Chapter 3

Discretization of the heat equation

3.1 On the derivation of the heat equation

We will use a standard example to describe the numerical schemes for parabolic differential equations throughout the course. We are talking about the linear heat equation in one space dimension, which for example can be used to model the flow of heat on a straight homogeneous rod over time. The rod is insulated everywhere except at the two ends.

The rod in the picture has length $L = 1$, and let the coordinate $x$ describes a point along the rod. At time $t \geq 0$ the rod has temperature $u(x, t)$ at the point $x$. We can derive the differential equation by using Fourier’s law. The flux of heat $\phi$ through a cross section of the rod at $x$ is proportional to the temperature gradient, such that

$$\phi = -\lambda u_x, \quad \lambda > 0,$$

and the following conservation law holds true

$$\rho c u_t + \phi_x = 0, \quad \rho$$ is the rod’s density.

These two equations imply together that

$$u_t = a u_{xx}, \quad a = \frac{\lambda}{\rho c}.$$

By introducing scales for time, space and temperature

$$w = \frac{u}{u_0}, \quad y = \frac{x}{L}, \quad \tau = \frac{at}{L^2},$$

where $w$, $y$ and $\tau$ are dimensionless variables, $u_0$ is the characteristic temperature, and $L$ is the rod’s length, we get

$$w_{\tau} = w_{yy}, \quad 0 < y < 1.$$

We have seen that after scaling it is always possible to assume that the space interval is $[0, 1]$ and the coefficient $a$ can be set equal to 1. From now on we will usually look at the problem

$$u_t = u_{xx}, \quad 0 < x < 1, \quad t > 0.$$
Together with the differential equation we need to provide appropriate boundary conditions and initial conditions. The kind of boundary and initial conditions necessary and sufficient to get a unique solution vary from differential equation to differential equation. We will consider few options for the heat equations.

**Pure initial value problem.** In this case we assume the rod is infinitely long.

\[
\begin{align*}
    u_t &= u_{xx}, & x &\in \mathbb{R}, & t &> 0, \\
    u(x,0) &= f(x), & x &\in \mathbb{R}.
\end{align*}
\]

**Initial/Boundary value problem (I/BVP).** This case includes the situation of heat transport in a homogeneous rod of length 1. We must consider an initial function and boundary conditions at the two ends of the rod.

\[
\begin{align*}
    u_t &= u_{xx}, & 0 &< x < 1, & t &> 0, \\
    u(x,0) &= f(x), & 0 &\leq x \leq 1, \\
    u(0,t) &= g_0(t), & t &> 0, \\
    u(1,t) &= g_1(t), & t &> 0.
\end{align*}
\]

(3.1)

3.2 Numerical solution of the initial/boundary value problem

3.2.1 Numerical approximation on a grid.

We adopt a *step-size* \( h \) in the \( x \)-direction, and one in the \( t \)-direction which we denote \( k \). We assume at first that \( h = 1/(M+1) \) for a given integer \( M \).

We then define *gridpoints* or *nodes* \((x_m, t_n)\) by

\[
x_m = mh, \quad 0 \leq m \leq M + 1, \quad t_n = nk, \quad n = 0, 1, 2, \ldots.
\]

Observe that this means that \( x_0 = 0 \) and \( x_{M+1} = 1 \) are the boundary points. The solution in the point \((x_m, t_n)\) is denoted \( u^n_m := u(x_m, t_n) \). And from now on we denote with \( U^n_m \) the approximation to the solution in \((x_m, t_n)\) produced by the numerical method.
3.2. NUMERICAL SOLUTION OF THE INITIAL/BOUNDARY VALUE PROBLEM

3.2.2 Euler, Backward Euler and Crank–Nicolson

We present three different difference schemes for the heat equation.

The Euler’s method. We adopt a simpler notation for the derivatives and set

\[
\begin{align*}
\partial_x u &= u_x = \frac{\partial u}{\partial x}, \\
\partial_x^k u &= \frac{\partial^k u}{\partial x^k}, \\
\partial_t u &= u_t = \frac{\partial u}{\partial t}.
\end{align*}
\]

We expand \( u_m^{n+1} = u(x_m, t_n + k) \) for constant \( x = x_m \), around \( t = t_n \), and get

\[
\begin{align*}
u_m^{n+1} &= u_m^n + k \partial_t u_m^n + \varphi_m^n, \\
\varphi_m^n &= \frac{1}{2} k^2 \partial_t^2 u_m^n + \cdots.
\end{align*}
\]

But we can now use the heat equation ensuring \( \partial_t u_m^n = \partial_x^2 u_m^n \), we then approximate this second derivative with central differences as in (2.6)

\[
u_m^{n+1} = u_m^n + k \frac{h^2}{\delta_x^2} u_m^n - \psi_m^n + \varphi_m^n
\]

where the index on \( \delta \) means that we apply this operator in the \( x \)-direction i.e.

\[
\delta_x^2 u_m^n = u_{m+1}^n - 2u_m^n + u_{m-1}^n.
\]

From the expression in (2.6) we find that

\[
\psi_m^n = \frac{1}{12} k h^2 \delta_x^4 u_m^n + \cdots.
\]

Summarizing we have

\[
u_m^{n+1} = u_m^n + k \frac{h^2}{\delta_x^2} u_m^n + \tau_m^n = u_m^n + \frac{k}{h^2} (u_{m+1}^n - 2u_m^n + u_{m-1}^n) + \tau_m^n
\]

where

\[
\tau_m^n = \varphi_m^n - \psi_m^n = \left( \frac{1}{2} k^2 \partial_t^2 - \frac{1}{12} k h^2 \partial_x^4 \right) u_m^n + \cdots
\]

the Euler’s formula is obtained by replacing all the exact \( u \)-values (small letters) with approximate values \( U \) (capital letters) in the above formula and discard the truncation error \( \tau_m^n \).
Euler’s method

\begin{equation}
U_{m}^{n+1} = U_{m}^{n} + r \delta_{x}^{2} U_{m}^{n}
\end{equation}

\[ r = \frac{k}{h^{2}} \]

The picture above to the right is called computational molecule, it is a sort of local chart over the grid, indicating which grid-points are used in the formula. The idea is now to start at \( n = 0 \), corresponding to \( t_{0} = 0 \) where \( u(x, t_{0}) = u(x, 0) = f(x) \) which is known. It is then possible to order the values \( U_{m}^{0} = f(x_{m}), m = 0, \ldots, M + 1 \). Then we set \( n = 1 \) and use first the boundary values to get \( U_{0}^{1} = g_{0}(k) \) and \( U_{M+1}^{0} = g_{1}(k) \). For the remaining values we use the formula (3.2) above. It is possible to see that the grid-point at level \( t_{n+1} = t_{1} \) in the computational molecule can be computed using known values.

