Chapter 4

FD methods for linear parabolic PDE

A linear PDE of the form

\[ u_t = Lu, \tag{4.1} \]

where \( t \) usually denotes the time and \( L \) is a linear elliptic differential operator in one or more spatial variables, is called parabolic. Furthermore, the second order canonical form

\[ a(x,t)u_{tt} + 2b(x,t)u_{xt} + c(x,t)u_{xx} + \text{lower order terms} = f(x,t) \]

is parabolic if \( b^2 - ac = 0 \) in the entire \( x-t \) domain. Note that, we can transform this second order PDE into a system of two PDEs by setting \( v = u_t \), where the \( t \)-derivative is first order. Some important parabolic PDE are as follows.

- 1D heat equation with a source

\[ u_t = u_{xx} + f(x,t). \]

The dimension refers to the space variable (\( x \) direction).

- General heat equation

\[ u_t = \nabla \cdot (\beta \nabla u) + f(x,t) , \tag{4.2} \]

where \( \beta \) is the diffusion coefficient and \( f(x,t) \) is the source (or sink) term.

- Diffusion-advection equation

\[ u_t = \nabla \cdot (\beta \nabla u) + \mathbf{w} \cdot \nabla u + f(x,t) , \]

where \( \nabla \cdot (\beta \nabla u) \) is the diffusion term and \( \mathbf{w} \cdot \nabla u \) the advection term.
• Canonical form of diffusion-reaction equation

\[ u_t = \nabla \cdot (\beta \nabla u) + f(x, t, u). \]

The nonlinear source term \( f(x, t, u) \) is the reaction.

The steady state solutions (when \( u_t = 0 \)) are the solutions of the corresponding elliptic PDEs, i.e.,

\[ \nabla \cdot (\beta \nabla u) + \bar{f}(x, u) = 0 \]

for the last case, assuming \( \lim_{t \to \infty} f(x, t, u) = \bar{f}(x, u) \) exists.

Initial and boundary conditions

In time-dependent problems, there is an initial condition that is usually specified at \( t = 0 \) i.e., \( u(x, 0) = u_0(x) \) for the above PDE, in addition to relevant boundary conditions. If the initial condition is given at \( t = T \neq 0 \), it can of course be rendered at \( t = 0 \) by a translation \( t' = t - T \). Thus for the 1D heat equation \( u_t = u_{xx} \) on \( a < x < b \) for example, we expect to have an initial condition at \( t = 0 \) in addition to boundary conditions at \( x = a \) and \( x = b \) say. Note that the boundary conditions at \( t = 0 \) may or may not be consistent with the initial condition, e.g., if a Dirichlet boundary condition is prescribed at \( x = a \) and \( x = b \) such that \( u(a, t) = g_1(t) \) and \( u(b, t) = g_2(t) \), then \( u_0(a) = g_1(0) \) and \( u_0(b) = g_2(0) \) for consistency.

Dynamical stability

The fundamental solution \( u(x, t) = e^{-x^2/4t}/\sqrt{4\pi t} \) for the 1D heat equation \( u_t = u_{xx} \) is uniformly bounded. However, for the backward heat equation \( u_t = -u_{xx} \), if \( u(x, 0) \neq 0 \) then \( \lim_{t \to \infty} u(x, t) = \infty \). The solution is said to be dynamically unstable if it is not uniformly bounded, i.e., if there is no constant \( C > 0 \) such that \( |u(x, t)| \leq C \). Some applications are dynamically unstable and “blow up”, but we do not discuss how to solve such dynamically unstable problems in this book, i.e., we only consider the numerical solution of dynamically stable problems.

Some commonly used FD methods

We discuss the following FD methods for parabolic PDE in this chapter:

• the forward and backward Euler methods;
• the Crank-Nicolson and \( \theta \) methods;
• the method of lines (MOL), provided a good ODE solver can be applied; and
FD methods for linear parabolic PDE

- the alternating directional implicit (ADI) method, for high-dimensional problems.

FD methods applicable to elliptic PDE can be used to treat the spatial discretization and boundary conditions, so let us focus on the time discretization and initial condition(s). To consider the stability of the consequent numerical methods, we invoke a Fourier transformation and von Neumann stability analysis.

\[ k+1 \quad k \]
\[ FW-CT \quad BW-CT \]
\[ t=0 \]
\[ MOL \quad IC \quad BC \quad BC \]

Figure 4.1: Diagram of the FD stencil for the forward and backward Euler methods, and the method of lines (MOL).

4.1 The Euler method

For the following problem involving the heat equation with a source term,

\[ u_t = \beta u_{xx} + f(x,t), \quad a < x < b, \quad t > 0, \]
\[ u(a,t) = g_1(t), \quad u(b,t) = g_2(t), \quad u(x,0) = u_0(x), \]

let us seek a numerical solution for \( u(x,t) \) at a particular time \( T > 0 \) or at certain times in the interval \( 0 < t < T \).

