

Numerical solution of Partial Differential Equations

Markus Grasmair

Dec 5, 2022

1 Introduction

In this note, we will discuss the numerical solution of partial differential equations, specifically the one-dimensional wave and heat equations. The approach we are using is the *finite difference method*, which is the same method we have already discussed for the numerical solution of 2-point boundary value problems. In contrast to the situation there, however, we now have to deal with an additional time variable.

Before we proceed with the solution of PDEs, we recall the different finite differences we have discussed previously:

- Forward difference for the first derivative:

$$f'(x) = \frac{f(x+h) - f(x)}{h} + O(h).$$

- Backward difference for the first derivative:

$$f'(x) = \frac{f(x) - f(x-h)}{h} + O(h).$$

- Central difference for the first derivative:

$$f'(x) = \frac{f(x+h) - f(x-h)}{2h} + O(h^2).$$

- Central difference for the second derivative:

$$f''(x) = \frac{f(x+h) - 2f(x) + f(x-h)}{h^2} + O(h^2).$$

2 Wave Equation

We start with the one-dimensional wave equation

$$\partial_{tt}u(x, t) = c^2 \partial_{xx}u(x, t) \quad \text{for } 0 < x < L \text{ and } t > 0$$

with the boundary conditions

$$\left. \begin{array}{l} u(0, t) = 0 \\ u(L, t) = 0 \end{array} \right\} \quad \text{for all } t > 0,$$

and the initial conditions

$$\left. \begin{array}{l} u(x, 0) = f(x) \\ \partial_t u(x, 0) = g(x) \end{array} \right\} \quad \text{for all } 0 < x < L.$$

Here $c > 0$ is a given constant (the *sound speed*), and $f(x)$, $g(x)$ are given functions. This equation describes, amongst others, the vibrations of a string of length L with initial excitation $f(x)$ and initial speed $g(x)$.

We also recall the analytical solution for this equation:

$$u(x, t) = \sum_{n=1}^{\infty} \sin\left(\frac{n\pi x}{L}\right) \left(B_n \cos\left(\frac{cn\pi t}{L}\right) + B_n^* \sin\left(\frac{cn\pi t}{L}\right) \right),$$

where the coefficients B_n are the Fourier-sine coefficients of f , and $\frac{cn\pi}{L} B_n^*$ are the Fourier-sine coefficients of g . Thus we have, in principle, a perfectly fine solution for our equation, and there is no obvious need for an alternative numerical solution. However, the method of separation of variables, which we have used for the derivation of this solution formula, fails or becomes much more complicated if we add additional complications to the equation like a varying sound speed or an inhomogeneity. In contrast, the ideas we use for the numerical solution will still be applicable.

Example. In this example we consider the case $L = 1$, $c = \frac{1}{2}$, and the initial condition

$$\begin{aligned} f(x) &= \sin(2\pi x), \\ g(x) &= \sin(\pi x) - 2\sin(5\pi x). \end{aligned}$$

Since f and g are already given as Fourier-sine series (or: are trigonometric polynomials), we almost immediately get the solution

$$u(x, t) = \frac{2}{\pi} \sin(\pi x) \sin(\pi t/2) + \sin(2\pi x) \cos(\pi t) - \frac{4}{5\pi} \sin(5\pi x) \sin(5\pi t/2).$$

You can find an animated plot of this solution in the jupyter version of this note.

2.1 Finite differences for the wave equation

We now try to follow the same procedure as for 2-point boundary value problems in order to define our numerical method:

Step 1 (discretisation of the domain): Choose a number of discretisation points M on the interval $[0, L]$, define the step length in x -direction $h := L/M$, and define the grid points $x_i = ih$, $i = 0, \dots, M$.

Also, define a step length in t -direction $k > 0$, and define the grid points $t_n = nk$, $n = 0, 1, 2, \dots$

(Do note here that there is no reason to choose the same step length in x - and in t -direction. In fact x is a space variable measured, for instance, in meters, whereas t is a time variable measured, for instance, in seconds. Thus comparing h and k does not make any physical sense, as they carry different units.)

Step 2 (approximation of the derivatives): For each inner point (x_i, t_n) , $i = 1, \dots, M - 1$, $k = 1, 2, \dots$, replace the derivatives in our equation by a suitable finite difference. In our case, we use central differences for the second order derivatives in x - and t -direction, respectively. That is, we approximate

$$\begin{aligned} \partial_{tt} u(x_i, t_n) &= \frac{u(x_i, t_n + k) - 2u(x_i, t_n) + u(x_i, t_n - k)}{k^2} + O(k^2), \\ \partial_{xx} u(x_i, t_n) &= \frac{u(x_i + h, t_n) - 2u(x_i, t_n) + u(x_i - h, t_n)}{h^2} + O(h^2), \end{aligned}$$

and thus obtain

$$\frac{u(x_i, t_n + k) - 2u(x_i, t_n) + u(x_i, t_n - k)}{k^2} + O(k^2) = c^2 \frac{u(x_i + h, t_n) - 2u(x_i, t_n) + u(x_i - h, t_n)}{h^2} + O(h^2)$$

for all $i = 1, \dots, M - 1$ and all $n = 1, 2, \dots$

Step 3 (approximation of the PDE): Ignore the error terms, and replace the exact solution at the discretisation points by a (yet unknown) numerical approximation $U_i^n \approx u(x_i, t_n)$. Here we denote the space indices by subscripts and the time indices by superscripts in order to distinguish more clearly between the two. *Note that the superscript U_i^n does not indicate the n -th power of U_i !*

