FEniCS: Time Dependent Heat Equation

Ida Ang, Edited April 22, 2023

Contents

1 Problem Definition

This document references A Gallary of finite element solvers, the heat equation. Some of the text is taken from either the documentation online or the code, but the goal of this document is to provide more details of the theory and conversion from the strong form, or partial differential equations, to the weak form, or integral equation. This document provides the detailed steps for beginners who might have just been exposed to indicial notation. In direct notation, bolded uncapitalized notation refers to vectors, unbolded notation refers to scalar variables.

2 Formulations

The simplest formulation for Poisson's problem involves solving for one variable, u, which is known as a scalar field of scalar potential. The Poisson's equation shows up in many contexts, and the variable u could be the concentration of some chemical solute, as a function of position x, or the temperature T in some heat conducting medium. Classically, the equation can be referred to as the divergence of the gradient of some scalar field which then returns a scalar, some source term f.

$$\nabla \cdot \nabla u = \nabla^2 u = \Delta u = f \quad \text{in } \Omega \tag{1a}$$

It can also be referred to as the Laplacian of the variable field, where several notations are listed in Eq. 1. As an extension of the Poisson problem, which can be used to describe the stationary distribution of heat in a body, we can consider the time-dependent heat equation (or time-dependent diffusion equation), which describes the distribution of heat in a body over time.

$$\frac{\partial u}{\partial t} = \nabla^2 u + f \quad \text{in } \Omega \times (0, T], \tag{2a}$$

$$u = u_{\mathrm{D}} \quad \text{on } \partial\Omega \times (0, T] \tag{2b}$$

$$u = u_0 \quad \text{at } t = 0. \tag{2c}$$

where in this particular problem, the Dirichlet boundary condition consists of the full boundary, $\partial\Omega = \Gamma_D$, and there is no Neumann boundary condition. The Dirichlet boundary condition (BC) is the essential boundary condition applied within the function space. Recall, that the Neumann boundary condition can be stated as in Eq. 3,

$$\nabla u \cdot \mathbf{n} = \frac{\partial u}{\partial \mathbf{n}} = g \qquad \text{on } \Gamma_N \tag{3a}$$

$$\frac{\partial u}{\partial x_i} n_i = g \tag{3b}$$

where it is the natural boundary condition which is subsumed into the variational weak form. The source term f can be defined as follows:

$$f(x, y, t) = 1 + x^2 + \alpha y^2 + \beta t \tag{4}$$

In order to obtain the weak form, 1) multiply the strong form with a test function, 2) Integrate over the domain of interest, and 3) use integration by parts.

$$\int_\Omega \frac{\partial u}{\partial t} v \, dv = \int_\Omega \frac{\partial^2 u}{\partial x^2} v \, dv + \int_\Omega f v \, dV v \tag{5}$$

Use integration by parts,

$$(fg)' = f'g + fg' \rightarrow f'g = (fg)' - fg' \rightarrow f''g = (f'g)' - f'g' \tag{6}$$

Where we can substitute $f'' = \frac{\partial^2 u}{\partial x_i^2}$ and $g = v$ into Eq. 6

$$\frac{\partial^2 u}{\partial x^2} v = \nabla \cdot \left(\frac{\partial u}{\partial x} v \right) - \frac{\partial u}{\partial x}\frac{\partial v}{\partial x}$$

Substitute into Eq. 5

$$\begin{aligned}
\int_\Omega \frac{\partial u}{\partial t} v\, dv &= \int_\Omega \frac{\partial^2 u}{\partial x^2} v\, dv + \int_\Omega f v\, dv \\
\int_\Omega \frac{\partial u}{\partial t} v\, dv &= \int_\Omega \nabla \cdot \left(\frac{\partial u}{\partial x} v \right) dv - \int_\Omega \frac{\partial u}{\partial x}\frac{\partial v}{\partial x}\, dv + \int_\Omega f v\, dv \quad \text{Divergence Theorem} \\
\int_\Omega \frac{\partial u}{\partial t} v\, dv &= \int_{\partial\Omega} \left(\frac{\partial u}{\partial x} \cdot \mathbf{n} \right) v\, ds - \int_\Omega \frac{\partial u}{\partial x}\frac{\partial v}{\partial x}\, dv + \int_\Omega f v\, dv \\
\int_\Omega \frac{\partial u}{\partial t} v\, dv &= \int_{\partial\Omega} g v\, ds - \int_\Omega \frac{\partial u}{\partial x}\frac{\partial v}{\partial x}\, dv + \int_\Omega f v\, dv
\end{aligned}$$

Therefore, the weak form in direct notation

$$\int_\Omega \frac{\partial u}{\partial t} v\, dv + \int_\Omega \nabla u \cdot \nabla v\, dv = \int_\Omega f \cdot v\, dv + \int_{\Gamma_N} g \cdot v\, ds \tag{7}$$