Algorithm (Euler’s method for the heat equation)

\begin{verbatim}
U_{m}^{0} := f(x_{m}), \ m = 0, \ldots, M + 1
for n = 0, 1, 2, \ldots
    U_{0}^{n+1} := g_{0}(t_{n+1})
    U_{M+1}^{n+1} := g_{1}(t_{n+1})
    U_{m}^{n+1} := U_{m}^{n} + r \delta_{x}^{2} U_{m}^{n}, \ m = 1, \ldots, M
end
\end{verbatim}

Example.

\[ u_{t} = u_{xx}, \ 0 < x < 1, \ t > 0, \]
\[ f(x) = \begin{cases} 2x, & 0 \leq x \leq \frac{1}{2}, \\ 2(1-x), & \frac{1}{2} < x \leq 1, \end{cases} \]
\[ g_{0}(t) = g_{1}(t) = 0, \ t > 0. \]

In the picture above we display the initial function. By running a simulation in Matlab based on this example, where we let \( h = 0.1 \ (M = 9) \), and \( k = 0.0045 \). The reason why we take \( k \) so small compared to \( h \) will be explained later on. Figure 3.1 shows the numerical solution in the grid-points as small rings, at time \( t = 0.5, 10 \) and 20.
3.2. NUMERICAL SOLUTION OF THE INITIAL/BOUNDARY VALUE PROBLEM

Backward Euler. We expand now instead $u^n_m$ omkring $x = x_m$, $t = t_{n+1}$, and obtain

\[
\begin{align*}
    u^n_m &= u(x_m, t_{n+1} - k) \\
    &= u^n_{m+1} - k \partial_t u^n_{m+1} + \frac{1}{2} k^2 \partial^2_t u^n_{m+1} + \cdots \\
    &= u^n_{m+1} - k \partial_x^2 u^n_{m+1} + \frac{1}{2} k^2 \partial^2_x u^n_{m+1} + \cdots \\
    &= u^n_{m+1} - k \left( \frac{1}{12} k h^2 \partial_x^4 u^n_{m+1} + \frac{1}{2} k^2 \partial^2_t u^n_{m+1} + \cdots \right) + \frac{1}{2} k^2 \partial^2_x u^n_{m+1} + \cdots \\
    &= u^n_{m+1} - r \delta_x^2 u^n_{m+1} + \tau^n_m
\end{align*}
\]

where

\[
\tau^n_m = \left( \frac{1}{12} k h^2 \partial_x^4 + \frac{1}{2} k^2 \partial^2_t \right) u^n_{m+1} + \cdots.
\]

By replacing the $u$’s med $U$’s and discard the truncation error $\tau^n_m$ we obtain

\[
U^{n+1}_m - r \delta_x^2 U^{n+1}_m = U^n_m,
\]

where

\[
r = \frac{k}{h^2}.
\]

The Backward Euler method is an implicit method. This means that at each time step we must solve a system of linear equations to compute $U^{n+1}_m$, $m = 1, \ldots, M$. We
will discuss later on the solution of linear systems. We now will present another implicit method.

**Crank–Nicolson’s method.** This method is based on the trapezoidal rule, we can represent the truncation error in the trapezoidal rule by

\[
\int_0^k f(t) \, dt = \frac{1}{2} k (f(0) + f(k)) - \frac{1}{12} k^3 f''(\frac{k}{2}) + \cdots.
\]

To derive the method we use the obvious formula

\[
u(x_m, t_{n+1}) - u(x_m, t_n) = \int_{t_n}^{t_{n+1}} u_t(x_m, t) \, dt,
\]

and approximate the integral with the trapezoidal rule where we use the notation \(u_{n+1/2} = u(x_m, t_n + \frac{1}{2}k)\).

\[
u_{m}^{n+1} = u_{m}^{n} + \frac{1}{2} k (\partial_t u_{m}^{n} + \partial_t u_{m}^{n+1}) - \frac{1}{12} k^3 \partial_t^3 u_{m}^{n+1/2} + \cdots
\]

\[
= u_{m}^{n} + \frac{1}{2} k (\partial_x^2 u_{m}^{n} + \partial_x^2 u_{m}^{n+1}) - \frac{1}{12} k^3 \partial_t^3 u_{m}^{n+1/2} + \cdots
\]

\[
= u_{m}^{n} + \frac{1}{2} k \left( \frac{1}{h^2} \delta_x^2 u_{m}^{n} + \frac{1}{h^2} \delta_x^2 u_{m}^{n+1} \right) - \frac{1}{2} k \left( \frac{1}{12} h^2 \partial_x^4 u_{m}^{n} + \frac{1}{12} h^2 \partial_x^4 u_{m}^{n+1} + \cdots \right)
\]

\[
- \frac{1}{12} k^3 \partial_t^3 u_{m}^{n+1/2} + \cdots.
\]

We simplify and summarize

\[
u_{m}^{n+1} = u_{m}^{n} + \frac{r}{2} (\delta_x^2 u_{m}^{n} + \delta_x^2 u_{m}^{n+1}) + \tau_{m}^{n},
\]

\[
\tau_{m}^{n} = - \frac{1}{12} k^3 \partial_t^3 u_{m}^{n+1/2} - \frac{1}{12} k h^2 \partial_x^4 u_{m}^{n+1/2} + \cdots,
\]

where we have used that

\[
\frac{1}{2} (\delta_x^4 u_{m}^{n} + \delta_x^4 u_{m}^{n+1}) = \partial_x^4 u_{m}^{n+1/2} + O(k^2).
\]

\[
\begin{align*}
\text{Crank–Nicolson method} & \quad \begin{array}{c}
\text{m} - 1, n + 1 \\
m, n + 1 \\
m + 1, n + 1
\end{array} \\
\begin{array}{c}
m - 1, n \\
m, n \\
m + 1, n
\end{array}
\end{align*}
\]

\[
(1 - \frac{r}{2} \delta_x^2) U_{m}^{n+1} = (1 + \frac{r}{2} \delta_x^2) U_{m}^{n},
\]

\[
r = \frac{k}{h^2}.
\]

\(\text{This particular form of truncation error has not been presented in the course TMA4215, but we will consider it known anyway.}\)
3.2. NUMERICAL SOLUTION OF THE INITIAL/BOUNDARY VALUE PROBLEM

We summarize by writing all the formulae in a compact form

\[(E)\quad U_m^{n+1} = (1 + r \delta^2_x) U_m^n \quad \text{or} \quad \frac{1}{k} \Delta t U_m^n = \frac{1}{h^2} \delta^2_x U_m^n,\]

\[(BE)\quad (1 - r \delta^2_x) U_m^{n+1} = U_m^n \quad \text{or} \quad \frac{1}{k} \nabla t U_m^{n+1} = \frac{1}{h^2} \delta^2_x U_m^{n+1},\]

\[(CN)\quad (1 - \frac{r}{2} \delta^2_x) U_m^{n+1} = (1 + \frac{r}{2} \delta^2_x) U_m^n \quad \text{or} \quad \frac{1}{k} \delta_t U_m^{n+1/2} = \frac{1}{h^2} \delta^2_x \mu_t U_m^{n+1/2}.\]

Note that \((E)\) is explicit while both \((BE)\) and \((CN)\) are implicit.

