MATH 676

–

Finite element methods in scientific computing

Wolfgang Bangerth, Texas A&M University

Lecture 29:

The *step-26* tutorial program
–
The heat equation

Part 1: The basics

step-26

Step-26 shows:

- The time and space discretization of the heat equation
- The typical structure of a code for time dependent problems

The program

Step-26 solves the heat equation:

$$\begin{aligned} \frac{\partial u}{\partial t} - \Delta u &= f \quad \text{in } \Omega \times [0,T] \\ u &= g \quad \text{on } \partial\Omega \times [0,T] \end{aligned}$$

$$u(x,0) = u_0(x) \quad \text{in } \Omega$$

Let us use the Rothe method to discretize it:

- **Step 1:** Introduce $u^n(x)$ to approximate $u(x,t_n)$
- **Step 2:** Decide what to do with the time derivative
- **Step 3:** Decide what to do with all the other $u(x,t)$, $f(x,t)$
- **Step 4:** Spatial discretization

Time discretization

Step-26 solves the heat equation:

$$\frac{\partial u}{\partial t} - \Delta u = f \quad \text{in } \Omega$$

Step 2: Decide what to do with the time derivative

Approach: The simplest time stepping methods use

$$\frac{\partial u}{\partial t} \rightarrow \frac{u^n - u^{n-1}}{k_n}$$

$$\text{where} \quad k_n := t_n - t_{n-1}$$

Time discretization

We now have:

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

Step 3: Decide what to do with the other *u(x,t), f(x,t)*

Approach 1: Implicit Euler

$$\frac{u^n - u^{n-1}}{k_n} - \Delta u^n \quad = \quad f\left(x, t_n\right)$$

Note 1: This is first order accurate in time.
Note 2: Explicit methods are not useful for this equation.

Time discretization

We now have:

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

Step 3: Decide what to do with the other *u(x,t), f(x,t)*

Approach 2: Crank-Nicolson

$$\frac{u^n - u^{n-1}}{k_n} - \Delta\left(\frac{1}{2}u^{n-1} + \frac{1}{2}u^n\right) = \frac{1}{2}f\left(x, t_{n-1}\right) + \frac{1}{2}f\left(x, t_n\right)$$

Note: This is second order accurate in time.

Time discretization

We now have:

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

Step 3: Decide what to do with the other *u(x,t), f(x,t)*

Approach 3: The *theta*-scheme

$$\frac{u^n - u^{n-1}}{k_n} - \Delta\left((1-\theta)u^{n-1} + \theta u^n\right) = (1-\theta)f(x,t_{n-1}) + \theta f(x,t_n)$$

Note: This generalizes explicit Euler (Θ=0), implicit Euler (Θ=1), and Crank-Nicolson (Θ=1/2).

Time discretization

We now have:

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

Step 3: Decide what to do with the other *u(x,t), f(x,t)*

Approach n: There are a million more ways to set up time discretizations.

Other popular ones for PDEs are

- BDF-2, BDF-3
- Low-order Runge-Kutta methods (maybe up to RK4)

Note: High-order methods are rarely used for PDEs.

Spatial discretization

We now have:

$$\frac{u^n - u^{n-1}}{k_n} - \Delta\left((1-\theta)u^{n-1} + \theta u^n\right) = (1-\theta)f(x,t_{n-1}) + \theta f(x,t_n)$$

Step 4: Spatial discretization works as usual:

– Write the discrete solution as

$$u^n(x) \approx u_h^n(x) := \sum_j U_j^n \varphi_j(x)$$

– Multiply from the left with a test function, integrate
– Obtain a linear system at each time step:

$$M\frac{U^n - U^{n-1}}{k_n} + A\left((1-\theta)U^{n-1} + \theta U^n\right) = (1-\theta)F^{n-1} + \theta F^n$$

Time stepping

We now have:

$$M\frac{U^n-U^{n-1}}{k_n}+A\left((1-\theta)U^{n-1}+\theta U^n\right) = (1-\theta)F^{n-1}+\theta F^n$$

We already know U^{n-1} from the previous step. In time step n we then need to solve:

$$(M+k_n\theta A)U^n = MU^{n-1}-k_n(1-\theta)AU^{n-1}+k_n(1-\theta)F^{n-1}+\theta k_n F^n$$

Let us look at this linear system in a couple of different ways!