As the first step, we expect to generate a grid

\[ x_i = a + ih, \quad i = 0, 1, \ldots, m, \quad h = \frac{b-a}{m}, \]
\[ t^k = k\Delta t, \quad k = 0, 1, \ldots, n, \quad \Delta t = \frac{T}{n}. \]

It turns out that any arbitrary \( \Delta t \) cannot be used for explicit methods because of numerical instability concerns. The second step is to approximate the derivatives with FD approximations. Since we already know how to discretize the spatial derivatives, let us focus on possible FD formulas for the time derivative.
4.1.1 Forward Euler method (FT-CT)

At a grid point \((x_i, t^k), k > 0\), on using the forward FD approximation for \(u_t\) and central FD approximation for \(u_{xx}\) we have

\[
\frac{u(x_i, t^{k+1}) - u(x_i, t^k)}{\Delta t} = \beta \frac{u(x_{i-1}, t^k) - 2u(x_i, t^k) + u(x_{i+1}, t^k)}{h^2} + f(x_i, t^k) + T(x_i, t^k) .
\]

The local truncation error is

\[
T(x_i, t^k) = \frac{h^2 \beta}{12} u_{xxxx}(x_i, t^k) + \frac{\Delta t}{2} u_{tt}(x_i, t^k) + \cdots ,
\]

where the dots denote higher order terms, so the discretization is \(O(h^2 + \Delta t)\). The discretization is first order in time and second order in space, when the FD equation is

\[
\frac{U_{i}^{k+1} - U_{i}^{k}}{\Delta t} = \beta \frac{U_{i-1}^{k} - 2U_{i}^{k} + U_{i+1}^{k}}{h^2} + f_i^k \tag{4.1}
\]

where \(f_i^k = f(x_i, t^k)\), with \(U_i^k\) again denoting the approximate values for the true solution \(u(x_i, t^k)\). When \(k = 0\), \(U_i^0\) is the initial condition at the grid point \((x_i, 0)\); and from the values \(U_i^k\) at the time level \(k\) the solution of the FD equation at the next time level \(k + 1\) is

\[
U_{i}^{k+1} = U_{i}^{k} + \Delta t \left( \beta \frac{U_{i-1}^{k} - 2U_{i}^{k} + U_{i+1}^{k}}{h^2} + f_i^k \right) , \quad i = 1, 2, \cdots, m - 1 . \tag{4.2}
\]

The solution of the FD equations is thereby directly obtained from the approximate solution at previous time steps and we do not need to solve a system of algebraic equations, so the method is called \textit{explicit}. Indeed, we successively compute the solution at \(t^1\) from the initial condition at \(t^0\), and then at \(t^2\) using the approximate solution at \(t^1\). Such an approach is often called a time marching method.

\textbf{Remark 4.1} The local truncation error under our definition is

\[
T(x, t) = \frac{u(x, t + \Delta t) - u(x, t)}{\Delta t} - \beta \frac{u(x - h, t) - 2u(x, t) + u(x + h, t)}{h^2} - f(x, t) = O(h^2 + \Delta t) .
\]

In passing, we note an alternative definition of the truncation error in the literature

\[
T(x, t) = u(x, t + \Delta t) - u(x, t) - \Delta t \left( \beta \frac{u(x - h, t) - 2u(x, t) + u(x + h, t)}{h^2} - f(x, t) \right) = O(\Delta t(h^2 + \Delta t))
\]

introduces an additional factor \(\Delta t\), so it is one order higher in \(\Delta t\).
FD methods for linear parabolic PDE

Remark 4.2 If \( f(x,t) \equiv 0 \) and \( \beta \) is a constant, then from \( u_t = \beta u_{xx} \) and \( u_{tt} = \beta \partial^2 u_t / \partial x^2 = \beta^2 u_{xxxx} \), the local truncation error is

\[
T(x,t) = \left( \frac{\beta^2 \Delta t}{2} - \frac{\beta h^2}{12} \right) u_{xxxx} + O((\Delta t)^2 + h^4). \tag{4.3}
\]

Thus if \( \beta \) is constant we can choose \( \Delta t = h^2 / (6 \beta) \) to get \( O(h^4 + (\Delta t)^2) = O(h^4) \), i.e., the local truncation error is fourth order accurate without further computational complexity, which is significant for an explicit method.

On testing the forward Euler method with different \( \Delta t \) and checking the error in a problem with a known exact solution, we find the method works for some values of \( \Delta t \) but blows up for some others. Since the method is consistent, we anticipate that this is a question of numerical stability. Intuitively, to prevent the errors in \( u^k_i \) being amplified, one can set

\[
\frac{2\beta \Delta t}{h^2} \leq 1, \quad \text{or} \quad \Delta t \leq \frac{h^2}{2\beta}. \tag{4.4}
\]

This is a time step constraint, often called the CFL (Courant-Friedrichs-Lewy) stability condition.

4.1.2 The backward Euler method (BW-CT)

If the backward FD formula is used for \( u_t \) and the central FD approximation for \( u_{xx} \) at \((x_i, t^k)\), we get

\[
\frac{U^k_i - U^{k-1}_i}{\Delta t} = \beta \frac{U^{k-1}_{i-1} - 2U^{k}_i + U^{k+1}_{i+1}}{h^2} + f^k_i, \quad k = 1, 2, \cdots,
\]

which is conventionally re-expressed as

\[
\frac{U^{k+1}_i - U^k_i}{\Delta t} = \beta \frac{U^{k+1}_{i-1} - 2U^{k+1}_i + U^{k+1}_{i+1}}{h^2} + f^{k+1}_i, \quad k = 0, 1, \cdots. \tag{4.5}
\]

The backward Euler method is also consistent, and the discretization error is again \( O(\Delta t + h^2) \).