2.1 Time Integration

The time derivative can be discretized by a finite difference approximation, where the following approximation is given for a backwards difference at time t_{n+1}

$$\left(\frac{\partial u}{\partial t} \right)^{n+1} \approx \frac{u^{n+1} - u^n}{\Delta t} \tag{8}$$

where we can substitute Eq. 8 into Eq. 7 at time t_{n+1}

$$\begin{aligned}
\int_\Omega \left(\frac{\partial u}{\partial t} \right)^{n+1} v\, dv + \int_\Omega \nabla u^{n+1} \cdot \nabla v\, dv &= \int_\Omega f^{n+1} \cdot v\, dv + \int_{\Gamma_N} g^{n+1} \cdot v\, ds \\
\int_\Omega \frac{u^{n+1} - u^n}{\Delta t} v\, dv + \int_\Omega \nabla u^{n+1} \cdot \nabla v\, dv &= \int_\Omega f^{n+1} \cdot v\, dv + \int_{\Gamma_N} g^{n+1} \cdot v\, ds \\
\int_\Omega \left(u^{n+1} - u^n\right) v\, dv + \Delta t \int_\Omega \nabla u^{n+1} \cdot \nabla v\, dv &= \Delta t \int_\Omega f^{n+1} \cdot v\, dv + \Delta t \int_{\Gamma_N} g^{n+1} \cdot v\, ds
\end{aligned}$$

where we assume u^n is a quantity from the previous time step, t_n, while we are computing the solution at time t_{n+1}; therefore, we can introduce u instead of u^{n+1}, etc. giving the following equation.

$$\int_\Omega \left(u - u^n\right) v\, dv + \Delta t \int_\Omega \nabla u \cdot \nabla v\, dv = \Delta t \int_\Omega f \cdot v\, dv + \Delta t \int_{\Gamma_N} g \cdot v\, ds \tag{9}$$

Eq. 9 is the general form, but for our specific problem there is no Neumann boundary condition leading to the following equation.

$$\int_\Omega \left(u - u^n\right) v\, dv + \Delta t \int_\Omega \nabla u \cdot \nabla v\, dv = \Delta t \int_\Omega f \cdot v\, dv \tag{10}$$

3 FEniCS Implementation

Just as in the prior section, the variable u is used for the unknown value at time step t_{n+1} for the unknown u^{n+1} and the variable u_n is used for u^n at the previous (known) time step. The initial value of u_n can be computed by either projection or interpolation of u_0, which in this case is defined with the expression in Eq. 4

```
u_D = Expression('1 + x[0]*x[0] + alpha*x[1]*x[1] + beta*t',
                 degree=2, alpha=alpha, beta=beta, t=0)
```

Note, to recover the exact solution to machine precision, it is important to compute the discrete initial condition by interpolating u_0, which ensures that the degrees of freedom are exact (to machine precision) at t=0. Projection results in approximate values at the nodes. The syntax for either of these commands are included below.

```
u_n = interpolate(u_D, V)
u_n = project(u_D, V)
```

Equation 10 can be stated by moving all the variables to the left hand side of the equation and letting FEniCS determine the bilinear and linear forms.

```
F = u*v*dx + dt*dot(grad(u), grad(v))*dx - (u_n + dt*f)*v*dx
a, L = lhs(F), rhs(F)
```

Within the loop, the problem an be solved for each time step. For verification, the expression can be interpolated into the function space as the exact solution u_e. The error can be computed and saved to an array called `PostTxt` which is then saved to a text file for plotting in `PostProc.py`

```
PostTxt = np.zeros((NumSteps, 3))
for n in range(NumSteps):
  solve(a == L, u, bc)
  u_e = interpolate(u_D, V)
  error = np.abs(u_e.vector() - u.vector()).max()
  PostTxt[n, :]  = np.array([n, t, error])
  np.savetxt(Dir + '/PostProc.txt', PostTxt)
```

Lastly, the proper variables must be updated. Inside the time loop, observe that u_D.t must be updated to enforce computation of Dirichlet conditions at the next time step. A Dirichlet condition defined in terms of an Expression looks up and applies the value of a parameter such as t when it gets evaluated and applied to the linear system. Furthermore, we must assign the new computed solution u to the variable u_n (previous). If we set u_n = u, the u_n variable will be the SAME variable as u, but we need two values, one for the previous and current time step.

```
for n in range(NumSteps):
  t += dt
  u_D.t = t
  u_n.assign(u)
```

4 Results

Snapshots of the simulation for $N_x = N_y = 100$ are included in Fig. 1.

