
- **NumPy**: Imports NumPy for numerical operations.
- **Matplotlib**: Imports Matplotlib for plotting the results.

5.2.2 Define Physical Parameters

```
# Define physical parameters
alpha = 1.0                  # Thermal diffusivity (W/m*K)
rho = 1.0                    # Density (kg/m^3)
cp = 1.0                     # Specific heat capacity (J/kg*K)
Q = Constant(0.0)            # Internal heat generation (W/m^3)
```

- **alpha**: Thermal diffusivity $\alpha = \frac{k}{\rho c_p}$.
- **rho**: Density ρ.
- **cp**: Specific heat capacity c_p.
- **Q**: Internal heat generation term. Set to zero for no internal heat sources.

5.2.3 Define Domain Parameters

```
# Define domain parameters
L = 1.0                      # Length of the rod (m)
nx = 50                      # Number of finite elements
```

- **L**: Length of the rod.
- **nx**: Number of finite elements for spatial discretization.

5.2.4 Define Boundary Conditions

```
# Define boundary conditions
q0 = Constant(100.0)         # Prescribed heat flux at x=0 (W/m^2)
T0 = Constant(300.0)         # Prescribed temperature at x=L (K)
```

- **q0**: Prescribed heat flux at $x = 0$ (Neumann boundary condition).
- **T0**: Prescribed temperature at $x = L$ (Dirichlet boundary condition).

5.2.5 Define Time-Stepping Parameters

```
# Time-stepping parameters
T_final = 1.0                # Final time (s)
num_steps = 50               # Number of time steps
dt = T_final / num_steps     # Time step size (s)
```

- **T_final**: Total simulation time.
- **num_steps**: Number of time steps.
- **dt**: Time step size, calculated as the total time divided by the number of steps.

5.2.6 Create Mesh and Define Function Space

```
# Create mesh and define function space
mesh = IntervalMesh(nx, 0, L)
V = FunctionSpace(mesh, 'P', 1)
```

- **mesh**: Creates a 1D mesh from $x = 0$ to $x = L$ with ‘nx‘ elements.
- **V**: Defines the function space using first-order (linear) Lagrange elements.

5.2.7 Define Boundary Identification Functions

```
# Define boundary identification functions
def boundary_neumann(x, on_boundary):
    return on_boundary and near(x[0], 0.0)

def boundary_dirichlet(x, on_boundary):
    return on_boundary and near(x[0], L)
```

- **boundary_neumann**: Identifies the Neumann boundary at $x = 0$.
- **boundary_dirichlet**: Identifies the Dirichlet boundary at $x = L$.

5.2.8 Define Dirichlet Boundary Condition

```
# Define boundary condition for Dirichlet at x=L
bc_dirichlet = DirichletBC(V, T0, boundary_dirichlet)
```

- **bc_dirichlet**: Applies the Dirichlet boundary condition $T = T_0$ at $x = L$.

5.2.9 Define Initial Condition

```
# Define initial condition
T_n = Function(V)
T_n.assign(Constant(300.0))  # Initial temperature distribution
```

- **T_n**: Represents the temperature at the previous time step.
- **assign**: Initializes the temperature distribution to 300 K across the entire domain.

5.2.10 Define Trial and Test Functions

```
# Define trial and test functions
T = TrialFunction(V)
v = TestFunction(V)
```

- **T**: Trial function representing the unknown temperature at the current time step.
- **v**: Test function used in the variational formulation.

5.2.11 Define Measures for Boundary Integration

```
# Define measures for boundary integration
boundary_markers = MeshFunction('size_t', mesh, mesh.topology().dim()-1,
    0)
neumann_marker = 1
dirichlet_marker = 2

class NeumannBoundary(SubDomain):
    def inside(self, x, on_boundary):
        return on_boundary and near(x[0], 0.0)

class DirichletBoundary(SubDomain):
    def inside(self, x, on_boundary):
        return on_boundary and near(x[0], L)

# Mark boundaries
NeumannBoundary().mark(boundary_markers, neumann_marker)
DirichletBoundary().mark(boundary_markers, dirichlet_marker)

# Define measures with boundary markers
ds = Measure('ds', domain=mesh, subdomain_data=boundary_markers)
```

- **boundary_markers**: Creates a mesh function to mark different parts of the boundary.
- **neumann_marker** and **dirichlet_marker**: Assign unique integer labels to the Neumann and Dirichlet boundaries, respectively.
- **SubDomain Classes**: Define subdomains for the Neumann and Dirichlet boundaries.
- **mark**: Marks the boundaries with the specified markers.
- **ds**: Defines a boundary measure that can be used to integrate over specific marked boundaries.