Using the backward Euler method, we cannot get \( U^{k+1}_i \) with a few simple algebraic operations because all of the \( U^{k+1}_i \)'s are coupled together. Thus we need to solve the following tridiagonal system of equations, in order to get the approximate solution at the
time level $k + 1$:

$$
\begin{bmatrix}
1 + 2\mu & -\mu \\
-\mu & 1 + 2\mu & -\mu \\
-\mu & 1 + 2\mu & -\mu \\
& \ddots & \ddots & \ddots \\
-\mu & 1 + 2\mu & -\mu \\
\end{bmatrix}
\begin{bmatrix}
U_{k+1}^{1} \\
U_{k+1}^{2} \\
U_{k+1}^{3} \\
\vdots \\
U_{k+1}^{m-1} \\
\end{bmatrix}
= 
\begin{bmatrix}
U_{1}^{k} + \Delta t f_{1}^{k+1} + \mu g_{1}^{k+1} \\
U_{2}^{k} + \Delta t f_{2}^{k+1} \\
U_{3}^{k} + \Delta t f_{3}^{k+1} \\
\vdots \\
U_{m-2}^{k} + \Delta t f_{m-2}^{k+1} \\
U_{m-1}^{k} + \Delta t f_{m-1}^{k+1} + \mu g_{2}^{k+1} \\
\end{bmatrix},
$$

(4.6)

where $\mu = \frac{\beta \Delta t}{h^2}$ and $f_{i}^{k+1} = f(x_i, t^{k+1})$. Note that we can use $f(x_i, t^{k})$ instead of $f(x_i, t^{k+1})$, since the method is first order accurate in time. Such a numerical method is called an implicit, because the solution at time level $k + 1$ are coupled together. The advantage of the backward Euler method is that it is stable for any choice of $\Delta t$. For 1D problems, the computational cost is only slightly more than the explicit Euler method if we can use an efficient tridiagonal solver, such as the Grout factorization method at cost $O(5n)$, cf. Ref. [?], for example.

### 4.2 The method of lines (MOL)

With a good solver for ODE or systems of ODE, we can use the method of lines (MOL) to solve parabolic partial differential equations. In Matlab, we can use the ODE Suite to solve a system of ordinary differential equations. The ODE Suite contains Matlab functions such as ode23, ode23s, ode15s, ode45 and others. The Matlab function ode23 uses a combination of Runge-Kutta methods of order 2 and 3 and uses an adaptive time step size. The Matlab function ode23s is designed for a stiff system of ODE.

Consider a general parabolic equation of the form

$$u_t(x, t) = Lu(x, t) + f(x, t),$$

where $L$ is an elliptic operator. Let $L_h$ be a corresponding FD operator acting on a grid
FD methods for linear parabolic PDE

$x_i = a + ih$. We can form a semi-discrete system of ordinary differential equations of form

$$\frac{\partial U_i}{\partial t} = L_h U_i(t) + f_i(t),$$

where $U_i(t) \approx u(x_i, t)$ is the spatial discretization of $u(x, t)$ along the line $x = x_i$, i.e., we only discretize the spatial variable. For example, the heat equation with a source $u_t = \beta u_{xx} + f$ where $L = \beta \partial^2 / \partial x^2$ is represented by $L_h = \beta \delta_{xx}$ produces the discretized system of ODE

$$\frac{\partial U_1(t)}{\partial t} = \beta -2U_1(t) + U_2(t) + \frac{g_1(t)}{h^2} + f(x_1, t),$$
$$\frac{\partial U_i(t)}{\partial t} = \beta U_{i-1}(t) - 2U_2(t) + U_{i+1}(t) + f(x_i, t), \quad i = 2, 3, \ldots, m - 2,$$
$$\frac{\partial U_{m-1}(t)}{\partial t} = \beta U_{m-2}(t) - 2U_{i-1}(t) + \frac{g_2(t)}{h^2} + f(x_{m-1}, t),$$

and the initial condition is

$$U_i(0) = u_0(x_i, 0), \quad i = 1, 2, \ldots, m - 1.$$  \hfill (4.2)

The ODE system can be written in the vector form

$$\frac{dy}{dt} = f(y, t), \quad y(0) = y_0.$$  \hfill (4.3)

The MOL is especially useful for nonlinear PDE of the form $u_t = f(\partial / \partial x, u, t)$. For linear problems, we typically have

$$\frac{dy}{dt} = Ay + c,$$

where $A$ is a matrix and $c$ is a vector. Both $A$ and $c$ may depend on $t$.

There are many efficient solvers for a system of ODE. Most are based on high order Runge-Kutta methods with adaptive time steps, e.g., ODE suite in Matlab, or dsode.f available through Netlib. However, it is important to recognise that the ODE system obtained from the MOL is typically stiff, i.e., the eigenvalues of $A$ have very different scales. For example, for the heat equation the magnitude of the eigenvalues range from $O(1)$ to $O(1/h^2)$. In Matlab, we can call an ODE solver using the format

$$[t, y] = \text{ode23s}('yfun-mol', [0, t_{final}], y0);$$

The solution is stored in the last row of $y$, which can be extracted using

$$[\text{mr}, \text{nc}] = \text{size}(y);$$
$$\text{ysol} = y(\text{mr},:);$$

Then $\text{ysol}$ is the approximate solution at time $t = t_{final}$. To define the ODE system of the MOL, we can create a Matlab file, say yfun-mol.m whose contents contain the following
function yp = yfun-mol(t,y)
global m h x
k = length(y); yp=size(k,1);
yp(1) = (-2*y(1) + y(2))/(h*h) + f(t,x(1)) + g1(t)/(h*h);
for i=2:m-2
    yp(i) = (y(i-1) -2*y(1) + y(2))/(h*h) + f(t,x(i));
end
yp(m-1) = (y(m-2) -2*y(m-1) )/(h*h) + f(t,x(i)) + g2(t)/(h*h);

where g1(t) and g2(t) are two matlab functions for the boundary conditions at x = a and x = b; and f(t,x) is the source term.