The linear system

In each time step we have to solve:

$$\left(M + k_n \theta A\right) U^n \quad = \ \ldots$$

Notes about the matrix, part 1:

- *M,A* are both symmetric, positive semidefinitive
- *M* is in fact positive *definite*
- The sum of the matrices is therefore also SPD

Result: We can use CG as a solver, and one of the usual preconditioners. (step-26 uses SSOR.)

The linear system

In each time step we have to solve:

$$\left(M+k_n\theta A\right)U^n \;=\; \ldots$$

Notes about the matrix, part 2:

- Eigenvalues of M are clustered around h^d
- Eigenvalues of A is

$$\lambda_{\min}\left(A\right) \;\approx\; h^d\pi^2, \qquad \lambda_{\max}\left(A\right) \;\approx\; h^d\frac{\pi^2}{h^2}$$

- The eigenvalues of the system matrix are

$$\lambda_{\min}\left(M+k_n\theta A\right) \;\approx\; \left(1+k_n\theta\right)h^d\pi^2, \qquad \lambda_{\max}\left(M+k_n\theta A\right) \;\approx\; h^d\left(1+k_n\theta\frac{\pi^2}{h^2}\right)$$

Result: This is not usually a badly conditioned system!

The linear system

In each time step we have to solve:

$$\left(M+k_n\theta A\right)U^n \;=\; M U^{n-1}-k_n\left(1-\theta\right)A U^{n-1}+k_n\left(1-\theta\right)F^{n-1}+k_n\theta F^n$$

Notes about the right hand side:

- We could implement this as

```
system_rhs = mass_matrix * old_solution
             - time_step * (1-theta) * laplace_matrix * old_solution
             + time_step * (1-theta) * old_forcing_rhs
             + time_step * theta * forcing_rhs;
```

  with appropriately overloaded *operator**

- However: *This would be very inefficient!*

Digression on how to build up a vector

Consider this piece of code:

```
system_rhs = mass_matrix * old_solution
                  - time_step * (1-theta) * laplace_matrix * old_solution
                  + time_step * (1-theta) * old_forcing_rhs
                  + time_step * theta * forcing_rhs;
```

- When computing *mass_matrix*old_solution*, compiler has to allocate memory for the result vector
- (At least) Three more times for the second line
- The one more to compute the sum of the previous two
- Four more times for the rest
- Then assign the result to the left hand side

Result: 9 allocations/deallocations of vectors!

Digression on how to build up a vector

Consider this piece of code:

```
system_rhs = mass_matrix * old_solution
             - time_step * (1-theta) * laplace_matrix * old_solution
             + time_step * (1-theta) * old_forcing_rhs
             + time_step * theta * forcing_rhs;
```

A better way (only 1 allocation/deallocation):

```
system_rhs = mass_matrix * old_solution;
tmp        = laplace_matrix * old_solution;
system_rhs.add (- time_step * (1-theta), tmp);
forcing_term = F^n;
forcing_term *= time_step * theta;
tmp          = F^(n-1);
tmp          *= time_step * (1-theta);
forcing_term += tmp;
system_rhs   += forcing_terms;
```

The linear system

In each time step we have to solve:

$$\left(M + k_n \theta A\right) U^n \quad = \ \ldots$$

Notes about the matrix, part 3:

- We also have boundary values
- *MatrixTools::apply_boundary_values* needs matrix entries to modify the right hand side
- However, it also modifies the matrix

Consequence: We need to put together and modify the matrix *in every time step*. But we can store *M,A*.

MATH 676

–

Finite element methods in scientific computing

Wolfgang Bangerth, Texas A&M University