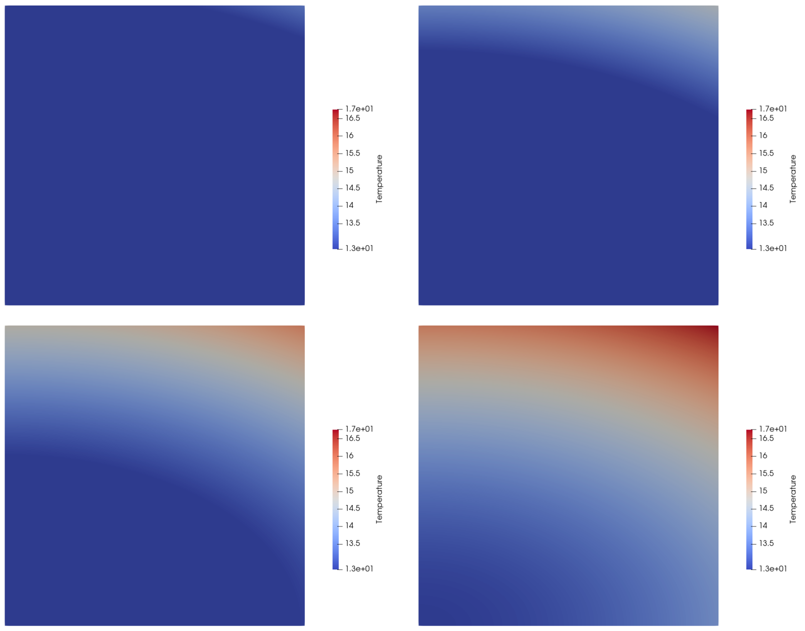

Figure 1: Temperature at step 35, 40, 45, and 49 for $\Delta t = 0.2$

Fig. 2 demonstrates the error over the course of the simulation steps and simulation time for different mesh resolutions using the `interpolation` command, while Fig. 3 demonstrates the same except the `project` command is used.

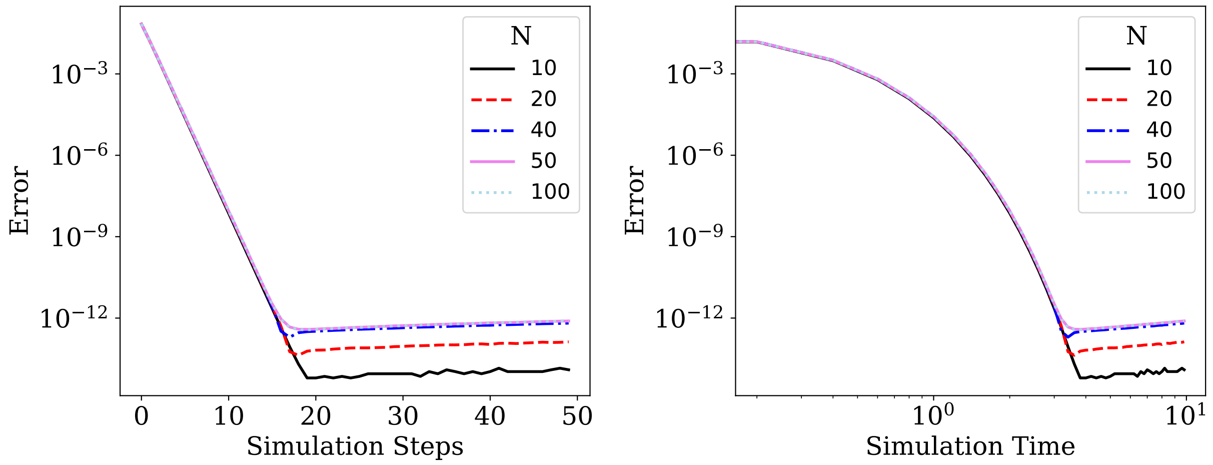

Figure 2: Error plotting for different mesh resolutions using the interpolation command.

There is no discernible difference in the error from these plots, and the error is relatively low and demonstrates small but negligible differences between the computation and exact solution.

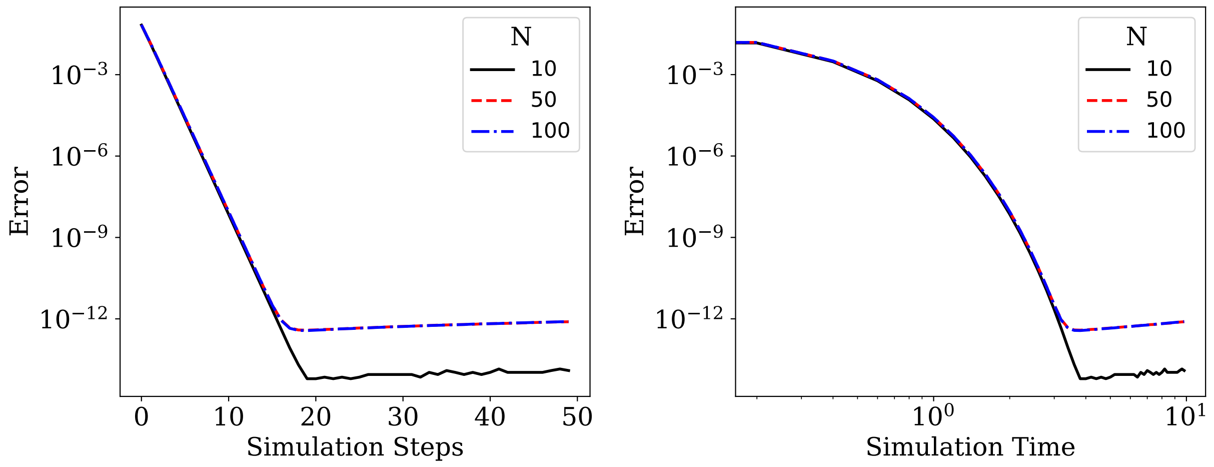

Figure 3: Error plotting for different mesh resolutions using the project command.