Remark on the truncation error. In the methods presented we have defined the local truncation error \(\tau_m^n\) in the point \((x_m, t_n)\). Given a formula in general, we find the corresponding local truncation error by inserting the exact solution in the grid points in the formula and moving all the terms on one side. The truncation error is expressed in powers of the step-sizes \(h\) and \(k\) and the derivatives of the exact solution by using the Taylor expansion.

3.2.3 Solution of the linear systems in Backward Euler’s method and Crank–Nicolson

Our starting point is that we know \(U^n_m, 0 \leq m \leq M + 1\), together with \(U^0_0\) and \(U^{M+1}_M\) given by the boundary conditions. We need to compute \(U^{n+1}_m, 1 \leq m \leq M\). Let us consider Crank–Nicolson first. The right hand side (r.h.s.) of the equations is known, we set \(d_m^{n+1} = (1 + r \delta^2_x) U_m^n = \frac{r}{2} U_{m-1}^n + (1 - r) U_m^n + \frac{r}{2} U_{m+1}^n, 1 \leq m \leq M\).

For the left hand side (l.h.s.) we get component-wise

\[
\left(1 - \frac{r}{2} \delta^2_x\right) U_m^{n+1} = -\frac{r}{2} U_{m-1}^{n+1} + (1 + r) U_m^{n+1} - \frac{r}{2} U_{m+1}^{n+1}, 1 \leq m \leq M,
\]

where we substitute \(U^0_0 = g_0^{n+1} = g_0(t_{n+1})\) and \(U^{n+1}_M = g_1^{n+1} = g_1(t_{n+1})\). We can now express the equation in matrix-vector form

\[
\begin{bmatrix}
1 + r & -\frac{r}{2} \\
-\frac{r}{2} & 1 + r & -\frac{r}{2} \\
\vdots & \ddots & \ddots \\
-\frac{r}{2} & 1 + r & -\frac{r}{2} \\
-\frac{r}{2} & 1 & 1 + r
\end{bmatrix}
\begin{bmatrix}
U_1^{n+1} \\
U_2^{n+1} \\
\vdots \\
U_{M-1}^{n+1} \\
U_M^{n+1}
\end{bmatrix}
= \begin{bmatrix}
d_1^{n+1} + \frac{r}{2} g_0^{n+1} \\
d_2^{n+1} \\
\vdots \\
d_{M-1}^{n+1} \\
d_M^{n+1} + \frac{r}{2} g_1^{n+1}
\end{bmatrix}.
\]

In a similar way we get the following linear system for Backward-Euler

\[
\begin{bmatrix}
1 + 2r & -r \\
-r & 1 + 2r & -r \\
\vdots & \ddots & \ddots \\
-r & 1 + 2r & -r \\
-r & 1 + 2r
\end{bmatrix}
\begin{bmatrix}
U_1^{n+1} \\
U_2^{n+1} \\
\vdots \ \\
U_{M-1}^{n+1} \\
U_M^{n+1}
\end{bmatrix}
= \begin{bmatrix}
U_1^n + r g_0^{n+1} \\
U_2^n \\
\vdots \\
U_{M-1}^n \\
U_M^n + r g_1^{n+1}
\end{bmatrix}.
\]
The two matrices in Crank-Nicolson and Backward Euler are examples of tridiagonal matrices. These special tridiagonal matrices are also called Toeplitz matrices, i.e. the elements along each of the three diagonals are equal. In Toeplitz matrices we need only to specify the first row and the first column to determine the whole matrix. If in addition we know that the matrix is symmetric, it is enough to specify the first row (or column).

In general when we solve differential equations with difference methods we obtain sparse matrices. PDEs in one space dimension and at most second order derivatives give rise typically to (at least) tridiagonal matrices. Higher order derivatives imply a larger bandwidth in the matrix. There is a clear correspondence between the bandwidth of the matrix and the number of neighboring values $U_{m-p}, \ldots, U_m, U_{m+1}, \ldots, U_{m+q}$ used to approximate the highest order derivative of $u$ with respect to $x$, at the grid point $x_m$ (see also the method of undetermined coefficients form chapter 1). In the case of several space dimensions we obtain a block structure in the matrices, for example the heat equation in two space dimensions will typically give a block-tridiagonal matrix. Toeplitz structure is lost when considering a heat equation where the rod's material is inhomogeneous (see the chapter on then the differential equation becomes

$$u_t = a(x)u_{xx}$$

where $a(x)$ is a given function.

There are special algorithms which can be used to solve linear systems with tridiagonal matrices. Among the direct methods a variant of Gaussian elimination called Thomas algorithm. We are not going to discuss this algorithm in detail here.

Instead we consider some simple examples on how it is possible to use Matlab to assemble and solve the above equations.

### 3.2.4 Solution of linear systems in Matlab

When working with sparse matrices, that is with matrices the majority of whose elements are zero, it is important to store the matrix in a cheap way in the computer. For a $M \times M$-matrix as the one considered in the previous section it is unpractical to store all the elements, it is possible instead to store a list of all the indexes corresponding to nonzero elements and the corresponding value. If you choose $M = 1000$ in the last example, the matrix will have $10^6 = $ one million elements in total, while there are about 3000 elements which are different from zero. Another issue is that if we multiply a large and sparse matrix with a vector, we will make a lot of multiplications by and additions with zero which could be avoided.

Matlab has built in facilities for this. Start Matlab and try typing > help sparse or > help spdiags. The function sparse converts a full matrix into a sparse matrix, meaning that only the non-zero elements are stored. The conversion of a matrix from sparse format to full format can be achieved using the function full. The function spdiags is used to generate matrices in sparse format using their diagonals. To create the matrix for the Crank–Nicolson method in sparse format, we proceed as follows. We assume $M = 10$ so that $h = 1/11$ and choose $k$ such that $r = k/h^2 = 1$. Try the following sequence of commands

```matlab
> M=10;
> r=1;
> e=ones(M,1);
> A=spdiags([-r/2*e, (1+r)*e, -r/2*e], -1:1:M,M);
```

Try to remove ';' in the last command to see how Matlab shows a matrix in sparse format. Matlab assigns indexes to the diagonals in a matrix by giving the index 0 to the main
diagonal, the sub-diagonal gets index \(-1\), the super-diagonal gets index \(1\) and so on. The second input argument of `spdiags` is a vector \(d\) of integer numbers such that the column \(j\) from the matrix given as the first input argument becomes the diagonal \(d(j)\) in the result. The two last input arguments in the call to the function specify that the result must be a \(M \times M\) matrix.

