Thermo-Elastic Analysis: Equations and Code Explanation

Yazhuo Liu

November 9, 2024

Contents

1 Overview

The provided code performs a thermo-mechanical finite element analysis using the FEniCS library. It simulates the temperature distribution and mechanical deformation (stress and displacement) in a material subjected to a moving laser heat source. The analysis couples the thermal and mechanical problems to capture the effects of thermal expansion and temperature-dependent material properties.

2 Equations and Derivations

2.1 Thermal Problem

The thermal analysis is based on the heat conduction equation with a moving heat source (laser), convection, and radiation:

$$\rho C_p \frac{\partial T}{\partial t} = \nabla \cdot (k \nabla T) + Q_{\text{laser}} - Q_{\text{conv}} - Q_{\text{rad}} \tag{1}$$

- ρ: Density (kg/m^3)
- C_p: Specific heat capacity (J/(kg·K))
- T: Temperature (K)
- k: Thermal conductivity (W/(m·K))
- Q_{laser}: Heat input from the laser (W/m^2)
- Q_{conv}: Convective heat loss (W/m^2)
- Q_{rad}: Radiative heat loss (W/m^2)

2.1.1 Laser Heat Flux Q_{laser}

The laser is modeled as a Gaussian moving heat source:

$$Q_{\text{laser}}(x, y, t) = \frac{2\eta P}{\pi r_b^2} \exp\left(-\frac{2\left[(x - x_0 - v_x t)^2 + (y - y_0 - v_y t)^2\right]}{r_b^2}\right) \tag{2}$$

- η: Absorption coefficient (dimensionless)
- P: Laser power (W)
- r_b: Laser beam radius (m)
- (x_0, y_0): Initial laser position (m)
- (v_x, v_y): Laser velocity components (m/s)
- t: Time (s)

2.1.2 Convective Heat Loss Q_{conv}

$$Q_{\text{conv}} = h(T - T_\infty) \tag{3}$$

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

2.1.3 Radiative Heat Loss Q_{rad}

$$Q_{\text{rad}} = \varepsilon \sigma_{\text{SB}} \left(T^4 - T_\infty^4\right) \tag{4}$$

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

2.2 Mechanical Problem

The mechanical analysis uses linear elasticity with thermal strain and temperature-dependent material properties.

2.2.1 Stress-Strain Relationship

$$\boldsymbol{\sigma} = \xi(T)\left[\lambda\,\mathrm{tr}\left(\boldsymbol{\varepsilon}(\mathbf{u}) - \boldsymbol{\varepsilon}_T(T)\right)\mathbf{I} + 2\mu\left(\boldsymbol{\varepsilon}(\mathbf{u}) - \boldsymbol{\varepsilon}_T(T)\right)\right] \tag{5}$$

- $\boldsymbol{\sigma}$: Stress tensor (Pa)
- $\boldsymbol{\varepsilon}(\mathbf{u})$: Strain tensor derived from displacement $\mathbf{u}$
- $\boldsymbol{\varepsilon}_T(T)$: Thermal strain tensor
- λ, μ: Lamé's first and second parameters (Pa)
- $\mathbf{I}$: Identity tensor
- $\xi(T)$: Degeneration function due to thermal softening

2.2.2 Strain Tensor

$$\boldsymbol{\varepsilon}(\mathbf{u}) = \frac{1}{2}\left(\nabla\mathbf{u} + (\nabla\mathbf{u})^\top\right) \tag{6}$$

2.2.3 Thermal Strain Tensor

$$\boldsymbol{\varepsilon}_T(T) = \alpha(T - T_0)\mathbf{I} \tag{7}$$

- α: Coefficient of thermal expansion (/K)
- T_0: Reference temperature (K)

2.2.4 Degeneration Function $\xi(T)$

$$\xi(T) = \begin{cases} 1 - 0.5\left(\dfrac{T - T_0}{T_l - T_0}\right), & T < T_l \\ \epsilon, & T \geq T_l \end{cases} \tag{8}$$

- T_l: Liquidus temperature (melting point) (K)
- ϵ: Small value to prevent zero stress (e.g., 1×10^{-6})

2.3 Von Mises Stress

The Von Mises stress σ_{VM} is calculated from the deviatoric stress tensor:

$$\sigma_{\mathrm{VM}} = \sqrt{\frac{3}{2}\boldsymbol{s} : \boldsymbol{s}} \tag{9}$$

- $\boldsymbol{s} = \boldsymbol{\sigma} - \dfrac{1}{3}\,\mathrm{tr}(\boldsymbol{\sigma})\mathbf{I}$: Deviatoric stress tensor

3 Code Explanation

3.1 Import Libraries

```python
from dolfin import *
import numpy as np
import os
import glob
```

Listing 1: Import Libraries

- `dolfin`: Main FEniCS library for finite element methods.
- `numpy`: Numerical operations.
- `os`, `glob`: File system operations for data management.

