

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

Yazhuo Liu

March 20, 2025

Contents

1	Introduction	3
2	Derivation of the Heat Conduction Equation	3
2.1	Integral Conservation Laws	3
2.1.1	Conservation of Mass	3
2.1.2	Conservation of Momentum	3
2.1.3	Conservation of Energy	4
2.2	Derivation of Differential Forms	4
2.2.1	Conservation of Mass	4
2.2.2	Conservation of Momentum	4
2.2.3	Conservation of Energy	5
2.3	Simplifying Under Assumptions	5
2.3.1	Simplified Conservation of Mass	5
2.3.2	Simplified Conservation of Momentum	5
2.3.3	Simplified Conservation of Energy	6
2.4	Relating Internal Energy to Temperature	6
2.5	Fourier's Law of Heat Conduction	6
2.6	Defining Thermal Diffusivity	6
2.7	Final Heat Conduction Equation	7
3	Weak Formulation of the Heat Conduction Equation	7
3.1	Strong Form of the PDE	7
3.2	Formation of the Weak Form	7
3.2.1	Integration by Parts	7
3.2.2	Incorporating Boundary Conditions	8
3.2.3	Final Weak Formulation	8
3.3	Function Spaces	8

4	LPBF Heat Sources	8
4.1	Classic Double Ellipsoid Laser Heat Source Model	8
4.1.1	Improvements for LPBF Process	9
4.2	Convective Heat Flux	9
4.3	Radiative Heat Flux	9
5	Numerical Solution Using FEniCS	10
5.1	FEniCS Code with Line-by-Line Explanations	10
6	Conclusion	13
7	References	14

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 :

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

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:

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

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

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:

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

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

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:

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

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

where:

- e is the internal energy per unit mass.
- \vec{q} is the heat flux vector.
- \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:

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

Applying the divergence theorem:

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

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

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

2.2.2 Conservation of Momentum

Starting with the integral form:

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

Applying the divergence theorem:

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

Simplifying:

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

2.2.3 Conservation of Energy

Starting with the integral form:

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

Applying the divergence theorem:

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

Simplifying:

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

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. **Negligible Viscous Dissipation:** $\nabla \cdot (\vec{\sigma} \cdot \vec{v}) = 0$

2.3.1 Simplified Conservation of Mass

With $\vec{v} = 0$:

$$\frac{\partial \rho}{\partial t} = 0 \quad (17)$$

Since ρ is constant, this equation is satisfied.

2.3.2 Simplified Conservation of Momentum

With $\vec{v} = 0$:

$$\nabla \cdot \sigma = \rho \vec{g} \quad (18)$$

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 $\nabla \cdot (\vec{\sigma} \cdot \vec{v}) = 0$:

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

2.4 Relating Internal Energy to Temperature

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

$$e = c_p T \quad (20)$$

where c_p is the specific heat capacity at constant volume.

Differentiating with respect to time:

$$\frac{\partial}{\partial t}(\rho e) = \rho c_p \frac{\partial T}{\partial t} \quad (21)$$

2.5 Fourier's Law of Heat Conduction

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

$$\vec{q} = -k \nabla T \quad (22)$$

where k is the thermal conductivity.

Substituting Fourier's law into the simplified energy equation:

$$\rho c_p \frac{\partial T}{\partial t} = -\nabla \cdot (-k \nabla T) + \dot{q} \quad (23)$$

Simplifying:

$$\rho c_p \frac{\partial T}{\partial t} = \nabla \cdot (k \nabla T) + \dot{q} \quad (24)$$

Assuming k is constant:

$$\rho c_p \frac{\partial T}{\partial t} = k \nabla^2 T + \dot{q} \quad (25)$$

2.6 Defining Thermal Diffusivity

Thermal diffusivity α is defined as:

$$\alpha = \frac{k}{\rho c_p} \quad (26)$$

Substituting α into the equation:

$$\frac{\partial T}{\partial t} = \alpha \nabla^2 T + \frac{\dot{q}}{\rho c_p} \quad (27)$$

This is the **heat conduction equation**.