Let us see how a time-step with Crank–Nicolson can be implemented in Matlab. Assume that the variable \(U_0\) has \(M + 1\) elements and stores the numerical approximation at time \(t = t_n\). If for example \(n = 0\), \(U_0\) is generated from the given starting values. It can be a good idea to let \(U_0\) have dimension \(M + 2\) and store the boundary points respectively in \(U_0(1)\) and \(U_0(M+2)\).

Let us now set \(n = 0\), and use the hat function as a starting value, i.e. \(f(x) = 1 - |2x - 1|\). We define \(U_0\) by

\[
> h=1/(M+1); \quad \text{definer romskrittlenge } h
> X=(0:h:1)'; \quad \text{Gitterpunktene i x-retning er en kolonnevektor}
> U0 = 1-abs(2*X-1); \quad \text{Definer hattfunksjonen som startverdi}
\]

Assume the above commands are executed, such that \(r, A, U_0, X\) are all defined.

We assume also that the boundary values are \(g_0(t) = g_1(t) = 0\). Then we can make a step with Crank–Nicolson as follows

\[
\begin{align*}
& d=r/2*U0(1:end-2)+(1-r)*U0(2:end-1)+r/2*U0(3:end); \\
& U1=[0;A;d;0]; \quad \text{Numerisk losning ved tidsskritt } n+1 \\
& \text{plot}(X,U1,'o') \quad \text{Plott resultatet}
\end{align*}
\]

Observe that the definition of \(A\) is done once and for all, and it is used in all the following time-steps. It would have been even better to LU-factorize \(A\) at the beginning, so that in the subsequent steps one uses just the backward substitution algorithm.

Try to plot or make a movie of the results in time, look at \(>\text{help movie}\), for example. You can also try with a bigger value of \(M\) to improve accuracy and decrease the numerical error.

### 3.2.5 The \(\theta\)-method

It is possible to express all the three methods defined in the previous sections in a unified format by writing

\[
(1 - \theta r \delta^2_x) U_{m}^{n+1} = (1 + (1 - \theta) r \delta^2_x) U_{m}^{n}.
\]

One gets then

- (E) \(\theta = 0\)
- (BE) \(\theta = 1\)
- (CN) \(\theta = \frac{1}{2}\)

Local truncation error for the \(\theta\)-method

\[
k \tau_{m}^{n} = (1 - \theta r \delta^2_x) u_{m}^{n+1} - (1 + (1 - \theta) r \delta^2_x) u_{m}^{n} = (1 - \theta r \delta^2_x)(u_{m}^{n+1} - u_{m}^{n}) - r \delta^2_x u_{m}^{n}.
\]
CHAPTER 3. DISCRETIZATION OF THE HEAT EQUATION

We expand all the expressions around \((x_m, t_n)\) and obtain

\[
k^{\tau_n}_{m} = \left(1 - \theta k(\partial_x^2 + \frac{1}{12}h^2\partial_x^4 + \cdots)\right) \left(k \partial_t + \frac{1}{2} k^2 \partial_t^2 + \frac{1}{6} k^3 \partial_t^3\right) u^n_m - k\left(\partial_x^2 + \frac{1}{12} h^2 \partial_x^4 + \cdots\right) u^n_m
\]

\[
= \left(k \partial_x^2 + \frac{1}{2} k^2 \partial_t^2 + \frac{1}{6} k^3 \partial_t^3 - \theta k^2 \partial_t^2 - \frac{1}{2} \theta k^3 \partial_t^3 - k \partial_x^2 - \frac{1}{12} k h^2 \partial_x^4 + \cdots\right) u^n_m + \cdots
\]

\[
= \left(\frac{1}{2} - \theta\right) k^2 \partial_t^2 u^n_m - \frac{1}{12} k h^2 \partial_x^4 u^n_m + \left(\frac{1}{6} - \frac{1}{2} \theta\right) k^3 \partial_t^3 u^n_m + \cdots.
\]

We conclude that

\[
k^{\tau_n}_{m} = O(k^2 + k h^2) \quad \text{når} \quad \theta \neq \frac{1}{2},
\]

\[
k^{\tau_n}_{m} = O(k^3 + k h^2) \quad \text{når} \quad \theta = \frac{1}{2}.
\]

And we then expect that (CN) is more accurate than (E) and (BE).

### 3.3 Semi-discretization

#### 3.3.1 Semi-discretization of the heat equation

We look again at the (I/BV) problem for the heat equation (3.1). Let us now draw vertical grid-lines as shown in the picture.

The lines are parallel to the \(t\)-axis and cross the \(x\)-axes in \(x = x_m, \ m = 0, \ldots, M + 1\). We consider the differential equation along such lines, this means

\[
\partial_t u(x_m, t) = \partial_x^2 u(x_m, t) = \frac{1}{h^2} \partial_x^2 u(x_m, t) + \varphi(x_m, t),
\]

\[
\varphi(x_m, t) = -\frac{1}{12} h^2 \partial_x^4 u(x_m, t) + \cdots.
\]
3.3. SEMI-DISCRETIZATION

We introduce the functions of one variable \( v_m(t), \ m = 0, \ldots, M + 1, \) as approximations to \( u(x_m, t). \) We require that

\[
\begin{align*}
v_0(t) &= g_0(t), \\
v_{M+1}(t) &= g_1(t), \\
v_m(t) &= \frac{1}{h^2} \delta_x^2 v_m(t), \quad v_m(0) = f(x_m), \quad m = 1, \ldots, M,
\end{align*}
\]

where \( \dot{v}_m(t) = \frac{dv_m(t)}{dt}. \) For a more compact notation, let \( \mathbf{v}(t) := [v_1(t), \ldots, v_M(t)]^T. \) We get

\[
\dot{\mathbf{v}} = \begin{bmatrix} \dot{v}_1 \\ \dot{v}_2 \\ \vdots \\ \dot{v}_{M-1} \\ \dot{v}_M \end{bmatrix} = \frac{1}{h^2} \begin{bmatrix} -2 & 1 & & & \\ 1 & -2 & 1 & & \\ & \ddots & \ddots & \ddots & \\ 1 & -2 & 1 & & \\ & & 1 & -2 & \end{bmatrix} \begin{bmatrix} v_1 \\ v_2 \\ \vdots \\ v_{M-1} \\ v_M \end{bmatrix} + \frac{1}{h^2} \begin{bmatrix} g_0(t) \\ 0 \\ \vdots \\ 0 \\ g_1(t) \end{bmatrix}.
\]