With this approximation we obtain

$$\frac{U_i^{n+1} - 2U_i^n + U_i^{n-1}}{k^2} = c^2 \frac{U_{i+1}^n - 2U_i^n + U_{i-1}^n}{h^2} \quad (1)$$

for $i = 1, \dots, M - 1$ and $n = 1, 2, \dots$

At this point, it is worth remembering that we are trying to find the solution of a (physically meaningful) time dependent system, where the solution at any given time only depends on what has happened at earlier time points. It thus makes sense to treat the numerical solution in a similar way. In order to compute the solutions at time t_{n+1} we thus assume that we have already computed all the approximations at times t_n, t_{n-1}, \dots . The only unknown in (1) is thus the value U_i^{n+1} . Solving for that value we obtain the explicit expression

$$U_i^{n+1} = 2U_i^n - U_i^{n-1} + c^2 \frac{k^2}{h^2} (U_{i+1}^n - 2U_i^n + U_{i-1}^n) \quad (2)$$

for $i = 1, \dots, M - 1$ and $n = 1, 2, \dots$

Step 4 (boundary conditions): For the indices $i = 1$ and $i = M - 1$, the updates (2) become

$$\begin{aligned} U_1^{n+1} &= 2U_1^n - U_1^0 + c^2 \frac{k^2}{h^2} (U_2^n - 2U_1^n + U_0^n), \\ U_{M-1}^{n+1} &= 2U_{M-1}^n - U_{M-1}^{n-1} + c^2 \frac{k^2}{h^2} (U_M^n - 2U_{M-1}^n + U_{M-2}^n), \end{aligned}$$

which contain the as of yet unknown values U_0^n and U_M^n . Since U_0^n is an approximation of $u(0, t_n)$, and U_M^n is an approximation of $u(x_M, t_n) = u(L, t_n)$, we can use the boundary conditions to determine these values. Before computing the approximations at the time point t_{n+1} , we therefore set

$$U_0^n = 0 \quad \text{and} \quad U_M^n = 0. \quad (3)$$

Step 5 (initial conditions): For the time index $n = 0$, that is, the start of the whole process, we see that the update (2) becomes

$$U_i^1 = 2U_i^0 - U_i^{-1} + c^2 \frac{k^2}{h^2} (U_{i+1}^0 - 2U_i^0 + U_{i-1}^0),$$

which depends on the values $U_i^0, U_{i-1}^0, U_{i+1}^0$, and U_i^{-1} . For these, we have to use the given initial conditions. The first initial condition

$$u(x, 0) = f(x)$$

can be used in the same way as the boundary conditions and yields the values

$$U_i^0 = f(x_i) \quad \text{for } i = 1, \dots, M - 1.$$

For the second initial condition,

$$\partial_t u(x, 0) = g(x),$$

we can use the central difference formula

$$\partial_t u(x, 0) = \frac{u(x, k) - u(x, -k)}{2k} + O(k^2).$$

Equating this to $g(x_i)$ at the grid points x_i provides the numerical approximation

$$\frac{1}{2k} (U_i^1 - U_i^{-1}) = g(x_i)$$

or

$$U_i^{-1} = U_i^1 - 2kg(x_i).$$

We can now insert this expression in our formula for U_i^1 and obtain

$$U_i^1 = 2U_i^0 - (U_i^1 - 2kg(x_i)) + c^2 \frac{k^2}{h^2} (U_{i+1}^0 - 2U_i^0 + U_{i-1}^0).$$

In this equation we still have a term U_i^1 on the right hand side. If we move it to the left hand side and divide the result by 2, we get

$$U_i^1 = U_i^0 + kg(x_i) + c^2 \frac{k^2}{2h^2} (U_{i+1}^0 - 2U_i^0 + U_{i-1}^0),$$

which finally yields an explicit formula for the values U_i^1 , $i = 1, \dots, M - 1$.