3.2 Set Up Directories

```python
# Define useful directories
crt_file_path = os.path.dirname(__file__)
data_dir = os.path.join(crt_file_path, "ThermalMechanicalData")
vtk_dir = os.path.join(data_dir, "VTK")
```

Listing 2: Define Useful Directories

- `crt_file_path`: Current script directory.
- `data_dir`: Directory for simulation data.
- `vtk_dir`: Directory for storing VTK files for visualization.

3.3 Initialize Output Files

```python
# Do some cleaning before running the simulation
files = glob.glob(os.path.join(vtk_dir, f"*"))
for f in files:
    os.remove(f)
vtkfile_temp = File(os.path.join(vtk_dir, "Temperature.pvd"))
vtkfile_disp = File(os.path.join(vtk_dir, "Displacement.pvd"))
vtkfile_stress = File(os.path.join(vtk_dir, "Stress.pvd"))
vtkfile_vonmises = File(os.path.join(vtk_dir, "VonMises.pvd"))
```

Listing 3: Initialize Output Files

- Remove old VTK files to start fresh.
- Create VTK file objects for temperature, displacement, stress, and Von Mises stress.

3.4 Material Properties

```python
# Define material parameters
E = 210e9           # Young's Modulus (Pa)
nu = 0.3            # Poisson's Ratio
alpha = 1.2e-5      # Thermal expansion coefficient (/deg C)
Cp = 588.           # Heat capacity, J/(kg K)
rho = 8440.         # Density, kg/m^3
k = 15.             # Thermal conductivity, W/(m K)
Tl = 1623.          # Liquidus temperature, K
h = 100.            # Heat convection coefficient, W/(m^2 K)
eta = 0.25          # absorption rate
SB_constant = 5.67e-8  # Stefan-Boltzmann constant, W/(m^2 K^4)
emissivity = 0.5    # Emissivity of the surface
T0 = 300.           # Ambient temperature, K
```

Listing 4: Define Material Parameters

- Elastic, thermal, and physical properties of the material.

3.5 Laser Parameters

```
# Define laser properties
vel = 0.5           # Laser velocity, m/s
rb = 0.05e-3        # Laser beam radius, m
P = 20.             # Laser power, W
```

Listing 5: Define Laser Properties

- `vel`: Speed at which the laser moves.
- `rb`: Radius of the laser beam.
- `P`: Power output of the laser.

3.6 Mesh Definition

```
# Define mesh
Lx, Ly, Lz = 0.5e-3, 0.2e-3, 0.05e-3
Nx, Ny, Nz = 50, 20, 5
mesh = BoxMesh(Point(0., 0., 0.), Point(Lx, Ly, Lz), Nx, Ny, Nz)
# ResultFile.write(mesh)
```

Listing 6: Define Mesh

- Create a 3D box mesh representing the material domain.
- `Lx`, `Ly`, `Lz`: Dimensions of the domain.
- `Nx`, `Ny`, `Nz`: Number of divisions in each direction.

3.7 Time Parameters

```
# Define time parameters
laser_on_t = 0.5*Lx / vel
simulation_time = 2*laser_on_t
dt = 10e-5
num_steps = int(simulation_time / dt)
ts = np.linspace(0, simulation_time, num_steps+1)
```

Listing 7: Define Time Parameters

- `laser_on_t`: Time until the laser reaches the middle of the domain.
- `simulation_time`: Total time to simulate the laser moving across the domain.
- `dt`: Time step size.
- `num_steps`: Number of time steps.
- `ts`: Array of time points.

3.8 Boundary Definitions

```
# Define boundary locations
def top(x, on_boundary):
    return near(x[2], Lz) and on_boundary

def bottom(x, on_boundary):
    return near(x[2], 0.) and on_boundary

def walls(x, on_boundary):
    left = near(x[0], 0.) and on_boundary
    right = near(x[0], Lx) and on_boundary
    front = near(x[1], 0.) and on_boundary
```

```
    back = near(x[1], Ly) and on_boundary
    return left | right | front | back
```

Listing 8: Define Boundary Locations

- Functions to identify the top, bottom, and side walls of the domain.

3.9 Boundary Marking

```
# Mark boundaries
boundaries = MeshFunction("size_t", mesh, mesh.topology().dim() - 1, 0)
Top = AutoSubDomain(top)
Top.mark(boundaries, 1)
Bottom = AutoSubDomain(bottom)
Bottom.mark(boundaries, 2)
Walls = AutoSubDomain(walls)
Walls.mark(boundaries, 3)
ds = Measure('ds', domain=mesh, subdomain_data=boundaries)  # Redefine the measure 'ds' with
    subdomains
```

Listing 9: Mark Boundaries

- Assign unique markers to each boundary for applying boundary conditions.
- `ds`: Redefine boundary measure to include subdomains.