2.7 Final Heat Conduction Equation

The general heat conduction equation in terms of temperature T is:

$$\rho c_p \frac{\partial T}{\partial t} = \nabla \cdot (k \nabla T) + \dot{q} \quad (28)$$

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:

$$\rho c_p \frac{\partial T}{\partial t} = \nabla \cdot (k \nabla T) + \dot{q} \quad \text{in } \Omega \quad (29)$$

with boundary conditions:

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

$$\vec{q} \cdot \mathbf{n} = \bar{q} = -k \nabla T \cdot \mathbf{n} \quad \text{on } \partial\Omega_q \quad (\text{Neumann condition}) \quad (31)$$

Note: \mathbf{n} is the outward normal direction; therefore $\bar{q} > 0$ if \vec{q} going outward, which means \bar{q} is the heat flux going outward.

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 :

$$\int_{\Omega} w \rho c_p \frac{\partial T}{\partial t} dV = \int_{\Omega} w \nabla \cdot (k \nabla T) dV + \int_{\Omega} w \dot{q} dV \quad (32)$$

3.2.1 Integration by Parts

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

$$\int_{\Omega} w \nabla \cdot (k \nabla T) dV = - \int_{\Omega} \nabla w \cdot (k \nabla T) dV + \int_{\partial\Omega} w (k \nabla T \cdot \mathbf{n}) dS \quad (33)$$

Substituting back:

$$\int_{\Omega} w \rho c_p \frac{\partial T}{\partial t} dV = - \int_{\Omega} \nabla w \cdot (k \nabla T) dV + \int_{\partial\Omega} w (k \nabla T \cdot \mathbf{n}) dS + \int_{\Omega} w \dot{q} dV \quad (34)$$

3.2.2 Incorporating Boundary Conditions

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

$$\int_{\partial\Omega_q} w(k\nabla T \cdot \mathbf{n})dS \quad (35)$$

Using the Neumann boundary condition $k\nabla T \cdot \mathbf{n} = -\bar{q}$:

$$\int_{\partial\Omega_q} w(k\nabla T \cdot \mathbf{n})dS = - \int_{\partial\Omega_q} w\bar{q}dS \quad (36)$$

3.2.3 Final Weak Formulation

Collecting terms, the weak form is:

$$\underbrace{\int_{\Omega} w\rho c_p \frac{\partial T}{\partial t} dV}_{\text{Transient Term}} + \underbrace{\int_{\Omega} \nabla w \cdot (k\nabla T) dV}_{\text{Diffusion Term}} = \underbrace{\int_{\Omega} w\dot{q} dV}_{\text{Source Term}} - \underbrace{\int_{\partial\Omega_q} w\bar{q} dS}_{\text{Neumann Boundary}} \quad (37)$$

3.3 Function Spaces

- **Trial Function Space V :**

$$V = \{T \in H^1(\Omega) \mid T = \bar{T} \text{ on } \partial\Omega_T\} \quad (38)$$

- **Test Function Space V_0 :**

$$V_0 = \{w \in H^1(\Omega) \mid w = 0 \text{ on } \partial\Omega_T\} \quad (39)$$

4 LPBF Heat Sources

4.1 Classic Double Ellipsoid Laser Heat Source Model

The double ellipsoid model divides the heat source into front and rear semi-ellipsoids, with the total power distribution $\dot{q} = \dot{q}_{\text{front}} + \dot{q}_{\text{rear}}$. The mathematical expression is:

$$\dot{q}(x, y, z) = \begin{cases} \frac{6\sqrt{3}f_1Q_0}{a_1bc\pi\sqrt{\pi}} \exp\left(-3\left(\frac{x^2}{a_1^2} + \frac{y^2}{b^2} + \frac{z^2}{c^2}\right)\right) & \text{(Front semi-ellipsoid, } x \geq 0) \\ \frac{6\sqrt{3}f_2Q_0}{a_2bc\pi\sqrt{\pi}} \exp\left(-3\left(\frac{x^2}{a_2^2} + \frac{y^2}{b^2} + \frac{z^2}{c^2}\right)\right) & \text{(Rear semi-ellipsoid, } x < 0) \end{cases} \quad (40)$$

Parameters:

- Q_0 : Total laser power (W).
- f_1, f_2 : Power distribution coefficients for the front and rear semi-ellipsoids, satisfying $f_1 + f_2 = 2$.