2.2 Numerical method

We now summarise the algorithm we have derived:

Initialisation: Choose step sizes $h = L/M$ and $k > 0$, define $x_i = ih$ for $i = 0, \dots, M$, and define $t_n = nk$ for $n = 0, 1, 2, \dots$

Define $U_0^0 = 0$ and $U_M^0 = 0$, and set

$$U_i^0 = f(x_i) \quad \text{for } i = 1, \dots, M - 1.$$

First step: Set $U_0^1 = 0$ and $U_M^1 = 0$.

$$U_i^1 = U_i^0 + kg(x_i) + c^2 \frac{k^2}{2h^2} (U_{i+1}^0 - 2U_i^0 + U_{i-1}^0) \quad \text{for } i = 1, \dots, M - 1.$$

Main loop: For $n = 1, 2, 3, \dots$ do:

- Set $U_0^{n+1} = 0$ and $U_M^{n+1} = 0$.
- Set

$$U_i^{n+1} = 2U_i^n - U_i^{n-1} + c^2 \frac{k^2}{h^2} (U_{i+1}^n - 2U_i^n + U_{i-1}^n) \quad \text{for } i = 1, \dots, M - 1.$$

Result: An array of values U_i^n , $i = 0, \dots, M$, $n = 0, 1, 2, \dots$, where U_i^n is an approximation of the analytical solution $u(x_i, t_n)$.

You can find an implementation of this method in the jupyter version of this note.

Example. We use again the same example as above with $L = 1$, $c = \frac{1}{2}$, and the initial condition

$$\begin{aligned} f(x) &= \sin(2\pi x), \\ g(x) &= \sin(\pi x) - 2\sin(5\pi x). \end{aligned}$$

As we have already seen, this yields the analytic solution

$$u(x, t) = \frac{2}{\pi} \sin(\pi x) \sin(\pi t/2) + \sin(2\pi x) \cos(\pi t) - \frac{4}{5\pi} \sin(5\pi x) \sin(5\pi t/2).$$

You can find an animated plot of this solution in the jupyter version of this note.

2.3 Stability and the CFL condition

If we implement the method above, we see that we obtain a reasonable approximation to the true solution provided that we choose suitable, sufficiently small parameters h and k .

However, something strange happens if we keep the step length k at a constant, small level, but decrease the grid size h : Up to a certain point, the solution improves and the difference to the analytical solution decreases, but at a certain point we suddenly obtain strange oscillations in the solutions that increase rapidly over time. If h is too small, it might even happen that we obtain overflow errors, meaning that

the numerical solutions become larger than $2^{1023} \approx 10^{308}$, which no longer can be represented in double precision. That is, our numerical solution becomes *unstable*.

Moreover, if we try to track the point where this instability occurs more closely, we see that it happens precisely at a point where the ratio h/k becomes smaller than the sound speed c . Put differently, our numerical method is *stable*, and thus produces sensible results, if and only if the condition

$$c \frac{k}{h} \leq 1$$

holds. This stability condition is called the *Courant–Friedrichs–Lewy condition*, or simply *CFL-condition*.

A condition of a similar form will almost always be required when we want to solve a time-dependent PDE with an explicit numerical method (that is, a method where we obtain the next time steps without having to solve a system of equations). In particular, this means that if we want to decrease the grid size h (and improve the space resolution), we have to simultaneously decrease the step length k (and compute a larger number of time steps).

In the jupyter version of this note, you can find an example of the instability that occurs if the CFL-condition is violated.

3 Heat equation

We now look at the one-dimensional heat equation

$$\partial_t u(x, t) = c^2 \partial_{xx} u(x, t) \quad \text{for } 0 < x < L \text{ and } t > 0,$$

which describes, amongst others, the time-dependent temperature distribution in a thin, insulated wire or rod of length L . The parameter $c^2 > 0$ is the thermal diffusivity in the wire.

For a unique solution of the PDE (or a complete description of the temperature in the wire), we need in addition suitable boundary and initial conditions. For the initial condition we assume that we know the temperature distribution in the wire at time $t = 0$, that is,

$$u(x, 0) = f(x) \quad \text{for } 0 < x < L,$$

where $f(x)$ is some given function.

For the boundary condition, there are different possibilities:

Dirichlet boundary conditions: Here we assume that the endpoints of the wire are held at temperatures $g_0(t)$ and $g_L(t)$, respectively. This results in boundary conditions of the form

$$u(0, t) = g_0(t) \quad \text{and} \quad u(L, t) = g_L(t) \quad \text{for } t > 0.$$

Neumann boundary conditions: Here we assume that the temperature flux $\partial_x u$ at the endpoints is given, which results in

$$\partial_x u(0, t) = g_0(t) \quad \text{and} \quad \partial_x u(L, t) = g_L(t) \quad \text{for } t > 0.$$

Of particular interest is that of *homogeneous* Neumann boundary conditions

$$\partial_x u(0, t) = 0 \quad \text{and} \quad \partial_x u(L, t) = 0 \quad \text{for } t > 0.$$

Here we do not have any temperature flux through the endpoints of the wire; in other words the endpoints are insulated.

Robin (or mixed) boundary conditions: These are as a linear combination of Dirichlet and Neumann boundary conditions:

$$a_0 \partial_x u(0, t) + b_0 u(0, t) = g_0(t) \quad \text{and} \quad a_L \partial_x u(L, t) + b_L u(L, t) = g_L(t) \quad \text{for } t > 0,$$

with parameters $a_0, b_0, a_L, b_L \neq 0$.