So we obtain a (linear) system of ordinary differential equations (ODEs) of the type

\[
\dot{\mathbf{v}} = A\mathbf{v} + \mathbf{b}(t), \quad \mathbf{v}(0) = \mathbf{v}_0 = [f(x_1), \ldots, f(x_M)]^T. \tag{3.5}
\]

This system is a special case of the general format for ODEs which is used in standard numerical codes for ODEs. The general format is

\[
\dot{\mathbf{v}} = \mathbf{F}(t, \mathbf{v}), \quad \mathbf{v}(0) = \mathbf{v}_0, \tag{3.6}
\]

where \( \mathbf{v} \) and \( \mathbf{F}(t, \mathbf{v}) \) are vectors in \( \mathbb{R}^M. \) Three of the simplest methods for the numerical solution of (3.6) with time-step \( k \) are

- (E) Euler : \( \mathbf{V}^{n+1} = \mathbf{V}^n + k \mathbf{F}(t_n, \mathbf{V}^n) \)
- (BE) Backward Euler : \( \mathbf{V}^{n+1} = \mathbf{V}^n + k \mathbf{F}(t_{n+1}, \mathbf{V}^{n+1}) \)
- (T) Trapezoidal rule : \( \mathbf{V}^{n+1} = \mathbf{V}^n + \frac{k}{2} \left( \mathbf{F}(t_n, \mathbf{V}^n) + \mathbf{F}(t_{n+1}, \mathbf{V}^{n+1}) \right). \)

By setting \( \mathbf{F}(t, \mathbf{v}) = A\mathbf{v} + \mathbf{b}(t) \) from (3.5), it is possible to reproduce the three methods presented earlier. In particular (T) becomes (CN).

3.3.2 Semidiscretization principle in general

This strategy can be applied also to differential equations other than the heat equation. For such equations we replace all derivatives with difference approximations, while the time, \( t, \) remains continuous. The result is

\[
\text{PDE} \quad \rightarrow \quad \text{System of ODEs}
\]

An advantage with this approach is that now it is possible to exploit off-the-shelf software for ODEs, where advanced routines for error and step-size control are already incorporated.
In particular the method can be of interest in case of nonlinear PDEs, because in that case also the semi-discretized ODE problem will be nonlinear, and standard ODE software is designed to solve numerically also such problems in an efficient way.

A problem often encountered is that the resulting ODE system is stiff. For the linear semi-discrete system (3.5) this means typically that the eigenvalues $\lambda_1, \ldots, \lambda_M$ of $A$ have negative real part and the quotient

$$\alpha = \frac{\max_i |\text{Re} \lambda_i|}{\min_i |\text{Re} \lambda_i|}.$$ 

is relatively large. If $A$ is the tridiagonal matrix reported above then

$$\lambda_s = -\frac{4}{h^2} \sin^2 \frac{s\pi}{2(M + 1)}, \quad m = 1, \ldots, M,$$

that is all the eigenvalues are real. For small values of $x$, $\sin x \approx x$, such that, for big $M$, the eigenvalue with the smallest absolute value is

$$|\lambda_1| \approx \frac{4}{h^2} \frac{\pi^2}{2^2} = \pi^2,$$

and the eigenvalue with the biggest absolute value is

$$|\lambda_M| = -\frac{4}{h^2} \sin^2 \frac{M\pi}{2(M + 1)} \approx \frac{4}{h^2} \sin^2 \frac{\pi}{2} = \frac{4}{h^2}.$$ 

Then we get $\alpha \approx \frac{4}{\pi h^2} \gg 1$ when $h$ is small.

Later on we will see that this fact implies that (BE) and (CN) perform better than (E).

### 3.3.3 General approach

Abstractly we can write a partial differential equation (evolution equation) in the form

$$\partial_t u = Lu,$$

where $L$ is a differential operator with space derivatives. For example the heat equation is obtained by considering $L = \partial_x^2$. Generally, in one space dimension, we have

$$L = L(x, t, \partial_x, \partial_x^2, \ldots).$$

The semi-discretization leads to

$$L \quad \rightarrow \quad L_h,$$

that is, $L_h$, is a discrete operator now acting on the components of a vector of functions in one variable instead of functions of two variables. For each component we write

$$\partial_t u(x_m, t) = L_h u(x_m, t) + \varphi(x_m, t),$$

where $\varphi(x_m, t)$ is the truncation error in the space direction. We let now $v_m(t) \approx u(x_m, t)$ and define

$$\dot{v}_m(t) = L_h v_m(t),$$

(including the boundary conditions).

We will next look at the truncation error due to the time-discretization, that is after having chosen an integration method for ODEs. Assume we use the trapezoidal rule for example.
3.3. SEMI-DISCRETIZATION

Let \( y(t) \) be the solution of
\[
\dot{y} = F(t, y), \quad y \in \mathbb{R}^M.
\]
It is possible to show that with step-size \( k \) this gives
\[
y_{m+1} := y_m(t_{n+1}) = y_m + \frac{k}{2} \left( F_m(t_n, y^n) + F_m(t_{n+1}, y^{n+1}) \right) + \psi_m^n,
\]
where
\[
\psi_m^n = -\frac{1}{12} k^3 y_m^{(3)}(t_n) + \cdots.
\]
But \( u(x, t) \) is such that \( \partial_t u(x, t) = L_h u(x, t) + \varphi(x, t) \). Let us insert \( y_m(t) = u(x_m, t) \) in the general ODE-formulation above, such that
\[
F_m(t, y) = L_h u(x_m, t) + \varphi(x_m, t),
\]
i.e., a system of ODEs whose solution is the solution of the PDE problem along the vertical lines \( x = x_m, t \). We get then
\[
u_{m+1} = u_m + \frac{k}{2} \left( L_h u_m + \varphi_m + L_h u_{m+1} + \varphi_{m+1} \right) + \psi_{m+1} = u_m + \frac{k}{2} \left( L_h u_m + L_h u_{m+1} \right) + k \tau_m^n,
\]
where
\[
k \tau_m^n = \frac{k}{2} (\varphi_m^n + \varphi_{m+1}^n) + \psi_m^n.
\]
Note that this is true in general for the semi-discretization principle. In particular the result is consistent with what we know for the three methods (E), (BE) and (CN) applied to the heat equation.

3.3.4 \( u_t = Lu \) with different choices of \( L \)

Case A.
\[
u_t = \underbrace{a(x) u_{xx}}_{L_a} + b(x) u_x + c(x) u.
\]
We can also write
\[
L = a \partial_x^2 + b \partial_x + c.
\]
Requirements: \( a, b, c \) are continuous in \([0, 1]\),
\( a(x) > 0 \) in \([0, 1]\).