- a_1, a_2 : Axial lengths along the laser scanning direction (x -axis) for the front and rear semi-ellipsoids (m).
- b : Transverse axial length perpendicular to the scanning direction (m).
- c : Depth-wise axial length (m).

4.1.1 Improvements for LPBF Process

- **Dynamic Coordinate Transformation**

The laser scanning speed v significantly affects the melt pool shape. A fixed coordinate system is transformed into a **moving coordinate system** to dynamically track the heat source position:

$$x' = x - vt, \quad y' = y, \quad z' = z$$

In the moving coordinate system, the heat source model incorporates time t to reflect the transient behavior during scanning.

- **Anisotropic Axial Length Adjustment**

The melt pool shape is influenced by material thermophysical properties (e.g., thermal conductivity, specific heat) and process parameters (e.g., power, scanning speed). The axial lengths a_1, a_2, b, c can be calibrated via experiments or simulations:

$$a_1(t) = k_1 \frac{Q_0}{v(t)\rho c_p}, \quad a_2(t) = k_2 \frac{Q_0}{v(t)\rho c_p}, \quad b(t) = k_3 \frac{Q_0}{v(t)\rho c_p}, \quad c(t) = k_4 \frac{Q_0}{v(t)\rho c_p}$$

where k_1, k_2, k_3, k_4 are empirical coefficients, ρ is density, and c_p is specific heat.

- **Thermal Property at Different Temperature**

The thermal conductivity k of material differs significantly with respect to temperature. It can be expressed as a function of temperature or density:

$$k(T) = k_{\text{bulk}} \cdot (1 + \alpha(T - T_{\text{melt}}))$$

4.2 Convective Heat Flux

$$\bar{q}_{\text{conv}} = h(T - T_{\infty}) \tag{41}$$

- h : Convective heat transfer coefficient ($\text{W}/(\text{m}^2\cdot\text{K})$)
- T_{∞} : Ambient temperature (K)

4.3 Radiative Heat Flux

$$\bar{q}_{\text{rad}} = \varepsilon \sigma_{\text{SB}} (T^4 - T_{\infty}^4) \tag{42}$$

- ε : Emissivity (dimensionless)
- σ_{SB} : Stefan-Boltzmann constant ($\text{W}/(\text{m}^2\cdot\text{K}^4)$)

5 Numerical Solution Using FEniCS

We will solve the heat conduction equation using FEniCS for a cubic domain of size $1000 \times 600 \times 300 \mu\text{m}$ with the following boundary conditions:

- **Bottom Face ($z = 0$):** $T = 300 \text{ K}$
- **Top Face ($z = 300$):** A moving double ellipsoid laser heat source.
- **Other Boundaries:** Convection and radiation to room temperature.