They appear, for instance, in the case where the endpoints of the wire are imperfectly insulated and we have some heat flux through the insulation. In this case, one obtains boundary conditions of the form

$$\partial_x u(0, t) = a(u(0, t) - g_0(t)) \quad \text{and} \quad \partial_x u(L, t) = a(g_L(t) - u(L, t)) \quad \text{for } t > 0,$$

where $g_0(t)$ and $g_L(t)$ are the outside temperatures at the points $x = 0$ and $x = L$, respectively, and a is the thermal diffusivity of the insulation.

Other possibilities: It is also possible that we have different types of boundary conditions in the two endpoints. For instance, the right endpoint of the wire might be insulated, whereas the left endpoint is kept at a fixed temperature. In this case, we would have the conditions $u(0, t) = g_0(t)$ and $\partial_x u(L, t) = 0$.

3.1 Explicit Euler method

We now construct a numerical solution method based on the same approach as for the wave equation.

Step 1 (discretisation of the domain): Choose a number of discretisation points M on the interval $[0, L]$, define the step length in x -direction $h := L/M$, and define the grid points $x_i = ih$, $i = 0, \dots, M$.

Also, define a step length in t -direction $k > 0$, and define the grid points $t_n = nk$, $n = 0, 1, 2, \dots$

Step 2 (approximation of the derivatives): For each inner point (x_i, t_n) , $i = 1, \dots, M - 1$, $k = 1, 2, \dots$, replace the derivatives in our equation by a suitable finite difference. In our case, we use central differences for the second order derivatives in x -direction, which means that we approximate

$$\partial_{xx} u(x_i, t_n) = \frac{u(x_i + h, t_n) - 2u(x_i, t_n) + u(x_i - h, t_n)}{h^2} + O(h^2) \quad (4)$$

For the time derivative, we have different choices. We start with the simplest case of a forward difference

$$\partial_t u(x_i, t_n) = \frac{u(x_i, t_n + k) - u(x_i, t_n)}{k} + O(k).$$

Then we obtain the approximation

$$\frac{u(x_i, t_n + k) - u(x_i, t_n)}{k} + O(k) = \frac{u(x_i + h, t_n) - 2u(x_i, t_n) + u(x_i - h, t_n)}{h^2} + O(h^2)$$

for all $i = 1, \dots, M - 1$ and all $n = 1, 2, \dots$

Step 3 (approximation of the PDE): Ignore the error terms, and replace the exact solution at the discretisation points by a (yet unknown) numerical approximation $U_i^n \approx u(x_i, t_n)$. We then obtain

$$\frac{U_i^{n+1} - U_i^n}{k} = c^2 \frac{U_{i+1}^n - 2U_i^n + U_{i-1}^n}{h^2} \quad (5)$$

for $i = 1, \dots, M - 1$ and $n = 1, 2, \dots$

Again, we have a time dependent equation, which we solve forward in time. Solving (5) for U_i^{n+1} , we obtain the explicit expression

$$U_i^{n+1} = U_i^n + c^2 \frac{k}{h^2} (U_{i+1}^n - 2U_i^n + U_{i-1}^n) \quad (6)$$

for $i = 1, \dots, M - 1$ and $n = 1, 2, \dots$

Step 4 (boundary conditions): As for the wave equation, we need to use the boundary conditions in order to be able to compute the updates for U_1^{n+1} and U_{M-1}^{n+1} . However, we have three different types of boundary conditions that all have to be treated slightly differently:

Dirichlet boundary conditions: That is, conditions of the form

$$u(0, t) = g_0(t) \quad \text{and} \quad u(L, t) = g_L(t).$$

This is the simplest case and uses exactly the same idea as our discretisation of the wave equation above. We simply define

$$U_0^n = g_0(t_n) \quad \text{and} \quad U_M^n = g_L(t_n) \quad \text{for } n = 1, 2, 3, \dots$$

Neumann boundary conditions: That is, conditions of the form

$$\partial_x u(0, t) = g_0(t) \quad \text{and} \quad \partial_x u(L, t) = g_L(t).$$

In this case, the values U_0^{n+1} and U_M^{n+1} are not directly given, but rather have to be computed. For that, we use a similar idea as for the first step in the numerical solution of the wave equation.

