

# FEniCS Course

## Lecture 4: Time-dependent PDEs

---

### *Contributors*

Hans Petter Langtangen

Anders Logg

Marie E. Rognes



FENICS  
PROJECT

# The heat equation

We will solve the simplest extension of the Poisson problem into the time domain, the heat equation:

$$\begin{aligned}\frac{\partial u}{\partial t} - \Delta u &= f \quad \text{in } \Omega \text{ for } t > 0 \\ u &= g \quad \text{on } \partial\Omega \text{ for } t > 0 \\ u &= u^0 \quad \text{in } \Omega \text{ at } t = 0\end{aligned}$$

The solution  $u = u(x, t)$ , the right-hand side  $f = f(x, t)$  and the boundary value  $g = g(x, t)$  may vary in space ( $x = (x_0, x_1, \dots)$ ) and time ( $t$ ). The initial value  $u^0$  is a function of space only.

# Time-discretization of the heat equation

We discretize in time using the implicit Euler (dG(0)) method:

$$\frac{\partial u}{\partial t}(t^n) \approx \frac{u^n - u^{n-1}}{\Delta t}, \quad u(t^n) \approx u^n, \quad f^n = f(t^n)$$

Semi-discretization of the heat equation:

$$\frac{u^n - u^{n-1}}{\Delta t} - \Delta u^n = f^n$$

## Algorithm

- 1 Start with  $u^0$  and choose a timestep  $\Delta t > 0$ .
- 2 For  $n = 1, 2, \dots$ , solve for  $u^n$ :

$$u^n - \Delta t \Delta u^n = u^{n-1} + \Delta t f^n$$

# Variational problem for the heat equation

Find  $u^n \in V^n$  such that

$$a(u^n, v) = L^n(v)$$

for all  $v \in \hat{V}$  where

$$\begin{aligned} a(u, v) &= \int_{\Omega} uv + \Delta t \nabla u \cdot \nabla v \, dx \\ L^n(v) &= \int_{\Omega} u^{n-1} v + \Delta t f^n v \, dx \end{aligned}$$

Note that the bilinear form  $a(u, v)$  is constant while the linear form  $L^n$  depends on  $n$

# Detailed time-stepping algorithm for the heat equation

*Define the boundary condition*

*Compute  $u^0$  as the projection of the given initial value*

*Define the forms  $a$  and  $L$*

*Assemble the matrix  $A$  from the bilinear form  $a$*

$t \leftarrow \Delta t$

**while**  $t \leq T$  **do**

*Assemble the vector  $b$  from the linear form  $L$*

*Apply the boundary condition*

*Solve the linear system  $AU = b$  for  $U$  and store in  $u^1$*

$t \leftarrow t + \Delta t$

$u^0 \leftarrow u^1$  (get ready for next step)

**end while**

## Test problem

We construct a test problem for which we can easily check the answer. We first define the exact solution by

$$u = 1 + x^2 + \alpha y^2 + \beta t$$

We insert this into the heat equation:

$$f = u_t - \Delta u = \beta - 2 - 2\alpha$$

The initial condition is

$$u^0 = 1 + x^2 + \alpha y^2$$

This technique is called the *method of manufactured solutions*

## Handling time-dependent expressions

We define a time-dependent expression for the boundary value:

```
alpha = 3; beta = 1.2
t = 0.0
g = Expression("1 + x[0]*x[0] + \
               alpha*x[1]*x[1] + beta*t",
               alpha=alpha, beta=beta, t=t,
               degree=2)
```

Then, we must explicitly update t and g:

```
t = 1.0
g.t = t
```

An alternative (robust) approach is to define t as a Constant:

```
t = Constant(0.0)
g = Expression("...", ..., t=t, ...)
t.assign(1.0)
# No need to update g itself
```

# Projection and interpolation

We need to project the initial value into  $V_h$ :

```
u0 = project(g, V)
```

We can also interpolate the initial value into  $V_h$ :

```
u0 = interpolate(g, V)
```



# A closer look at solve

For linear problems, this code

```
solve(a == L, u, bcs)
```

is equivalent to this

```
# Assembling a bilinear form yields a matrix
A = assemble(a)
# Assembling a linear form yields a vector
b = assemble(L)

# Applying boundary condition info to system
for bc in bcs:
    bc.apply(A, b)

# Solve Ax = b
solve(A, u.vector(), b)
```

# Implementing the variational problem

```
# Decide on a time step
dt = 0.3

# Create Functions for previous and current sol.s
u0 = project(g, V)
u1 = Function(V)

# Define the variational formulation
u = TrialFunction(V)
v = TestFunction(V)
f = Constant(beta - 2 - 2*alpha)
a = u*v*dx + dt*inner(grad(u), grad(v))*dx
L = u0*v*dx + dt*f*v*dx

# Define the boundary condition
bc = DirichletBC(V, g, "on_boundary")

# Assemble only once, before time-stepping
A = assemble(a)
```

# Implementing the time-stepping loop

```
T = 2          # Set end time
t.assign(dt)   # Solve on [0, dt] first

while t <= T:
    b = assemble(L)  # Assemble the rhs vector
    bc.apply(A, b)   # Apply boundary conditions

    # Solve linear system
    solve(A, u1.vector(), b)

    # Update time and previous solution
    t1 = float(t + dt)
    t.assign(t1)      # t := t1 + dt
    u0.assign(u1)     # u0 := u1
```

# FEniCS programming exercise: heat equation

Consider the heat equation problem:

$$\frac{\partial u}{\partial t} - \Delta u = f \quad \text{in } \Omega = [0, 1]^2 \text{ for } t > 0$$

$$u(x, t) = g(x, t) \quad \text{for } x \in \partial\Omega \text{ for } t > 0$$

$$u(x, 0) = g(x, 0) \quad \text{for } x \in \Omega$$

with

$$f = \beta - 2 - 2\alpha$$

$$g(x, t) = 1 + x_0^2 + \alpha x_1^2 + \beta t \quad (x = (x_0, x_1))$$

**Ex. 1** Compute an approximate solution at  $T = 1.8$

**Ex. 2** Compare the approximate solution to the exact solution at  $T = 1.8$ . How large is the error (in the eyenorm and in the  $L^2(\Omega)$  norm)?

**Ex. 3** Compute an approximate solution with the same set-up but on  $\Omega = [0, 1]^3 \subset \mathbb{R}^3$ .