Space-discretization

<table>
<thead>
<tr>
<th>( u_{xx} )</th>
<th>Discretization</th>
<th>Truncation error</th>
</tr>
</thead>
<tbody>
<tr>
<td>( u_{xx} \rightarrow \frac{1}{h^2} \delta_x^2 u )</td>
<td>( \mathcal{O}(h^2) )</td>
<td></td>
</tr>
<tr>
<td>( u_x \rightarrow \begin{cases} \frac{1}{h} \Delta_x u \ \frac{1}{h} \nabla_x u \ \frac{1}{2h} (u(x + h) - u(x - h)) = \frac{1}{h} \mu \delta_x u \end{cases} )</td>
<td>( \mathcal{O}(h) )</td>
<td></td>
</tr>
</tbody>
</table>


CHAPTER 3. DISCRETIZATION OF THE HEAT EQUATION

We set therefore

\[
L_h u = a \frac{1}{h^2} \delta_x^2 u + \begin{cases} 
\frac{1}{h} \Delta_x u \\
\frac{1}{h} \nabla_x u \\
\frac{1}{h} \delta_x^2 u 
\end{cases} + cu.
\]

We get \(Lu = L_h u + \varphi\) where

\[
\varphi = -\frac{1}{12} a h^2 \partial_x^4 u - b \begin{cases} 
\frac{1}{2} h \partial_x^2 u \\
\frac{1}{2} h \partial_x^2 u \\
\frac{1}{6} h^2 \partial_x^3 u
\end{cases}.
\]

The choice of \(\Delta_x\) versus \(\nabla_x\) is called upwind/downwind differencing. One of these two is chosen in the case of the so called convection dominated problems. The sign of \(b\) determines if one should use \(\Delta_x\) or \(\nabla_x\). \(b > 0 \rightarrow \Delta, b < 0 \rightarrow \nabla\).

**Case B.** Consider now the equation

\[
\frac{d}{dt} (a(x) u_x) _{L^2} L = \partial_x (a \partial_x).
\]

\(L\) is self-adjoint. In particular this means that if you use the inner product between differentiable functions which are 0 in the endpoints 0 and 1, defined by

\[
\langle u, v \rangle = \int_0^1 u(x) v(x) \, dx,
\]

so we get \(\langle Lu, v \rangle = \langle u, Lv \rangle\) for all \(u, v\). If we look at an analogous situation with the inner product on \(\mathbb{R}^n\)

\[
\langle x, y \rangle = y^T x
\]

and replace \(L\) with a matrix, then the analogous condition becomes

\[
y^T A x = (Ax, y) = \langle x, Ay \rangle = y^T A^T x,
\]

for all \(x, y \in \mathbb{R}^n\), implying that \(A = A^T\), i.e. that \(A\) is symmetric.

A possible idea is to expand \(L\) by using the product rule of differentiation,

\[
\frac{d}{dt} (a(x) u_x) = au_{xx} + a' u_x.
\]

So that we get an equation of the same type as in the case A with \(b = a'\) and \(c = 0\). Another (and usually better) possibility is to discretize directly the original form, we let

\[
\partial_x (a \partial_x) u(x_m, t) \rightarrow \frac{1}{h} \delta_x (a \frac{1}{h} \delta_x U) = \frac{1}{h^2} \delta_x (a_m(U_{m+1/2} - U_{m-1/2}))
\]

\[
= \frac{1}{h^2} \left( a_{m+1/2}(U_{m+1} - U_m) - a_{m-1/2}(U_m - U_{m-1}) \right).
\]

The truncation error is \(O(h^2)\).

A method by Tikhonov and Samarski. Write the equation in conservative form

1) \(u_t + w_x = 0\),
2) \(w = -au_x\).
3.4. BOUNDARY CONDITIONS INVOLVING DERIVATIVES

We discretize 1) by
\[ \partial_t u_m = -\partial_x w_m \approx -\frac{1}{h} \delta_x w_m = -\frac{1}{h} \left( w_{m+1/2} - w_{m-1/2} \right). \]

For the other equation we get \( u_x = -w/a \) and
\[ \int_{x_m}^{x_{m+1}} u_x \, dx = -\int_{x_m}^{x_{m+1}} \frac{w}{a} \, dx \approx -w_{m+1/2} \int_{x_m}^{x_{m+1}} \frac{dx}{a}. \]

Set now
\[ A_m = \frac{1}{h} \int_{x_m}^{x_{m+1}} \frac{dx}{a}. \]

And therefore
\[ \frac{1}{h} w_{m+1/2} \approx -A_m (u_{m+1} - u_m), \quad \frac{1}{h} w_{m-1/2} \approx -A_{m-1} (u_m - u_{m-1}), \]

such that in the discretization of 1) we get
\[ \partial_t u_m \approx A_m (u_{m+1} - u_m) - A_{m-1} (u_m - u_{m-1}), \]

and the semi-discrete system is
\[ \dot{v}_m = A_m (v_{m+1} - v_m) - A_{m-1} (v_m - v_{m-1}). \]

3.4 Boundary conditions involving derivatives

3.4.1 Different types of boundary conditions

We look at boundary conditions used for the heat equation in 3 space dimensions.

The picture illustrates the flux of heat \( \phi \) through the surface \( A \), with normal vector \( \vec{n} \) from the side I to the side II. This flux is proportional to the directional derivative of the temperature in the direction of a normal vector pointing \textit{outwards} the domain. We write
\[ \phi = -\lambda \frac{\partial u}{\partial n} = -\lambda \vec{n} \cdot \nabla u. \]

You can think that the surface \( A \) is a part of the surface (boundary) of the domain of definition of the problem in \( \mathbb{R}^3 \) where we solve the equations. We name the domain in space \( \Omega \), and the boundary \( \partial \Omega \).

Physical situations with boundary conditions including derivatives.

1. The heat flux given (specified) on \( \partial \Omega \)
\[ -\lambda \frac{\partial u}{\partial n} = \phi \quad \text{given}. \]
2. Convection
\[-\lambda \frac{\partial u}{\partial n} = \alpha (u - u_0).\]
where we do not consider the boundary layer.

3. Radiation (Plancks radiation law from statistical mechanics)
\[-\lambda \frac{\partial u}{\partial n} = \sigma (u^4 - u_0^4).\]

In what follows we will consider the model
\[\frac{\partial u}{\partial n} + \eta u = g,\]
where \(\eta \in \mathbb{R}\) and the function \(g\) are defined on the boundary \(\partial \Omega\). We require \(\eta > 0\), and recall that the normal vector \(\vec{n}\) is pointing \textit{outwards} \(\Omega\). We consider the case of one space dimension.