We will first look at the point U_0^{n+1} . From the PDE we obtain here the equation

$$U_0^{n+1} = U_0^n + c^2 \frac{k}{h^2} (U_{-1}^n - 2U_0^n + U_{+1}^n). \quad (7)$$

At this point, we assume that the value U_0^n has already been computed, so the only unknown is the value U_{-1}^n . For that we use the boundary condition, which we write, using central differences, as

$$g_0(t_n) = \partial_x u(0, t_n) = \frac{u(h, t_n) - u(-h, t_n)}{2h} + O(h^2).$$

From this we obtain the numerical approximation

$$\frac{U_1^n - U_{-1}^n}{2h} = g_0(t_n),$$

or

$$U_{-1}^n = U_1^n + 2hg_0(t_n).$$

We can insert this into (7) and obtain the expression

$$U_0^{n+1} = U_0^n + 2c^2 \frac{k}{h^2} (U_1^n - U_0^n + hg_0(t_n)) \quad \text{for } n = 0, 1, 2, 3, \dots$$

For the point U_M^{n+1} we can use a similar argumentation, using the approximation

$$\frac{U_{M+1}^n - U_{M-1}^n}{2h} = g_L(t_n).$$

Solving this for U_{M+1}^n and inserting the result into the update for U_M^{n+1} yields the expression

$$U_M^{n+1} = U_0^n + 2c^2 \frac{k}{h^2} (U_{M-1}^n - U_M^n - hg_L(t_n)).$$

Robin boundary conditions: That is, conditions of the form

$$a_0 \partial_x u(0, t) + b_0 u(0, t) = g_0(t) \quad \text{and} \quad a_L \partial_x u(L, t) + b_L u(L, t) = g_L(t) \quad \text{for } t > 0.$$

Here we can use the same idea as for Neumann boundary conditions: For the point U_0^{n+1} , we obtain from the PDE the equation

$$U_0^{n+1} = U_0^n + c^2 \frac{k}{h^2} (U_{-1}^n - 2U_0^n + U_{+1}^n).$$

Moreover, discretising the boundary condition at $x = 0$ yields the approximation

$$a_0 \frac{U_1^n - U_{-1}^n}{2h} + b_0 U_0^n = g_0(t_n).$$

We solve this equation for U_{-1}^n and obtain

$$U_{-1}^n = U_1^n + \frac{2h}{a_0} (b_0 U_0^n - g_0(t_n)).$$

Next we insert this expression in the formula for U_0^{n+1} and obtain

$$U_0^{n+1} = U_0^n + 2c^2 \frac{k}{h^2} \left(U_{+1}^n - U_0^n + \frac{hb_0}{a_0} U_0^n - \frac{h}{a_0} g_0(t_n) \right).$$

For the point U_M^{n+1} we use the same approach and obtain

$$U_M^{n+1} = U_M^n + 2c^2 \frac{k}{h^2} \left(U_{M-1}^n - U_M^n - \frac{hb_L}{a_L} U_M^n + \frac{h}{a_L} g_L(t_n) \right).$$

Step 5 (initial conditions): In order to start the iterations, we require the values U_i^0 , $i = 0, \dots, M$. For that we simply use the given initial condition and define

$$U_i^0 = f(x_i) \quad \text{for } i = 0, \dots, M.$$

Resulting algorithms. To summarise, we obtain for the different boundary conditions the following algorithms:

Dirichlet boundary conditions: Here the resulting algorithm thus reads as follows:

- Choose step sizes $h = L/M$ and $k > 0$, define $x_i = ih$ for $i = 1, \dots, M-1$, and define $t_n = nk$ for $n = 1, 2, \dots$
- For $i = 0, \dots, M$ define $U_i^0 := f(x_i)$.
- For $n = 0, 1, 2, \dots$ do:
 - Define $U_0^{n+1} = g_0(t_{n+1})$ and $U_M^{n+1} = g_L(t_{n+1})$.
 - For $i = 1, \dots, M$ define

$$U_i^{n+1} = U_i^n + c^2 \frac{k}{h^2} (U_{i-1}^n - 2U_i^n + U_{i+1}^n).$$

Neumann boundary conditions: Here the algorithms is as follows:

- Choose step sizes $h = L/M$ and $k > 0$, define $x_i = ih$ for $i = 1, \dots, M-1$, and define $t_n = nk$ for $n = 1, 2, \dots$
- For $i = 0, \dots, M$ define $U_i^0 := f(x_i)$.
- For $n = 0, 1, 2, \dots$ do:
 - Define $U_0^{n+1} = g_0(t_{n+1})$ and $U_M^{n+1} = g_L(t_{n+1})$.
 - For $i = 1, \dots, M$ define

$$U_0^{n+1} = U_0^n + 2c^2 \frac{k}{h^2} (U_1^n - U_0^n + hg_0(t_n)),$$

$$U_i^{n+1} = U_i^n + c^2 \frac{k}{h^2} (U_{i-1}^n - 2U_i^n + U_{i+1}^n) \quad \text{for } i = 1, \dots, M-1,$$

$$U_M^{n+1} = U_M^n + 2c^2 \frac{k}{h^2} (U_{M-1}^n - U_M^n - hg_L(t_n)).$$

Example: Neumann boundary conditions. In this example we consider the case $L = 1$, $c^2 = 1/4$, the initial condition

$$f(x) = -\cos(2\pi x) + \cos(4\pi x) + \cos(9\pi x),$$

and *homogeneous Neumann boundary conditions*

$$\partial_x u(0, t) = 0 = \partial_x u(1, t).$$

In this case, we have the analytical solution

$$u(x, t) = -\cos(2\pi x)e^{-\pi^2 t} + \cos(4\pi x)e^{-4\pi^2 t} + \cos(9\pi x)e^{-81\pi^2 t/4},$$

which we can find by separation of variables.