The initial condition can be defined as

    global m h x
    for i=1:m-1
        y0(i) = u_0(x(i));
    end

where u0(x) is a Matlab function of the initial condition.

### 4.3 The Crank-Nicolson scheme

The time step constraint \( \Delta t = h^2/(2\beta) \) for the explicit Euler method is generally considered to be a severe restriction, e.g., if \( h = 0.01 \), the final time is \( T = 10 \) and \( \beta = 100 \), then we need \( 2 \times 10^7 \) steps to get the solution at the final time. The backward Euler method does not have the time step constraint, but it is only first order accurate. If we want second order accuracy \( O(h^2) \), we need to take \( \Delta t = O(h^2) \). One FD scheme that is second order accurate both in space and time, without compromising stability and computational complexity, is the Crank-Nicolson scheme.

The Crank-Nicolson scheme is based on the following lemma, which can be proved easily using the Taylor expansion.

**Lemma 4.1** Let \( \phi(t) \) be a function that has continuous first and second order derivatives, i.e., \( \phi(t) \in C^2 \). Then

\[
\phi(t) = \frac{1}{2} \left( \phi(t - \frac{\Delta t}{2}) + \phi(t + \frac{\Delta t}{2}) \right) + \frac{(\Delta t)^2}{8} u''(t) + h.o.t. \tag{4.1}
\]

Intuitively, the Crank-Nicolson scheme approximates the PDE

\[
u_t = (\beta u_x)_x + f(x,t)
\]
FD methods for linear parabolic PDE

at \((x_i, t^k + \Delta t/2)\), by averaging the time level \(t^k\) and \(t^{k+1}\) of the spatial derivative \(\nabla \cdot (\beta \nabla u)\) and \(f(x, t)\). Thus it has the following form

\[
\frac{U_i^{k+1} - U_i^k}{\Delta t} = \frac{\beta_{i-\frac{1}{2}}^{k} U_{i-1}^k - (\beta_{i-\frac{1}{2}}^{k} + \beta_{i+\frac{1}{2}}^{k}) U_i^k + \beta_{i+\frac{1}{2}}^{k} U_{i+1}^k}{2h^2} \\
+ \frac{\beta_{i-\frac{1}{2}}^{k+1} U_{i-1}^{k+1} - (\beta_{i-\frac{1}{2}}^{k+1} + \beta_{i+\frac{1}{2}}^{k+1}) U_i^{k+1} + \beta_{i+\frac{1}{2}}^{k+1} U_{i+1}^{k+1}}{2h^2} + \frac{1}{2} \left(f_i^k + f_i^{k+1}\right). \tag{4.2}
\]

The discretization is second order in time (central at \(t + \Delta t/2\) with step size \(\Delta t/2\)) and second order in space. This can easily be seen using the following relations, taking \(\beta = 1\) for simplicity:

\[
u(x, t + \Delta t) - u(x, t) \over \Delta t = u_t(x, t + \Delta t/2) + \frac{1}{3} \left(\frac{\Delta t}{2}\right)^2 + O((\Delta t)^4),
\]

\[
u(x - h, t) - 2u(x, t) + u(x + h, t) \over 2h^2 = u_{xx}(x, t) + O(h^2),
\]

\[
u(x - h, t + \Delta t) - 2u(x, t + \Delta t) + u(x + h, t + \Delta t) \over 2h^2 = u_{xx}(x, t + \Delta t) + O(h^2),
\]

\[
\frac{1}{2} \left(u_{xx}(x, t) + u_{xx}(x, t + \Delta t)\right) = u_{xx}(x, t + \Delta t/2) + O((\Delta t)^2),
\]

\[
\frac{1}{2} \left(f(x, t) + f(x, t + \Delta t)\right) = f(x, t + \Delta t/2) + O((\Delta t)^2).
\]

At each time step, we need to solve a tridiagonal system of equations to get \(U_i^{k+1}\). The computational cost is only slightly more than the explicit Euler method in one space dimension, and we can take \(\Delta t \approx h\) and have second order accuracy. Although the Crank-Nicolson scheme is an implicit method, it is much more efficient than the explicit Euler method since it is second order accurate both in time and space with the same computational complexity.

In the next section, we will prove it is unconditionally stable for the heat equation.