In one space dimension \(\frac{\partial u}{\partial n} = \pm u_x\), the sign depends on where the normal vector is pointing (to the right or to the left). We call the relative Initial/Boundary value problem (I/BVP).

\[u_t = u_{xx},\]
\[u(x, 0) = f(x),\]
\[-u_x(0, t) + \eta_0 u(0, t) = g_0(t),\]
\[u_x(1, t) + \eta_1 u(1, t) = g_1(t),\]
\[\eta_0, \eta_1 > 0.\]

\textbf{NB!} \(u(0, t)\) and \(u(1, t)\) are unknown.

Semi-discretization: we choose now \(h = 1/M\) and \(x_m = mh, \ 0 \leq m \leq M\). We get all together \(M + 1\) unknowns \(v_0, \ldots, v_M\) where \(v_m(t) \approx u(x_m, t)\).

\subsection*{3.4.2 Discretization of the boundary conditions}

We look at how the derivatives in the boundary conditions can be discretized. A useful technique is to introduce “fictitious grid lines”; one to the left for the left boundary, i.e. the line \(x = -h, \ t > 0\), and one to the right for the right boundary, that is the line \(x = 1 + h, \ t > 0\).
3.4. BOUNDARY CONDITIONS INVOLVING DERIVATIVES

**Left boundary.** We want to use a difference approximation with truncation error $O(h^2)$ and try the use of central differences, which will involve the solution at the fictitious grid line outside the domain of definition of the solution of the equation.

\[-\partial_x u(0, t) + \eta_0 u(0, t) = g_0(t),\]
\[-\frac{u_1 - u_{-1}}{2h} + \eta_0 u_0 = g_0 + \theta_0,\]

where
\[\theta_0 = -\frac{1}{6} h^2 \partial_x^3 u_0 + \cdots = \text{truncation error}.\]

Here we have $u_{-1} = u(x_{-1}, t) = u(-h, t)$ outside the domain where we look for $u(x, t)$. This might seem a bit dubious, but later on we will see that $u_{-1}$ is eliminated from the discrete equations.

**Right boundary.** Similarly we get

\[\partial_x u(0, t) + \eta_1 u(1, t) = g_1(t),\]
\[\frac{u_{M+1} - u_{M-1}}{2h} + \eta_1 u_M = g_1 + \theta_1,\]

where
\[\theta_1 = \frac{1}{6} h^2 \partial_x^3 u_M + \cdots.\]

The semi-discretization is then

\[
\begin{aligned}
\dot{v}_m &= \frac{1}{h^2} \delta_x^2 v_m, \quad 0 \leq m \leq M, \\
-\frac{v_1 - v_{-1}}{2h} + \eta_0 v_0 &= g_0, \\
-\frac{v_{M+1} - v_{M-1}}{2h} + \eta_1 v_M &= g_1,
\end{aligned}
\tag{3.7}
\]

that is $M + 3$ equations for the $M + 3$ unknowns $v_{-1}, v_0, \ldots, v_M, v_{M+1}$. We eliminate $v_{-1}$ and $v_{M+1}$. From the two last equations in (3.7) we obtain the following equations for the values relative to the fictitious grid lines

\[
v_{-1} = v_1 - 2h \eta_0 v_0 + 2h g_0,
\]
\[
v_{M+1} = v_{M-1} - 2h \eta_1 v_M + 2h g_1.
\]

We substitute the above expressions in the first equation of (3.7) for $m = 0, m = M$

\[
\begin{aligned}
\dot{v}_0 &= \frac{1}{h^2} \delta_x^2 v_0 = \frac{1}{h^2} (v_{-1} - 2v_0 + v_1) \\
&= \frac{1}{h^2} \left( -2(h \eta_0 + 1) v_0 + 2v_1 \right) + \frac{2}{h} g_0, \\
\dot{v}_M &= \frac{1}{h^2} \delta_x^2 v_M = \frac{1}{h^2} (v_{M-1} - 2v_M + v_{M+1}) \\
&= \frac{1}{h^2} \left( 2v_{M-1} - 2(h \eta_1 + 1) v_M \right) + \frac{2}{h} g_1.
\end{aligned}
\]
CHAPTER 3. DISCRETIZATION OF THE HEAT EQUATION

We can write this in matrix-vector notation. With \( \mathbf{v}(t) = [v_0(t), \ldots, v_M(t)]^T \) we get

\[
\dot{\mathbf{v}} = \frac{1}{h^2} Q \mathbf{v} + \frac{2}{h} \mathbf{d},
\]

where

\[
Q = \begin{bmatrix}
-2(h \eta_0 + 1) & 2 & & \\
1 & -2 & 1 & \\
& \ddots & \ddots & \ddots \\
1 & -2 & -2(h \eta_1 + 1) & \\
\end{bmatrix}, \quad \mathbf{d} = \begin{bmatrix} g_0 \\ 0 \\ \vdots \\ g_1 \end{bmatrix}.
\]

We note that this matrix is not symmetric, but it can be obtained from a symmetric matrix via a similarity transformation. With the diagonal matrix \( D = \text{diag}(\sqrt{2}, 1, \ldots, 1, \sqrt{2}) \) we find in fact that \( \tilde{Q} = D^{-1} Q D \) is symmetric and it has therefore real eigenvalues. \( \tilde{Q} \) is also negative definite\(^2\).

To get an example of a fully discrete method, we can use the trapezoidal rule, and obtain Crank–Nicolson. Let \( U^n = (U^n_0, \ldots, U^n_M)^T \), then

\[
U^{n+1} = U^n + \frac{k}{2} \left( \frac{1}{h^2} Q U^n + \frac{2}{h} \mathbf{d} + \frac{1}{h^2} Q U^{n+1} + \frac{2}{h} \mathbf{d}^{n+1} \right),
\]

or in matrix-vector notation

\[
(I - \frac{r}{2} Q) U^{n+1} = (I + \frac{r}{2} Q) U^n + \frac{k}{h} (\mathbf{d}^n + \mathbf{d}^{n+1}).
\]

**Alternative discretization of the boundary conditions.** It is possible to avoid fictitious grid lines and use difference approximations of lower order, obtaining a less accurate approximation

\[
-\partial_x u_0 + \eta_0 u_0 = g_0, \\
\downarrow \\
-\frac{u_1 - u_0}{h} + \eta_0 u_0 = g_0 + \tilde{\theta}_0,
\]

where \( \tilde{\theta}_0 = \frac{1}{2} h \partial_x^2 u_0 + \cdots \), and respectively backward differences for the right boundary,

\[
\partial_x u_M + \eta_1 u_M = g_1 \rightarrow \frac{u_M - u_{M-1}}{h} + \eta_1 u_M = g_1 + \tilde{\theta}_1.
\]