In the jupyter version of this note, you can find a comparison of the analytical solution with the numerical result obtained with the algorithm above. The results show that the method works in principle, but also that we have to be very careful with the choice of the step size k , if we want to avoid instabilities.

Instability. From the numerical experiments, we see that the solutions become unstable at a certain point. In order to understand this effect better, we consider specifically the case of the heat equation with homogeneous Neumann boundary conditions $\partial_x u(0, t) = 0 = \partial_x u(L, t)$. In this case, the updates become

$$\begin{aligned} U_0^{n+1} &= U_0^n + 2r(U_1^n - U_0^n), \\ U_i^{n+1} &= U_0^n + r(U_{i-1}^n - 2U_i^n + U_{i+1}^n), & \text{for } i = 1, \dots, M-1, \\ U_M^{n+1} &= U_M^n + 2r(U_{M-1}^n - U_M^n), \end{aligned}$$

where we have used the abbreviation

$$r = c^2 \frac{k}{h^2}.$$

Now define the matrix

$$A = \begin{pmatrix} -2 & 2 & 0 & \dots & & \dots & 0 \\ 1 & -2 & 1 & 0 & \dots & & 0 \\ 0 & 1 & -2 & 1 & 0 & \dots & 0 \\ \vdots & \ddots & \ddots & \ddots & \ddots & \ddots & \vdots \\ \vdots & & \ddots & \ddots & \ddots & \ddots & \vdots \\ 0 & \dots & \dots & 0 & 1 & -2 & 1 & 0 \\ 0 & \dots & & \dots & 0 & 1 & -2 & 1 \\ 0 & \dots & & \dots & 0 & 2 & -2 \end{pmatrix} \in \mathbb{R}^{(M+1) \times (M+1)}. \quad (8)$$

Let moreover $I_{M+1} \in \mathbb{R}^{(M+1) \times (M+1)}$ be the $(M+1)$ -dimensional identity matrix, that is,

$$I_{M+1} = \begin{pmatrix} 1 & 0 & \dots & \dots & 0 \\ 0 & 1 & 0 & \dots & 0 \\ \vdots & \ddots & \ddots & \ddots & \vdots \\ 0 & \dots & 0 & 1 & 0 \\ 0 & \dots & \dots & 0 & 1 \end{pmatrix} \in \mathbb{R}^{(M+1) \times (M+1)}.$$

Then we can also write the updates as

$$\vec{U}^{n+1} = (I_{M+1} + rA)\vec{U}^n.$$

Consider now in particular the vector $\vec{V} = (1, -1, 1, -1, \dots) \in \mathbb{R}^{M+1}$. Then it is easy to check that

$$(I_{M+1} + rA)\vec{V} = (1 - 4r)\vec{V}.$$

In other words, the vector \vec{V} is an eigenvector of the matrix $(I_{M+1} + rA)$ for the eigenvalue $(1 - 4r)$. Now assume that we are given the initial values $\vec{U}^0 = \vec{V}$. Then we obtain the iterates

$$\vec{U}^n = (1 - 4r)^n \vec{V}.$$

From this, we see that the solution \vec{U}^n remains bounded provided that $|1 - 4r| \leq 1$. Else, we have that $\|\vec{U}^n\| \rightarrow \infty$ as $n \rightarrow \infty$, which means that we have an unstable solution.

Now, the condition $|1 - 4r| \leq 1$ is equivalent to the two inequalities

$$1 - 4r \leq 1 \quad \text{and} \quad 1 - 4r \geq -1.$$

The first of these conditions is always satisfied, as $r > 0$. From the second inequality, we obtain the condition

$$r = c^2 \frac{k}{h^2} \leq \frac{1}{2}.$$

If this condition fails to hold, then the solution will become unstable at some point. Conversely, one can show that the solution remains stable if this condition is satisfied.

In the case of Dirichlet or Robin boundary conditions, the situation is similar and one again obtains a stable solution as long as $r \leq 1/2$.

3.2 Implicit Euler method

As we have seen in the example above, the explicit Euler method becomes unstable if the grid size h is chosen to small compared to the step length k . This should not come as a surprise to us, since we have already observed this instability for the wave equation. However, if we take a closer look at the conditions, we see that the situation for the heat equation is much worse than that for the wave equation: For the wave equation, the CFL-condition required that the ratio h/k had to be larger than the sound speed c . In other words, if we halve the grid size h , then we have to halve the step length k as well in order to maintain stability. In contrast, in the case of the heat equation the stability condition is an upper bound for the ratio h^2/k . Halving the grid size h therefore requires quartering the step length k . More drastically, if we multiply the grid size h by a factor $1/10$, we have to simultaneously multiply the step length by a factor $1/100$. Put differently, we have to perform 100-times as many time steps in order to obtain an approximation at the same given time point t . This makes the computational effort prohibitively large if one requires a solution with a fine space discretisation. Because of that, we will discuss alternative *implicit* methods that are usually used for solving the heat equation and similar PDEs.