5.1 FEniCS Code with Line-by-Line Explanations

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

```
1 from fenics import *
2 import numpy as np
3 import os
4 from mpi4py import MPI
5 import glob
6
7 # Enable optimization for compilation
8 parameters["form_compiler"]["optimize"] = True
9 parameters["form_compiler"]["cpp_optimize"] = True
10 parameters["form_compiler"]["cpp_optimize_flags"] = "-O3 -ffast-math -march=native"
11 parameters["form_compiler"]["quadrature_degree"] = 2
12 parameters["form_compiler"]["representation"] = "uflacs"
13
14 comm = MPI.COMM_WORLD
15 rank = comm.Get_rank()
16
17 if rank > 0: # Suppress output from non-master processes
18     set_log_level(LogLevel.WARNING)
19
20 if rank == 0: # Only the master process creates the directory
21     if not os.path.exists("results"):
22         os.makedirs("results")
23     else:
24         # Clear existing files
25         files = glob.glob('results/*')
26         for f in files:
27             os.remove(f)
28 comm.barrier() # Ensure all processes wait until the directory is created
29
30 # Material Properties
31 rho = Constant(2700.0) # Density [kg/m^3]
32 cp = Constant(900.0) # Specific heat capacity [J/(kg*K)]
33 k_bulk = Constant(237.0) # Base thermal conductivity [W/(m*K)]
34 alpha = Constant(1e-3) # Thermal conductivity temperature coefficient [1/K]
35
36 # Boundary condition parameters
37 h = 10.0 # Convection coefficient [W/(m^2*K)]
38 T_inf = 300.0 # Ambient temperature [K]
39 epsilon = 0.5 # Emissivity
40 sigma_SB = 5.67e-8 # Stefan-Boltzmann constant
41
42 # Double ellipsoid heat source parameters
43 Q0 = 150 # Laser power [W]
44 v = 1.0 # Scanning speed [m/s]
45 a1, a2, b, c = 50e-6, 200e-6, 50e-6, 50e-6 # Ellipsoid parameters
46 f1, f2 = 0.6, 1.4 # Power distribution coefficients
47
48 # Define computational domain (unit: meters)
```

```

49 Lx, Ly, Lz = 1000e-6, 600e-6, 300e-6
50 meshsz = 30e-6
51 mesh = BoxMesh(Point(0, 0, 0), Point(Lx, Ly, Lz), int(Lx/meshsz), int(Ly/meshsz), int(Lz/
    meshsz))
52 # mesh = BoxMesh(comm, Point(0, 0, 0), Point(Lx, Ly, Lz), 100, 60, 30)
53
54 # Time parameters
55 t_total = Lx/v # Total time [s]
56 dt = t_total/10 # Time step [s]
57 num_steps = int(t_total/dt)
58
59 # Define function space
60 V = FunctionSpace(mesh, 'P', 1)
61
62 # Define test functions and unknown functions
63 w = TestFunction(V)
64 T = Function(V) # Temperature field at current time step
65 T_n = Function(V) # Temperature field at previous time step
66
67 # Initial condition
68 T_n = interpolate(Constant(T_inf), V) # Initial temperature field
69
70 # Boundary condition definitions
71 # Define boundary locations
72 def top(x, on_boundary):
73     return near(x[2], Lz) and on_boundary
74
75 def bottom(x, on_boundary):
76     return near(x[2], 0.) and on_boundary
77
78 def walls(x, on_boundary):
79     left = near(x[0], 0.) and on_boundary
80     right = near(x[0], Lx) and on_boundary
81     front = near(x[1], 0.) and on_boundary
82     back = near(x[1], Ly) and on_boundary
83     return left | right | front | back
84
85 # Mark boundaries
86 boundaries = MeshFunction("size_t", mesh, mesh.topology().dim() - 1, 0)
87 Top = AutoSubDomain(top)
88 Top.mark(boundaries, 1)
89 Bottom = AutoSubDomain(bottom)
90 Bottom.mark(boundaries, 2)
91 Walls = AutoSubDomain(walls)
92 Walls.mark(boundaries, 3)
93 ds = Measure('ds', domain=mesh, subdomain_data=boundaries) # Redefine the measure 'ds' with
    subdomains
94
95 bc = DirichletBC(V, Constant(T_inf), bottom)
96 bcs = [bc]
97
98 # Temperature-dependent thermal conductivity
99 def thermal_conductivity(T):
100     return k_bulk * (1 + alpha*(T - 300))
101
102 # Define volumetric heat source for laser scanning
103 class HeatSource(UserExpression):
104     def __init__(self, position, velocity, Q0, f1, f2, a1, a2, b, c, t, **kwargs):
105         super().__init__(**kwargs)
106
107         # Validate that position is a vector of length 3
108         if len(position) != 3:
109             raise ValueError("Position must be a vector with exactly 3 elements (x, y, z).")
110
111         # Validate that velocity is a vector of length 3
112         if len(velocity) != 3:
113             raise ValueError("Velocity must be a vector with exactly 3 elements (vx, vy, vz)
    .")