The \(\theta\)-method

The \(\theta\)-method for the heat equation \(u_t = u_{xx} + f(x, t)\) has the following form:

\[
\frac{U_i^{k+1} - U_i^k}{\Delta t} = \theta \delta_{xx}^2 U_i^k + (1 - \theta) \delta_{xx}^2 U_i^{k+1} + \theta f_i^k + (1 - \theta) f_i^{k+1}.
\]

When \(\theta = 1\), the method is the explicit Euler method; when \(\theta = 0\), the method is the backward Euler method; and when \(\theta = 1/2\), it is the Crank-Nicolson scheme. If \(0 < \theta \leq 1/2\), then the method is unconditionally stable, and otherwise it is conditionally stable, i.e., there is a time step constraint. The \(\theta\)-method is generally first order in time and second order in space, except when \(\theta = 1/2\).
4.3.1 The backward Euler method (BW-CT) in 2D

The backward Euler scheme can be written as
\[
\frac{U_{ij}^{k+1} - U_{ij}^k}{\Delta t} = \frac{U_{i-1,j}^{k+1} + U_{i+1,j}^{k+1} + U_{i,j-1}^{k+1} + U_{i,j+1}^{k+1} - 4U_{ij}^{k+1}}{h^2} + f_{ij}^{k+1},
\]
which is first order in time and second order in space, and it is unconditionally stable. The coefficient matrix for the unknown \(U_{ij}^{k+1}\) is block tridiagonal, and strictly row diagonally dominant if the natural row ordering is used to index the \(U_{ij}^{k+1}\) and the FD equations.

4.3.2 The Crank-Nicolson (C-N) scheme in 2D

The Crank-Nicolson scheme can be written as
\[
\frac{U_{ij}^{k+1} - U_{ij}^k}{\Delta t} = \frac{1}{2} \left( \frac{U_{i-1,j}^{k+1} + U_{i+1,j}^{k+1} + U_{i,j-1}^{k+1} + U_{i,j+1}^{k+1} - 4U_{ij}^{k+1}}{h^2} + f_{ij}^{k+1} \right)
\]
\[
+ \frac{U_{i-1,j}^k + U_{i+1,j}^k + U_{i,j-1}^k + U_{i,j+1}^k - 4U_{ij}^k}{h^2} + f_{ij}^k \right)\). (4.4)
\]
Both the local truncation error and global error are \(O((\Delta t)^2 + h^2)\). The scheme is unconditionally stable for linear problems. However, we need to solve a system of equations with a strictly row diagonally dominant and block tridiagonal coefficient matrix, if we use the natural row ordering.

A structured multigrid method can be applied to solve the linear system of equations from the backward Euler method or the Crank-Nicolson scheme.

4.4 The alternating directional Implicit (ADI) method

The ADI is a time splitting or fractional step method. The idea is to use an implicit discretization in one direction and an explicit discretization in another direction. For the heat equation \(u_t = u_{xx} + u_{yy} + f(x,y,t)\), the ADI method is
\[
\frac{U_{ij}^{k+1/2} - U_{ij}^k}{(\Delta t)/2} = \frac{U_{i,j+1}^{k+1/2} - 2U_{ij}^{k+1/2} + U_{i,j-1}^{k+1/2}}{h_x^2} + \frac{U_{i+1,j}^{k+1/2} - 2U_{ij}^{k+1/2} + U_{i-1,j}^{k+1/2}}{h_y^2} + f_{ij}^{k+1/2},
\]
\[
\frac{U_{ij}^{k+1} - U_{ij}^{k+1/2}}{(\Delta t)/2} = \frac{U_{i,j+1}^{k+1} - 2U_{ij}^{k+1} + U_{i,j-1}^{k+1}}{h_x^2} + \frac{U_{i+1,j}^{k+1} - 2U_{ij}^{k+1} + U_{i-1,j}^{k+1}}{h_y^2} + f_{ij}^{k+1/2},
\]
which is second order in time and in space if \(u(x,y,t) \in C^4\). It is unconditionally stable for linear problems. We can use symbolic expressions to discuss the method, rewritten as
\[
U_{ij}^{k+1/2} = U_{ij}^k + \frac{\Delta t}{2} h_x^2 U_{ij}^{k+1/2} + \frac{\Delta t}{2} h_y^2 U_{ij}^k + \frac{\Delta t}{2} f_{ij}^{k+1/2},
\]
\[
U_{ij}^{k+1} = U_{ij}^{k+1/2} + \frac{\Delta t}{2} h_x^2 U_{ij}^{k+1/2} + \frac{\Delta t}{2} h_y^2 U_{ij}^{k+1} + \frac{\Delta t}{2} f_{ij}^{k+1/2}.
\]
Thus on moving unknowns to the left-hand side, in matrix-vector form we have

\[
\begin{align*}
(I - \frac{\Delta t}{2} D_x^2) U^{k+\frac{1}{2}} &= (I + \frac{\Delta t}{2} D_y^2) U^k + \frac{\Delta t}{2} F^{k+\frac{1}{2}}, \\
(I - \frac{\Delta t}{2} D_y^2) U^{k+1} &= (I + \frac{\Delta t}{2} D_x^2) U^{k+\frac{1}{2}} + \frac{\Delta t}{2} F^{k+\frac{1}{2}}.
\end{align*}
\]

leading to a simple analytically convenient result as follows. From the first equation,

\[
U^{k+\frac{1}{2}} = \left(I - \frac{\Delta t}{2} D_x^2\right)^{-1} \left(I + \frac{\Delta t}{2} D_y^2\right) U^k + \left(I - \frac{\Delta t}{2} D_x^2\right)^{-1} \frac{\Delta t}{2} F^{k+\frac{1}{2}},
\]

and substituting into the second equation yields

\[
\begin{align*}
(I - \frac{\Delta t}{2} D_y^2) U^{k+1} &= \left(I + \frac{\Delta t}{2} D_x^2\right) \left(I - \frac{\Delta t}{2} D_x^2\right)^{-1} \left(I + \frac{\Delta t}{2} D_y^2\right) U^k \\
&\quad + \left(I + \frac{\Delta t}{2} D_x^2\right) \left(I - \frac{\Delta t}{2} D_x^2\right)^{-1} \frac{\Delta t}{2} F^{k+\frac{1}{2}} + \frac{\Delta t}{2} F^{k+\frac{1}{2}}.
\end{align*}
\]