For the derivation of the explicit Euler method, we have approximated the time derivative $\partial_t u(x_i, t_n)$ using a forward difference. Instead, we can also use a backward difference

$$\partial_t u(x_i, t_n) = \frac{u(x_i, t_n) - u(x_i, t_n - k)}{k} + O(k) = \frac{u(x_i, t_n) - u(x_i, t_{n-1})}{k} + O(k). \quad (9)$$

With this, we obtain the approximation of the PDE

$$\frac{U_i^n - U_i^{n-1}}{k} = c^2 \frac{U_{i+1}^n - 2U_i^n + U_{i-1}^n}{h^2}.$$

Or, if we shift the time indices by one (and replace n by $n + 1$, and $n - 1$ by n),

$$\frac{U_i^{n+1} - U_i^n}{k} = c^2 \frac{U_{i+1}^{n+1} - 2U_i^{n+1} + U_{i-1}^{n+1}}{h^2}.$$

Here it is no longer possible to obtain an explicit expression for U_i^{n+1} that only depends on the (already computed) values at the time t_n . Instead, the values U_i^{n+1} , $i = 1, \dots, n$, are *implicitly* given as the solution of the system of equations

$$U_i^{n+1} - c^2 \frac{k}{h^2} (U_{i+1}^{n+1} - 2U_i^{n+1} + U_{i-1}^{n+1}) = U_i^n, \quad i = 1, \dots, M - 1. \quad (10)$$

This gives us $M - 1$ equations for the $M + 1$ unknown values $U_0^{n+1}, \dots, U_{M+1}^{n+1}$. In addition, we obtain two equations from the boundary conditions. For simplicity, we will only discuss Dirichlet and Neumann conditions. Robin conditions can be treated similarly, but the resulting equations are somewhat more complicated.

Dirichlet conditions: If we are given conditions

$$u(0, t) = g_0(t) \quad \text{and} \quad u(L, t) = g_L(t),$$

we use them directly and obtain the explicit expressions

$$U_0^{n+1} = g_0(t_{n+1}) \quad \text{and} \quad U_M^{n+1} = g_L(t_{n+1}).$$

We can now insert these values in the equations (10) with $i = 1$ and $i = M - 1$ and obtain the new equations

$$\begin{aligned} U_1^{n+1} - c^2 \frac{k}{h^2} (U_2^{n+1} - 2U_1^{n+1}) &= U_1^n + c^2 \frac{k}{h^2} g_0(t_{n+1}), \\ U_{M-1}^{n+1} - c^2 \frac{k}{h^2} (U_{M-2}^{n+1} - 2U_{M-1}^{n+1}) &= U_{M-1}^n + c^2 \frac{k}{h^2} g_L(t_{n+1}). \end{aligned}$$

This gives us a system of $M - 1$ equations for the unknowns $U_1^{n+1}, \dots, U_{M-1}^{n+1}$, which we have to solve in order to find the approximation at time t_{n+1} .

For the numerical solution, we will write it in matrix form. To that end, we abbreviate again

$$r = c^2 \frac{k}{h^2},$$

and we define the matrix

$$B = \begin{pmatrix} -2 & 1 & 0 & \cdots & & & \cdots & 0 \\ 1 & -2 & 1 & 0 & \cdots & & \cdots & 0 \\ 0 & 1 & -2 & 1 & 0 & \cdots & \cdots & 0 \\ \vdots & \ddots & \ddots & \ddots & \ddots & \ddots & & \vdots \\ \vdots & & \ddots & \ddots & \ddots & \ddots & \ddots & \vdots \\ 0 & \cdots & \cdots & 0 & 1 & -2 & 1 & 0 \\ 0 & \cdots & & \cdots & 0 & 1 & -2 & 1 \\ 0 & \cdots & & & \cdots & 0 & 1 & -2 \end{pmatrix} \in \mathbb{R}^{(M-1) \times (M-1)} \quad (11)$$

and

$$\vec{b}^{n+1} = \begin{pmatrix} g_0(t_{n+1}) \\ 0 \\ 0 \\ \vdots \\ 0 \\ g_L(t_{n+1}) \end{pmatrix} \in \mathbb{R}^{M-1}. \quad (12)$$

Then the vector $\vec{U}^{n+1} = (U_1^{n+1}, U_2^{n+1}, \dots, U_{M-1}^{n+1})$ solves the equation

$$(I_{M-1} - rB)\vec{U}^{n+1} = \vec{U}^n + r\vec{b}^{n+1},$$

with $I_{M-1} \in \mathbb{R}^{(M-1) \times (M-1)}$ being the $(M - 1)$ -dimensional identity matrix.