```

```

114
115     self.position = np.array(position, dtype=float) # Initial position [x0, y0, z0]
116     self.velocity = np.array(velocity, dtype=float) # Velocity [vx, vy, vz]
117     self.Q0 = Q0
118     self.f1 = f1
119     self.f2 = f2
120     self.a1 = a1
121     self.a2 = a2
122     self.b = b
123     self.c = c
124     self.t = t
125
126     def eval(self, value, x):
127
128         laser_center = self.position + self.velocity * self.t # Laser center position
129
130         # Relative position
131         x_prime = x - laser_center
132
133         # Calculate heat source intensity
134         if x_prime[0] >= 0:
135             coeff = 6*sqrt(3) * self.f1 * self.Q0/(self.a1*self.b*self.c*np.pi*sqrt(np.pi))
136             exponent = -3*((x_prime[0])**2/self.a1**2 + x_prime[1]**2/self.b**2 + x_prime
137                 [2]**2/self.c**2)
138             else:
139                 coeff = 6*sqrt(3) * self.f2 * self.Q0/(self.a2*self.b*self.c*np.pi*sqrt(np.pi))
140                 exponent = -3*((x_prime[0])**2/self.a2**2 + x_prime[1]**2/self.b**2 + x_prime
141                     [2]**2/self.c**2)
142
143         value[0] = coeff * exp(exponent)
144
145     def value_shape(self):
146         return ()
147
148 # Create heat source object
149 position = [-a1, Ly/2, Lz]
150 velocity = [v, 0.0, 0.0]
151 q_dot = HeatSource(position, velocity, Q0, f1, f2, a1, a2, b, c, t=0, degree=1)
152
153 # Define radiative heat flux
154 q_bar_conv = h * (T - T_inf) # Convective heat flux
155 q_bar_rad = epsilon * sigma_SB * (T**4 - T_inf**4) # Radiative heat flux
156
157 # Define variational form
158 F = w * rho * cp * (T - T_n) / dt * dx \
159     + inner(grad(w), thermal_conductivity(T)*grad(T))*dx \
160     - w * q_dot * dx \
161     + w * (q_bar_rad + q_bar_conv) * ds(3)
162
163 # Create nonlinear problem and solver
164 problem = NonlinearVariationalProblem(F, T, bc, J=derivative(F, T))
165 solver = NonlinearVariationalSolver(problem)
166 solver.parameters['newton_solver']['relaxation_parameter'] = 1.0
167 solver.parameters["newton_solver"]["linear_solver"] = "cg"
168 solver.parameters["newton_solver"]["maximum_iterations"] = 50
169
170 # Time-stepping loop =====
171 t = 0.0 # Initialize time
172
173 # Create file to save results
174 file = XDMFFile("results/temperature.xdmf")
175 file.write(mesh)
176 file.parameters["flush_output"] = True
177
178 # save initial condition
179 T_n.rename("Temperature", "Temperature")
180 file.write(T_n, t)

```

```

180
181 # Initialize heat source object (Note: time parameter must be initialized first)
182 q_dot.t = t
183
184 for step in range(num_steps):
185     # Update time
186     t += dt
187
188     # Print progress information
189     if rank == 0:
190         print(f"\n===== Computing time step {step + 1}/{num_steps} [t = {t:.6f} s] =====")
191
192     # Update heat source position (Critical step!)
193     # -----
194     q_dot.t = t # Update heat source time parameter
195
196     # Solve nonlinear problem
197     # -----
198     try:
199         # Call solver
200         solver.solve()
201
202         # Check solution validity
203         max_T = T.vector().max()
204         min_T = T.vector().min()
205         if max_T > 5000 or min_T < 0:
206             print(f"Warning: Temperature solution out of physical range! Max temperature: {
max_T} K, Min temperature: {min_T} K")
207             break
208
209     except Exception as e:
210         print(f"Solving failed at time step {step}, error message:")
211         print(str(e))
212         break
213
214     # Update solution from the previous time step
215     # -----
216     T_n.assign(T)
217
218     # Save results
219     # -----
220     if (step + 1) % 2 == 0:
221         T.rename("Temperature", "Temperature")
222         file.write(T, t)
223         if rank == 0:
224             print(f"Results saved at time step {step + 1}")
225
226
227 # Save final result
228 file.close()
229 if rank == 0:
230     print("\nComputation completed! Results saved to results/temperature.pvd")

```

Listing 1: FEniCS Code for LPBF Heat Conduction

6 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.

7 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/>