### 3.5 Nonlinear parabolic differential equations

In general one could consider equations of the form

\[
u_t = f(x, t, u, u_x, u_{xx}), \quad \frac{\partial f}{\partial u_{xx}} > 0, \quad \text{initial conditions & boundary conditions.}
\]

We semi-discretize the equation by introducing \( x_m = mh \) and \( v_m(t) \approx u(x_m, t), h = 1/(M + 1) \).

\[
\dot{v}_m = f \left( x_m, t, v_m, \frac{1}{2h} (v_{m+1} - v_{m-1}), \frac{1}{h^2} \delta_x^2 v_m \right).
\]

\(^2\)A matrix \( A \) is negative definite if and only if \(-A\) is positive definite
3.5. NONLINEAR PARABOLIC DIFFERENTIAL EQUATIONS

If we include the boundary conditions, we get a system of ordinary differential equations (ODEs). Taking \( v = [v_1, v_2, \ldots, v_M]^T \) and \( F = [F_1, F_2, \ldots, F_M]^T \) where

\[
F_m = f \left( x_m, t, v_m, \frac{1}{2h} (v_{m+1} - v_{m-1}), \frac{1}{h^2} \partial_x^2 v_m \right).
\]

We have found a nonlinear system of ODEs

\[
\dot{v} = F(t, v),
\]

which can be solved with suitable ODE-solvers (for example in Matlab).

**Burgers’ equation.**

\[
\frac{d}{dt} v = \varepsilon \partial_{xx} v - \partial_t u
\]

It is possible to semi-discretize by taking

\[
F_m = \varepsilon \frac{1}{h^2} (v_{m+1} - 2v_m + v_{m-1}) - v_m \frac{1}{2h} (v_{m+1} - v_{m-1}).
\]

Here it is possible for example to use a Runge-Kutta method applied to \( \dot{v} = F(v) \), see for example > help ode45 in Matlab. Note that Burgers’ equations can be written in the form

\[
\partial_t u = \varepsilon \partial_{xx} u - \frac{1}{2} \partial_u u^2,
\]

which can be discretized directly with central differences, to obtain

\[
F_m = \varepsilon \frac{1}{h^2} (v_{m+1} - 2v_m + v_{m-1}) - \frac{1}{4h} (v_{m+1}^2 - v_{m-1}^2).
\]

**A particular equation type.** Sometimes nonlinear partial differential equations are given in the following form

\[
(b(u) u_t) = (a(u) u_x)_x, \quad b(u) > 0, \quad a(u) > 0.
\]

Here it is possible to use the same strategy as above on the following problem

\[
\frac{d}{dt} u = \frac{a(u)}{b(u)} u_{xx} + \frac{a'(u)}{b(u)} u_x^2.
\]

But a better approach is to let

\[
((a(u) u_x)_x)_m = \frac{1}{h^2} (\partial_x (a \partial_x v))_m = \frac{1}{h^2} (a_{m+1/2} (v_{m+1} - v_m) - a_{m-1/2} (v_m - v_{m-1}))
\]

where

\[
a_{m\pm 1/2} = a(v_{m\pm 1/2}) = a(v(x_m \pm h/2)),
\]

these are quantities which are not defined on the grid. But we can use the following approximation

\[
 u_{m\pm 1/2} = \frac{1}{2} (u_m + u_{m\pm 1}) + O(h^2),
\]

so such approximation has a truncation error of the same order as the truncation error due to the approximation of the derivatives. We next define

\[
a_{m\pm 1/2} = a \left( \frac{v_m + v_{m\pm 1}}{2} \right).
\]

The semi-discretization becomes now

\[
b(v_m) \dot{v}_m = \frac{1}{h^2} \left( a_{m+1/2} (v_{m+1} - v_m) - a_{m-1/2} (v_m - v_{m-1}) \right), \quad \text{truncation error: } O(h^2).
\]
Crank–Nicolson for a nonlinear parabolic problem.

\[ U_{m}^{n+1} = U_{m}^{n} + \frac{k}{2} \left( F_{m}(U_{m}^{n}) + F_{m}(U_{m}^{n+1}) \right), \]

where

\[ F_{m}(U_{m}^{n}) = \frac{1}{b(U_{m}^{n})} \left( \alpha_{m+1/2} \Delta_{x}U_{m}^{n} - \alpha_{m-1/2} \nabla_{x}U_{m}^{n} \right). \]

This means that we must solve a nonlinear equation to compute \( U_{m}^{n+1} \). If we require the same accuracy as for Crank–Nicolson, but we would like to avoid solving nonlinear systems of equations at each time step, we could try and apply a 3-level formula. We use central differences in time and get

\[ \frac{U_{m}^{n+1} - U_{m}^{n-1}}{2k} = F_{m}(U_{m}^{n}), \]

or

\[ U_{m}^{n+1} = U_{m}^{n-1} + 2k F_{m}(U_{m}^{n}). \]

NB! This formula is always unstable for this equation.

**Modification.** In the unstable formula we have

\[ F_{m}(U_{m}^{n}) = \frac{1}{b(U_{m}^{n})} \frac{1}{h^2} \left( \alpha_{m+1/2} \Delta_{x}U_{m}^{n} - \alpha_{m-1/2} \nabla_{x}U_{m}^{n} \right). \]

Replace

\[ \Delta_{x}U_{m}^{n} \rightarrow \frac{1}{3} (\Delta_{x}U_{m}^{n-1} + \Delta_{x}U_{m}^{n} + \Delta_{x}U_{m}^{n+1}), \]

\[ \nabla_{x}U_{m}^{n} \rightarrow \frac{1}{3} (\nabla_{x}U_{m}^{n-1} + \nabla_{x}U_{m}^{n} + \nabla_{x}U_{m}^{n+1}). \]

We get the method

\[ U_{m}^{n+1} = U_{m}^{n-1} + \frac{2}{3} \frac{1}{b(U_{m}^{n})} \left( \alpha_{m+1/2}(\Delta_{x}U_{m}^{n-1} + \Delta_{x}U_{m}^{n} + \Delta_{x}U_{m}^{n+1}) \right. \]

\[ \left. -\alpha_{m-1/2}(\nabla_{x}U_{m}^{n-1} + \nabla_{x}U_{m}^{n} + \nabla_{x}U_{m}^{n+1}) \right), \]

which originally was proposed by Lees. The picture to the left shows the computational molecule for this scheme. The formula is *linearly implicit*, i.e. only one linear system is solved per time-step. Here, as for multi-step methods for ODEs, we need a starting procedure: we need to compute \( U^{1} \) with another method, which should have the same local truncation error as the method used for the rest of the integration. Starting procedures are needed in general in \( p \)-level formulae when \( p > 2 \).