We can go further to get

\[
\begin{align*}
(I - \frac{\Delta t}{2} D_x^2) \left(I - \frac{\Delta t}{2} D_y^2\right) U^{k+1} &= \left(I + \frac{\Delta t}{2} D_x^2\right) \left(I + \frac{\Delta t}{2} D_y^2\right) U^k \\
&\quad + \left(I + \frac{\Delta t}{2} D_x^2\right) \frac{\Delta t}{2} F^{k+\frac{1}{2}} + \frac{\Delta t}{2} F^{k+\frac{1}{2}}.
\end{align*}
\]

This is the equivalent one step time marching form of the ADI method, which will be use to show the stability of the ADI method later. Note that in this derivation we have used

\[
\begin{align*}
(I + \frac{\Delta t}{2} D_x^2) \left(I + \frac{\Delta t}{2} D_y^2\right) &= \left(I + \frac{\Delta t}{2} D_x^2\right) \left(I + \frac{\Delta t}{2} D_y^2\right)
\end{align*}
\]

and other commutative operations.

### 4.4.1 Implementation of the ADI algorithm

The key idea of the ADI method is to use the implicit discretization dimension by dimension, by taking advantage of fast tridiagonal solvers for

\[
U_{ij}^{k+\frac{1}{2}} = U_{ij}^k + \frac{\Delta t}{2} k_{xx} U_{ij}^{k+\frac{1}{2}} + \frac{\Delta t}{2} k_{yy} U_{ij}^k + \frac{\Delta t}{2} f_{ij}^{k+\frac{1}{2}}.
\]

For fixed \( j \), we get a tridiagonal system of equations for \( U_{1j}^{k+\frac{1}{2}}, U_{2j}^{k+\frac{1}{2}}, \ldots, U_{m-1,j}^{k+\frac{1}{2}} \), assuming a Dirichlet boundary condition at \( x = a \) and \( x = b \). The system of equations in matrix-
vector form is
\[
\begin{bmatrix}
1 + 2\mu & -\mu & \\
-\mu & 1 + 2\mu & -\mu \\
\vdots & \ddots & \ddots \\
-\mu & 1 + 2\mu & -\mu \\
-\mu & 1 + 2\mu \\
\end{bmatrix}
\begin{bmatrix}
U_{1j}^{k+\frac{1}{2}} \\
U_{2j}^{k+\frac{1}{2}} \\
U_{3j}^{k+\frac{1}{2}} \\
\vdots \\
U_{m-2,j}^{k+\frac{1}{2}} \\
U_{m-1,j}^{k+\frac{1}{2}} \\
\end{bmatrix}
= \hat{F},
\]

where
\[
\begin{align*}
U_{1,j}^k + \frac{\Delta t}{2} f_{1,j}^{k+\frac{1}{2}} + \mu u_{bc}(a,y_j)^{k+\frac{1}{2}} + \mu \left( U_{i,j-1}^k - 2U_{1,j}^k + U_{i,j+1}^k \right) \\
U_{2,j}^k + \frac{\Delta t}{2} f_{2,j}^{k+\frac{1}{2}} + \mu \left( U_{2,j-1}^k - 2U_{2,j}^k + U_{2,j+1}^k \right) \\
U_{3,j}^k + \frac{\Delta t}{2} f_{3,j}^{k+\frac{1}{2}} + \mu \left( U_{3,j-1}^k - 2U_{3,j}^k + U_{3,j+1}^k \right) \\
\vdots \\
U_{m-2,j}^k + \frac{\Delta t}{2} f_{m-2,j}^{k+\frac{1}{2}} + \mu \left( U_{m-2,j-1}^k - 2U_{m-2,j}^k + U_{m-2,j+1}^k \right) \\
U_{m-1,j}^k + \frac{\Delta t}{2} f_{m-1,j}^{k+\frac{1}{2}} + \mu \left( U_{m-1,j-1}^k - 2U_{m-1,j}^k + U_{m-1,j+1}^k \right) + \mu u_{bc}(b,y_j)^{k+\frac{1}{2}}
\end{align*}
\]

and \( \mu = \frac{\beta \Delta t}{2h^2} \), and \( f_i^{k+\frac{1}{2}} = f(x_i, t^{k+\frac{1}{2}}) \). For each \( j \), we need to solve a symmetric tridiagonal system of equations. The cost for the \( x \)-sweep is about \( O(5mn) \).