5.2.12 Define the Variational Problem

```
# Define the variational problem
a = (rho * cp / dt) * T * v * dx + alpha * dot(grad(T), grad(v)) * dx
L_form = (rho * cp / dt) * T_n * v * dx + Q * v * dx + q0 * v * ds(
    neumann_marker)
```

- **a**: Bilinear form representing the left-hand side of the weak form.
  - $(\rho c_p/\Delta t)Tv\,dx$: Time derivative term.
  - $\alpha\nabla T \cdot \nabla v\,dx$: Diffusion term.
- **L_form**: Linear form representing the right-hand side of the weak form.

- $(\rho c_p/\Delta t) T_n v\, dx$: Contribution from the previous time step.
- $Q v\, dx$: Internal heat generation.
- $q_0 v\, ds$(neumann_marker): Neumann boundary condition at $x = 0$.

5.2.13 Create Function to Hold the Solution

```
# Create function to hold the solution
T_sol = Function(V)
```

- **'T_sol = Function(V)'**: Initializes a FEniCS 'Function' in space 'V' to store the computed temperature distribution at the current time step.

5.2.14 Time-Stepping Loop

```
# Time-stepping loop using high-level solve
time = 0.0
for n_step in range(num_steps):
    # Update current time
    time += dt

    # Solve the weak form directly
    solve(a == L_form, T_sol, bc_dirichlet)

    # Update previous solution
    T_n.assign(T_sol)

    # Plot solution at certain intervals
    if n_step % 10 == 0 or n_step == num_steps - 1:
        plt.plot(mesh.coordinates(), T_sol.compute_vertex_values(mesh),
           label=f'Time = {time:.2f}s')
```

- **Initialization**:
  - **'time = 0.0'**: Initializes the simulation time to '0.0 seconds'.
- **Loop Over Time Steps**:
  - **'for n_step in range(num_steps):'**: Iterates over each time step from '0' to 'num_steps - 1'.
  - **Updating Time**:
    - **'time += dt'**: Increments the simulation time by the time step size 'dt'.
  - **Solving the Variational Problem**:
    - **'solve(a == L_form, T_sol, bc_dirichlet)'**: Directly solves the weak form equation by equating the bilinear form 'a' to the linear form 'L_form', storing the solution in 'T_sol' and applying the Dirichlet boundary condition 'bc_dirichlet'.

– **Updating Previous Solution**:

  * **'T_n.assign(T_sol)'**: Updates the temperature from the current time step to be used in the next iteration. This ensures that each time step uses the latest temperature distribution as the "previous" temperature.

– **Plotting the Solution**:

  * **'if n_step % 10 == 0 or n_step == num_steps - 1:'**: Checks if the current time step is a multiple of '10' or the final time step.
  * **'plt.plot(...)'**: Plots the temperature distribution along the rod at the current time step with a label indicating the simulation time.

5.2.15 Finalize and Display the Plot

```
# Finalize and display the plot
plt.xlabel('Position x (m)')
plt.ylabel('Temperature T (K)')
plt.title('Time-Dependent 1D Heat Conduction')
plt.legend()
plt.grid(True)
plt.show()
```

- **plt.xlabel**: Labels the x-axis as position in meters.
- **plt.ylabel**: Labels the y-axis as temperature in Kelvin.
- **plt.title**: Sets the plot title.
- **plt.legend**: Displays the legend showing different time steps.
- **plt.grid(True)**: Adds a grid for better readability.
- **plt.show()**: Renders and displays the plot.

6 Conclusion

This document provided a comprehensive introduction to time-dependent 1D heat conduction, including the derivation of the governing equation with energy balance, application of boundary conditions (Neumann at $x = 0$ and Dirichlet at $x = L$), formulation of the weak form for FEM, and implementation using FEniCS. By incorporating a Neumann boundary condition for heat flux and a Dirichlet boundary condition for temperature, the simulation accurately models scenarios where one end of the rod is subjected to a prescribed heat flux while the other end maintains a fixed temperature.

Understanding these concepts lays the foundation for analyzing more complex heat transfer problems in higher dimensions and with more intricate boundary conditions.

7 References

- **FEniCS Documentation**: https://fenicsproject.org/documentation/
- **Finite Element Method Theory**: Various textbooks and online resources provide in-depth coverage of FEM and weak form derivations.
- **Numerical Methods for PDEs**: Literature on numerical analysis for partial differential equations offers insights into time-stepping schemes and stability considerations.
- **Neumann Boundary Condition**: https://en.wikipedia.org/wiki/Boundary_condition#Neumann_boundary_condition
- **Dirichlet Boundary Condition**: https://en.wikipedia.org/wiki/Boundary_condition#Dirichlet_boundary_condition