

# FEniCS Course

## Lecture 4: Time-dependent PDEs

---

### *Contributors*

Hans Petter Langtangen

Anders Logg

André Massing



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

## Semi-discretization in space

Consider  $t$  as a parameter and formulate a

### Variational problem in space

Find for each  $t \in (0, T]$  a  $u(\cdot, t) \in V$  such that

$$\int_{\Omega} \partial_t u v \, dx + \int_{\Omega} \nabla u \cdot \nabla v \, dx = \int_{\Omega} f v \, dx \quad \forall v \in \widehat{V}$$

Short-hand notation

$$(\partial_t u, v) + a(u, v) = L(v)$$

### Discrete variational problem in space

Find for each  $t$  a  $u_h(\cdot, t) \in V_h$  such that

$$(\partial_t u_h, v_h) + a(u_h, v_h) = L(v_h) \quad \forall v_h \in \widehat{V}_h$$

## Semi-discrete system in space

Ansatz

$$u_h(t) = \sum_{j=0}^N U_j(t) \phi_j$$

Find  $[U_1(t), \dots, U_N(t)]^\top$

$$\sum_{j=1}^N \dot{U}_j(\phi_j, v_h) + \sum_{j=1}^N U_j a(\phi_j, v_h) = L(v_h)$$

or equivalently

$$\sum_{j=1}^N \dot{U}_j \underbrace{(\phi_j, \phi_i)}_{M_{ij}} + \sum_{j=1}^N U_j \underbrace{a(\phi_j, \phi_i)}_{A_{ij}} = \underbrace{L(\phi_i)}_{b_i} \quad \forall j = 1, \dots, N$$

or equivalently

$$M\dot{U}(t) + AU(t) = b(t)$$

## Semi-discrete system in space – part II

Find  $[U_1(t), \dots, U_N(t)]^\top$  such that

$$M\dot{U}(t) + AU(t) = b(t)$$

where

- $M = (\int_{\Omega} \phi_j(x)\phi_i(x))_{ij}$  is the mass matrix
- $A = (\int_{\Omega} \nabla\phi_j(x)\nabla\phi_i(x))_{ij}$  is the stiffness matrix
- $b(t) = (\int_{\Omega} f(t, x)\phi_i(x) dx)_i$  is the load vector

⇒ System of ordinary differential equations

**Note:**  $A$  and  $M$  are time-independent!

# A full discretization scheme: The $\theta$ -method

For  $0 \leq \theta \leq 1$  and  $U^k$  known from the previous time-step, compute  $U^{k+1}$  by solving

$$M \frac{U^{k+1} - U^k}{\Delta t} + A[\theta U^{k+1} + (1 - \theta)U^k] = \theta b^{k+1} + (1 - \theta)b^k$$

- First-order *explicit/forward Euler* for  $\theta = 0$ :

$$M \frac{U^{k+1} - U^k}{\Delta t} + AU^k = b^k$$

- First-order *implicit/backward Euler* for  $\theta = 1$ :

$$M \frac{U^{k+1} - U^k}{\Delta t} + AU^{k+1} = b^{k+1}$$

- Second-order *Crank-Nicolson* for  $\theta = 1/2$ :

$$M \frac{U^{k+1} - U^k}{\Delta t} + \frac{1}{2}A(U^{k+1} + U^k) = \frac{1}{2}(b^{k+1} + b^k)$$

# A full discretization scheme: The $\theta$ -method

For  $0 \leq \theta \leq 1$  and  $U^k$  known from the previous time-step, compute  $U^{k+1}$  by solving

$$M \frac{U^{k+1} - U^k}{\Delta t} + A[\theta U^{k+1} + (1 - \theta)U^k] = \theta b^{k+1} + (1 - \theta)b^k$$

- First-order *explicit/forward Euler* for  $\theta = 0$ :

$$M \frac{U^{k+1} - U^k}{\Delta t} + AU^k = b^k$$

- First-order *implicit/backward Euler* for  $\theta = 1$ :

$$M \frac{U^{k+1} - U^k}{\Delta t} + AU^{k+1} = b^{k+1}$$

- Second-order *Crank-Nicolson* for  $\theta = 1/2$ :

$$M \frac{U^{k+1} - U^k}{\Delta t} + \frac{1}{2}A(U^{k+1} + U^k) = \frac{1}{2}(b^{k+1} + b^k)$$

# A full discretization scheme: The $\theta$ -method

For  $0 \leq \theta \leq 1$  and  $U^k$  known from the previous time-step, compute  $U^{k+1}$  by solving

$$M \frac{U^{k+1} - U^k}{\Delta t} + A[\theta U^{k+1} + (1 - \theta)U^k] = \theta b^{k+1} + (1 - \theta)b^k$$

- First-order *explicit/forward Euler* for  $\theta = 0$ :

$$M \frac{U^{k+1} - U^k}{\Delta t} + AU^k = b^k$$

- First-order *implicit/backward Euler* for  $\theta = 1$ :

$$M \frac{U^{k+1} - U^k}{\Delta t} + AU^{k+1} = b^{k+1}$$

- Second-order *Crank-Nicolson* for  $\theta = 1/2$ :

$$M \frac{U^{k+1} - U^k}{\Delta t} + \frac{1}{2}A(U^{k+1} + U^k) = \frac{1}{2}(b^{k+1} + b^k)$$