### 4.4.2 A pseudo-code of the ADI method in Matlab

```matlab
for j = 2:n, % Loop for fixed j
    A = sparse(m-1,m-1); b=zeros(m-1,1);
    for i=2:m,
        b(i-1) = (u1(i,j-1) -2*u1(i,j) + u1(i,j+1))/h1 + ... \\
        f(t2,x(i),y(j)) + 2*u1(i,j)/dt;
        if i == 2
            b(i-1) = b(i-1) + uexact(t2,x(i-1),y(j))/h1;
            A(i-1,i) = -1/h1;
        else
            if i==m
                b(i-1) = b(i-1) + uexact(t2,x(i+1),y(j))/h1;
                A(i-1,i-2) = -1/h1;
            else
```

...
FD methods for linear parabolic PDE

\[ A(i-1,i) = -1/h1; \]
\[ A(i-1,i-2) = -1/h1; \]
end
end
\[ A(i-1,i-1) = 2/dt + 2/h1; \]
end
\[ ut = A\backslash b; \]
% Solve the diagonal matrix.

%%%%--loop in the y direction------------------
for i = 2:m,
A = sparse(m-1,m-1); b=zeros(m-1,1);
for j=2:n,
b(j-1) = (u2(i-1,j) -2*u2(i,j) + u2(i+1,j))/h1 + ... 
f(t2,x(i),y(j)) + 2*u2(i,j)/dt;
if j == 2
b(j-1) = b(j-1) + uexact(t1,x(i),y(j-1))/h1;
A(j-1,j) = -1/h1;
else
if j==n
b(j-1) = b(j-1) + uexact(t1,x(i),y(j+1))/h1;
A(j-1,j-2) = -1/h1;
else
A(j-1,j) = -1/h1;
A(j-1,j-2) = -1/h1;
end
end
\[ A(j-1,j-1) = 2/dt + 2/h1; \] % Solve the system
end
\[ ut = A\backslash b; \]

4.4.3 Consistency of the ADI method

Adding the two equations in (4.1) together, we get

\[
\frac{U_{ij}^{k+1} - U_{ij}^{k}}{(\Delta t)/2} = 2\delta^{2}_{xx} U_{ij}^{k+\frac{1}{2}} + \delta^{2}_{yy} \left( U_{ij}^{k+1} + U_{ij}^{k} \right) + 2f_{ij}^{k+\frac{1}{2}};
\]

(4.4)

and if we subtract the first equation from the second equation, we get

\[
4U_{ij}^{k+\frac{1}{2}} = 2 \left( U_{ij}^{k+1} + U_{ij}^{k} \right) - \Delta t \delta^{2}_{yy} \left( U_{ij}^{k+1} - U_{ij}^{k} \right).
\]

(4.5)

Substituting this into (4.4) we get

\[
\left( 1 + \frac{(\Delta t)^2}{4} \delta^{2}_{xx} \delta^{2}_{yy} \right) \frac{U_{ij}^{k+1} - U_{ij}^{k}}{\Delta t} = (\delta^{2}_{xx} + \delta^{2}_{yy}) \frac{U_{ij}^{k+1} - U_{ij}^{k}}{2} + f_{ij}^{k+\frac{1}{2}},
\]

(4.6)
and we can clearly see that the discretization is second order accurate in both space and time, i.e., \( T_{ij}^k = O((\Delta t)^2 + h^2) \).

### 4.5 An implicit-explicit method for diffusion and advection equations

Consider the equation

\[
 u_t + \mathbf{w} \cdot \nabla u = \nabla \cdot (\beta \nabla u) + f(x, y, t) .
\]

In this case, it is not so easy to get a second order implicit scheme such that the coefficient matrix is diagonally dominant or symmetric positive or negative definite, due to the advection term \( \mathbf{w} \cdot \nabla u \). One approach is to use an implicit scheme for the diffusion term and an explicit scheme for the advection term, of the following form from time level \( t^k \) to \( t^{k+1} \):

\[
 \frac{u^{k+1} - u^k}{\Delta t} + (\mathbf{w} \cdot \nabla_h u)^{k+\frac{1}{2}} = \frac{1}{2} \left( (\nabla_h \cdot \beta \nabla_h u)^k + (\nabla_h \cdot \beta \nabla_h u)^{k+1} \right) + f^{k+\frac{1}{2}} , \tag{4.1}
\]

where

\[
 (\mathbf{w} \cdot \nabla_h u)^{k+\frac{1}{2}} = \frac{3}{2} (\mathbf{w} \cdot \nabla_h u)^k - \frac{1}{2} (\mathbf{w} \cdot \nabla_h u)^{k-1} . \tag{4.2}
\]

We treat the advection term implicitly, since the term only contains the first order partial derivatives and the CFL constraint is not an issue, unless \( \|\mathbf{w}\| \) is very large. The time step constraint is

\[
 \Delta t \leq \frac{h}{2\|\mathbf{w}\|_2} . \tag{4.3}
\]

At each time step, we need to solve a generalized Helmholtz equation

\[
 (\nabla \cdot \beta \nabla u)^{k+1} - \frac{2u^{k+1}}{\Delta t} = -\frac{2u^k}{\Delta t} + 2(\mathbf{w} \cdot \nabla u)^{k+\frac{1}{2}} - (\nabla \cdot \beta \nabla u)^k - 2f^{k+\frac{1}{2}} . \tag{4.4}
\]

We need \( u^1 \) to get the scheme above started. We can use the explicit Euler method (FW-CT) to approximate \( u^1 \), as this should not affect the stability and global error \( O((\Delta t)^2 + h^2) \).