3.10 Function Spaces

```
# Define function space
R = FunctionSpace(mesh, "P", 1)
T = Function(R)           # Temperature at current time step
Q = TestFunction(R)       # Test function for temperature
T_old = Function(R)       # Temperature at previous time step
V = VectorFunctionSpace(mesh, 'P', 1, dim=3)
u = TrialFunction(V)      # Displacement trial function
v = TestFunction(V)       # Displacement test function
u_sol = Function(V)       # Displacement solution
S = TensorFunctionSpace(mesh, 'P', 1)
stress = Function(S)      # Stress tensor
Von = FunctionSpace(mesh, 'P', 1)
von_mises = Function(Von) # Von Mises stress
```

Listing 10: Define Function Spaces

- `R`: Scalar space for temperature.
- `V`: Vector space for displacement.
- `S`: Tensor space for stress.
- `Von`: Scalar space for Von Mises stress.

3.11 Initial Temperature

```
# Thermal problem
# Define initial conditions
T_init = Constant(T0)
T_old.assign(T_init)
```

Listing 11: Define Initial Conditions

- Set the entire domain to the ambient temperature T_0.

3.12 Laser Flux Class

```
# Define the laser flux expression
class LaserFlux(UserExpression):
    def __init__(self, switch, x0, y0, vx, vy, rb, P, eta, t, **kwargs):
        super().__init__(**kwargs)
        self.switch = switch
        self.x0 = x0
        self.y0 = y0
        self.vx = vx
        self.vy = vy
        self.rb = rb
        self.P = P
        self.eta = eta
        self.t = t

    def eval(self, values, x):
        laser_center_x = self.x0 + self.vx * self.t
        laser_center_y = self.y0 + self.vy * self.t
        distance = np.sqrt((x[0] - laser_center_x)**2 + (x[1] - laser_center_y)**2)
        values[0] = self.switch * 2 * self.eta * self.P / (np.pi * self.rb**2) * np.exp(-2 *
    distance**2 / self.rb**2)

    def value_shape(self):
        return ()
```

Listing 12: Define the Laser Flux Expression

- Custom expression to calculate the laser heat flux at any point and time.
- `switch`: Controls laser on/off.
- `eval`: Calculates the heat flux based on the distance from the laser center.

3.13 Initialize Heat Fluxes

```
# Define laser flux
q_laser = LaserFlux(switch=1, x0=Lx*0.25, y0=Ly/2, vx=vel, vy=0., rb=rb, P=P, eta=eta, t=0.,
    degree=2)
q_conv = h * (T0 - T_old)
q_rad = emissivity * SB_constant * (T0**4 - T_old**4)
```

Listing 13: Initialize Heat Fluxes

- `q_laser`: Laser heat input.
- `q_conv`: Convective heat loss (depends on current temperature).
- `q_rad`: Radiative heat loss (depends on current temperature).

3.14 Thermal Boundary Conditions

```
# Define boundary condition for the bottom face (Dirichlet condition)
bc_bottom = DirichletBC(R, T0, bottom)
bcsT = [bc_bottom]
```

Listing 14: Define Boundary Condition for the Bottom Face

- Fix temperature at the bottom surface to the ambient temperature T_0.

3.15 Variational Formulation for Thermal Problem

```
# Weak form for the thermal problem
FT_1 = rho * Cp * (T - T_old) / dt * Q * dx + k * inner(grad(T), grad(Q)) * dx
FT_2 = - (q_conv + q_rad + q_laser) * Q * ds(1) - (q_conv + q_rad) * Q * ds(3)
FT = FT_1 + FT_2
```

Listing 15: Weak Form for the Thermal Problem

- `FT_1`: Time derivative and conduction terms.
- `FT_2`: Heat fluxes applied on boundaries.
  - `ds(1)`: Top surface (laser, convection, radiation).
  - `ds(3)`: Side walls (convection, radiation).
- `FT`: Total residual for the thermal problem.

3.16 Solver Setup for Thermal Problem

```
# Define the variational problem
problemT = NonlinearVariationalProblem(FT, T, bcs=bcsT, J=derivative(FT, T))
solverT = NonlinearVariationalSolver(problemT)
solverT.parameters["newton_solver"]["absolute_tolerance"] = 1e-8
solverT.parameters["newton_solver"]["relative_tolerance"] = 1e-7
solverT.parameters["newton_solver"]["maximum_iterations"] = 100
solverT.parameters["newton_solver"]["relaxation_parameter"] = 1.0
```

Listing 16: Define the Variational Problem and Solver for Thermal Problem

- Define the nonlinear problem for temperature.
- Use Newton's method with specified tolerances and parameters.