# A full discretization scheme: The $\theta$ -method

For  $0 \leq \theta \leq 1$  and  $U^k$  known from the previous time-step, compute  $U^{k+1}$  by solving

$$M \frac{U^{k+1} - U^k}{\Delta t} + A[\theta U^{k+1} + (1 - \theta)U^k] = \theta b^{k+1} + (1 - \theta)b^k$$

- First-order *explicit/forward Euler* for  $\theta = 0$ :

$$M \frac{U^{k+1} - U^k}{\Delta t} + AU^k = b^k$$

- First-order *implicit/backward Euler* for  $\theta = 1$ :

$$M \frac{U^{k+1} - U^k}{\Delta t} + AU^{k+1} = b^{k+1}$$

- Second-order *Crank-Nicolson* for  $\theta = 1/2$ :

$$M \frac{U^{k+1} - U^k}{\Delta t} + \frac{1}{2}A(U^{k+1} + U^k) = \frac{1}{2}(b^{k+1} + b^k)$$

# Implementation of the implicit Euler method

## Recast as a discrete variational problem

First-order *implicit/backward Euler* for  $\theta = 1$ :

$$\left(\frac{M}{\Delta t} + A\right)U^{k+1} = \frac{M}{\Delta t}U^k + b^{k+1}$$

can be reformulated as: Find  $u_h^{k+1} \in V_h$  such that

$$(u_h^{k+1}, v_h) + \Delta t a(u_h^{k+1}, v_h) = (u_h^k, v_h) + \Delta t (f^{k+1}, v_h) \quad \forall v_h \in \widehat{V}_h$$

**Exercise:** Find the corresponding variational problems for the explicit Euler and Crank-Nicolson schemes

## Initial condition

$$u_h^0 \approx u^0$$

Choose  $L^2$ -projection  $\Pi_h u^0$  on  $V_h$  or interpolation  $I_h(u^0)$

# Time-stepping algorithm

*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**

## Method of manufactured solutions

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

$$u(x, y, t) = e^{-4\pi^2 t} \cos(2\pi x) \cos(2\pi y)$$

We compute

$$\begin{aligned}\partial_t u(x, y, t) &= -4\pi^2 e^{-4\pi^2 t} \cos(2\pi x) \cos(2\pi y) \\ -\Delta u(x, y, t) &= +8\pi^2 e^{-4\pi^2 t} \cos(2\pi x) \cos(2\pi y)\end{aligned}$$

So we have to find  $u$  such that

$$\begin{aligned}(\partial_t - \Delta)u(x, y, t) &= +4\pi^2 e^{-4\pi^2 t} \cos(2\pi x) \cos(2\pi y) \quad \text{in } \Omega \times (0, T] \\ u(x, y, t) &= e^{-4\pi^2 t} \cos(2\pi x) \cos(2\pi y) \quad \text{on } \partial\Omega \times (0, T] \\ u(x, y, 0) &= \cos(2\pi x) \cos(2\pi y) \quad \text{on } \Omega \times \{0\}\end{aligned}$$

**Our mission:** Solve this problem choosing  $T = 0.1$ , a fixed time-step  $\Delta t = 0.001$  and using the implicit Euler method. Visualise  $u$ ,  $u_h$  and  $u - u_h$ .

## Handling time-dependent expressions

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

```
# Start time
t0 = 0
g = Expression("exp(-4*DOLFIN_PI*DOLFIN_PI*t) \
               *cos(2*DOLFIN_PI*x[0]) \
               *cos(2*DOLFIN_PI*x[1])", t=t0)

f = Expression("4*DOLFIN_PI*DOLFIN_PI \
               *exp(-4*DOLFIN_PI*DOLFIN_PI*t) \
               *cos(2*DOLFIN_PI*x[0]) \
               *cos(2*DOLFIN_PI*x[1])", t=t0)
```

Updating parameter values:

```
g.t = t
f.t = t
```

# Projection and interpolation

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

$$u_0 = \text{project}(g, V)$$

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

$$u_0 = \text{interpolate}(g, V)$$

# Implementing the variational problem

```
u0 = interpolate(g,V)

u = TrialFunction(V)
v = TestFunction(V)

# time step
dt = 0.001

# Define variational forms
a = u*v*dx + dt*inner(grad(u),grad(v))*dx
L = u0*v*dx + dt*f*v*dx

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

# Implementing the time-stepping loop

```
u1 = Function(V)
T = 0.1
t = dt

while t <= T:
    g.t = t
    f.t = t

    b = assemble(L)

    bc.apply(A, b)
    solve(A, u1.vector(), b)

    t += dt
    u0.assign(u1)
```



Let's start!

## The FEniCS homework!

- Implement the explicit/forward Euler scheme and the Crank-Nicolson scheme. Compute the numerical solutions and repeat the post-processing steps.
- What do you observe when you use the explicit/forward Euler scheme? Why?
- Repeat the computation for a  $N = 10$  and  $\Delta t = 0.0001$  for the explicit Euler method. What happens if you now increase  $N$  again?