### 4.6 Solving elliptic PDE using numerical methods for parabolic PDE

We recall the steady state solution of a parabolic PDE is the solution of the corresponding elliptic PDE, \( e.g., \) the steady state solution of the parabolic PDE

\[
 u_t = \nabla \cdot (\beta \nabla u) + \mathbf{w} \cdot u + f(x, t)
\]
FD methods for linear parabolic PDE

is the solution to the elliptic PDE

\[ \nabla \cdot (\beta \nabla u) + w \cdot u + \bar{f}(x) = 0, \]

if the limit

\[ \bar{f}(x) = \lim_{t \to \infty} f(x, t) \]

exists. The initial condition is irrelevant to the steady state solution, but the boundary condition is relevant. This approach has some advantages, especially for nonlinear problems where the solution is not unique. We can control the variation of the intermediate solutions, and the linear system of equations is more diagonally dominant. Since we only require the steady state solution, we prefer to use implicit methods with large time steps since the accuracy in time is unimportant.

4.7 Exercises

1. Show that a scheme for

\[ u_t = \beta u_{xx} \] (4.1)

of the form

\[ U_i^{k+1} = \alpha U_i^k + \frac{1 - \alpha}{2} \left( U_{i+1}^k + U_{i-1}^k \right) \]

where \( \alpha = 1 - 2\beta \mu, \mu = \Delta t/h^2 \) is consistent with the heat equation (4.1). Find the order of the discretization.

2. Show that the implicit scheme

\[ \left( 1 - \frac{\Delta t \beta}{2} \delta_{xx}^2 \right) \left( \frac{U_i^{k+1} - U_i^k}{\Delta t} \right) = \beta \left( 1 - \frac{h^2}{12} \delta_{xx}^2 \right) \delta_{xx}^2 U_i^k \] (4.2)

for the heat equation (4.1) has order of accuracy \((\Delta t)^2, h^4\), where

\[ \delta_{xx}^2 U_i = \frac{U_{i-1} - 2U_i + U_{i+1}}{h^2}, \]

and \( \Delta t = O(h^2) \). Compare this method with FW-CT, BW-CT, and Crank-Nicolson schemes and explain the advantages and limitations. (Note: The stability condition of the scheme is \( \beta \mu \leq \frac{3}{2} \)).

3. For the implicit Euler method applied to the heat equation \( u_t = u_{xx} \), is it possible to choose \( \Delta t \) such that the discretization is \( O((\Delta t)^2 + h^4) \)?
4. Consider the diffusion and advection equation
\[ u_t + u_x = \beta u_{xx}, \quad \beta > 0. \tag{4.3} \]

Use the von Neumann analysis to derive the time step restriction for the scheme
\[ \frac{U_j^{k+1} - U_j^k}{\Delta t} + \frac{U_{j+1}^k - U_{j-1}^k}{2h} = \beta \frac{U_{j-1}^k - 2U_j^k + U_{j+1}^k}{h^2}. \]

5. Implement and compare the Crank-Nicolson and the MOL methods using Matlab for the heat equation:
\[ u_t = \beta u_{xx} + f(x,t), \quad a < x < b, \quad t \geq 0, \]
\[ u(x,0) = u_0(x); \quad u(a,t) = g_1(t); \quad u(b,t) = g_2(t), \]
where \( \beta \) is a constant. Use \( u(x,t) = (\cos t)x^2 \sin(\pi x) \), \( 0 < x < 1 \), \( t_{final} = 1.0 \) to test and debug your code. Write a short report about these two methods. Your discussion should include the grid refinement analysis, error and solution plots for \( m = 80 \), comparison of cpu time, and any conclusions you can draw from your results. You can use Matlab code \textit{ode15s} or \textit{ode23s} to solve the semi-discrete ODE system.

Assume that \( u \) is the temperature of a thin rod with one end \( (x = b) \) just heated. The other end of the rod has a room temperature \( (70^\circ C) \). Solve the problem and find the history of the solution. Roughly how long does it take for the temperature of the rod to reach the steady state? What is the exact solution of the steady state? **Hint:** Take the initial condition as \( u(x,0) = T_0 e^{-(x-b)^2/\gamma} \), where \( T_0 \) and \( \gamma \) are two constants, \( f(x,t) = 0 \), and the Neumann boundary condition \( u_x(b,t) = 0 \).

6. Carry out the von Neumann analysis to determine the stability of the \( \theta \) method
\[ \frac{U_j^{(n+1)} - U_j^n}{k} = b \left( \theta \delta_x^2 U_j^{(n)} + (1 - \theta) \delta_x^2 U_j^{(n+1)} \right) \tag{4.4} \]
for the heat equation \( u_t = \beta u_{xx} \), where
\[ \delta_x^2 U_j = \frac{U_{j-1} - 2U_j + U_{j+1}}{h^2} \quad \text{and} \quad 0 \leq \theta \leq 1. \]

7. Implement and compare the ADI and Crank-Nicolson methods with the SOR(\( \omega \)) (try to test optimal \( \omega \)) for the following problem involving the 2D heat equation:
\[ u_t = u_{xx} + u_{yy} + f(t,x,y), \quad a < x < b, \quad c \leq y \leq d, \quad t \geq 0, \]
\[ u(0,x,y) = u_0(x,y), \]
and Dirichlet boundary conditions. Choose two examples with known exact solutions to test and debug your code. Write a short report about the two methods. Your discussion should include the grid refinement analysis (with a fixed final time, say $T = 0.5$), error and solution plots, comparison of cpu time and flops, and any conclusions you can draw from your results.