3.17 Lamé Parameters Calculation

```
# Mechanical problem
# Lame parameters
mu = E / (2 * (1 + nu))
lmbda = E * nu / ((1 + nu)*(1 - 2*nu))
```

Listing 17: Calculate Lamé Parameters

- `mu`: Shear modulus.
- `lmbda`: First Lamé parameter.

3.18 Mechanical Boundary Conditions

```
# Define the boundary conditions
bc_bottom = DirichletBC(V, Constant((0.0, 0.0, 0.0)), bottom)
bcsu = [bc_bottom]
```

Listing 18: Define Mechanical Boundary Conditions

- Fix displacement at the bottom surface (clamped boundary).

3.19 Strain Definitions

```
# Define the strain tensor (symmetric gradient of displacement)
def epsilon(u):
    return sym(grad(u))

# Define the thermal strain tensor using as_tensor for correct shape
def epsilon_t(T):
    alpha_T = conditional(T < Tl, alpha, 0)
    return alpha_T * (T - T0) * as_tensor(np.eye(3))
```

Listing 19: Define Strain Tensors

- `epsilon(u)`: Mechanical strain tensor.
- `epsilon_t(T)`: Thermal strain tensor, active only below the liquidus temperature.

3.20 Degeneration Function

```
# Define the degeneration function for thermal softening and melting.
def degeneration(T):
    return conditional(T < T0, 1, conditional(T < Tl, 1 - (T - T0) / (Tl - T0), 1e-3))
```

Listing 20: Define the Degeneration Function

- Reduces material stiffness as temperature approaches melting point.

3.21 Stress Tensor Definition

```
# Define the stress tensor using Hooke's Law with thermal strain
def sigma(u, T):
    sig = lmbda * tr(epsilon(u) - epsilon_t(T)) * Identity(3) + 2 * mu * (epsilon(u) - epsilon_t(T
    ))
    xi = conditional(T < Tl, 1 - 0.5 * (T - T0) / (Tl - T0), 1e-6)
    return sig * xi
```

Listing 21: Define the Stress Tensor

- Calculates stress considering thermal strain and temperature-dependent material degradation.

3.22 Variational Formulation for Mechanical Problem

```
# Weak form of the mechanical problem
Fu = inner(sigma(u, T), epsilon(v)) * dx
problemu = LinearVariationalProblem(lhs(Fu), rhs(Fu), u_sol, bcsu)
solveru = LinearVariationalSolver(problemu)
```

Listing 22: Weak Form of the Mechanical Problem

- Defines the weak form of the mechanical equilibrium equations.
- Sets up a linear solver since the problem is linear in displacement.

3.23 Time-Stepping Loop

```
for t in ts:

    print(f"Time: {t:.5f} s")

    q_laser.t = t
    if t > laser_on_t:
        q_laser.switch = 0

```

```
    solverT.solve()
    T_old.assign(T)
    T.rename("Temperature", "Temperature")

    solveru.solve()
    u_sol.rename("Displacement", "Displacement")

    stress.assign(project(sigma(u_sol, T), S))
    stress.rename("Stress", "Stress")

    stress_dev = stress - (1./3)*tr(stress)*Identity(3)
    von_mises.assign(project(sqrt(3./2*inner(stress_dev, stress_dev)), Von))
    von_mises.rename("Von Mises Stress", "Von Mises Stress")

    vtkfile_temp << (T, t)
    vtkfile_disp << (u_sol, t)
    vtkfile_stress << (stress, t)
    vtkfile_vonmises << (von_mises, t)
```

Listing 23: Time-Stepping Loop

- **Update Laser Position and Status:**
  - Update laser position for current time t.
  - Turn off the laser after `laser_on_t`.
- **Solve Thermal Problem:**
  - Compute temperature distribution.
  - Update previous temperature `T_old`.
- **Solve Mechanical Problem:**
  - Compute displacement field.
- **Compute Stress and Von Mises Stress:**
  - Project stress tensor and Von Mises stress for visualization.
- **Save Results:**
  - Write temperature, displacement, stress, and Von Mises stress to VTK files for each time step.

4 Summary

The code simulates the coupled thermo-mechanical behavior of a material under a moving laser heat source. It accounts for:

- Heat conduction with a moving Gaussian heat source, convection, and radiation.
- Thermal expansion leading to mechanical deformation.
- Temperature-dependent material properties and degradation near the melting point.
- Computation of stress tensors and Von Mises stress for assessing material response.

The results can be visualized using VTK-compatible software, allowing analysis of temperature distribution, displacement fields, stress tensors, and regions of high stress concentration over time.