The resulting algorithm thus reads as follows:

- Choose step sizes $h = L/M$ and $k > 0$, define $x_i = ih$ for $i = 1, \dots, M - 1$, and define $t_n = nk$ for $n = 1, 2, \dots$
- Define the matrix B as in (11).
- For $i = 1, \dots, M - 1$ define $U_i^0 := f(x_i)$.
- For $n = 0, 1, 2, \dots$ do:

Compute \vec{b}^{n+1} as in (12).

Compute the solution \vec{U}^{n+1} of the equation $(I_{M-1} - rB)\vec{U}^{n+1} = \vec{U}^n + r\vec{b}^{n+1}$.

Neumann conditions: That is, conditions of the form

$$\partial_x u(0, t) = g_0(t), \quad \text{and} \quad \partial_x u(L, t) = g_L(t).$$

Here we use the same approach as for the explicit Euler method with Neumann boundary conditions. After a brief computation, we then obtain the system of linear equations

$$\begin{aligned} U_0^{n+1} - 2c^2 \frac{k}{h^2} (U_1^{n+1} - U_0^{n+1}) &= U_0^n + 2c^2 \frac{k}{h} g_0(t_{n+1}), \\ U_i^{n+1} - 2c^2 \frac{k}{h^2} (U_{i-1}^{n+1} - 2U_i^{n+1} + U_{i+1}^{n+1}) &= U_i^n, & \text{for } i = 1, \dots, M-1, \\ U_M^{n+1} - 2c^2 \frac{k}{h^2} (U_{M-1}^{n+1} - U_M^{n+1}) &= U_M^n - 2c^2 \frac{k}{h} g_L(t_{n+1}), \end{aligned}$$

Again, we can write this as a matrix vector system. To that end, let A be the matrix from (8) and let

$$\vec{a}^{n+1} = \begin{pmatrix} g_0(t_{n+1}) \\ 0 \\ \vdots \\ 0 \\ -g_L(t_{n+1}) \end{pmatrix} \in \mathbb{R}^{M+1}. \quad (13)$$

Then the vector $\vec{U}^{n+1} = (U_0^{n+1}, U_1^{n+1}, \dots, U_M^{n+1}) \in \mathbb{R}^{M+1}$ solves the equation

$$(I_{M+1} - rA)\vec{U}^{n+1} = \vec{U}^n + 2rh\vec{a}^{n+1}.$$

Thus we obtain the following algorithm:

- Choose step sizes $h = L/M$ and $k > 0$, define $x_i = ih$ for $i = 0, \dots, M$, and define $t_n = nk$ for $n = 1, 2, \dots$
- Define the matrix A as in (8).
- For $i = 0, \dots, M$ define $U_i^0 := f(x_i)$.
- For $n = 0, 1, 2, \dots$ do:

Compute \vec{a}^{n+1} as in (13).

Compute the solution \vec{U}^{n+1} of the equation $(I_{M+1} - rA)\vec{U}^{n+1} = \vec{U}^n + 2rh\vec{a}^{n+1}$.

Numerical solution of the linear equations. In the implicit Euler method, we have to solve a linear equation in every single step. At first glance, this appears to increase the numerical effort significantly, as the solution of an $M \times M$ system (with the Gauss algorithm) in general requires $O(M^3)$ floating point operations. For the particular type of linear systems that occur for the heat equation and similar PDEs, however, the situation is different. The matrices $I - rA$ and $I - rB$ that appear in the implicit Euler method for the heat equation are both *tri-diagonal* in the sense that only the central three diagonals of these matrices contain non-zero elements. For this type of matrices, the Gauss algorithm turns out to be much simpler in practice and only require $O(M)$ floating point operations. Thus the implicit Euler method is in practice only slightly slower than the explicit Euler method, although a linear system has to be solved in each step.

However, if one actually implements the method, one has to make sure to use a suitable linear solver where tri-diagonal linear systems are implemented efficiently. Because of that, we will make use of the `sparse` module in `scipy` for our implementation.

Example. We use the same example as above, that is, $L = 1$, $c^2 = 1/4$, the initial condition

$$f(x) = -\cos(2\pi x) + \cos(4\pi x) + \cos(9\pi x),$$

and homogeneous Neumann boundary conditions

$$\partial_x u(0, t) = 0 = \partial_x u(1, t).$$

In the jupyter version of this note, you can find a comparison of the analytical solution with the numerical result obtained with the algorithm above. The result shows that the solution is stable even though we have chosen a relatively large step length k compared to the grid size h . However, the numerical solution is somewhat inaccurate, especially for times t close to zero.

3.3 Crank–Nicolson method

The implicit Euler method for the heat equation is *unconditionally stable*. That is, we can in principle choose arbitrarily large step sizes without encountering any instabilities. However, in practice this does not make sense, as the quality of the numerical approximation depends on the size of h and k . If we want to have a sufficiently good approximation of the analytical solution of the PDE, we still have to choose suitably small values of h and k .

In the case of the implicit Euler method, the approximation errors for the numerical derivatives in (4) and (9) were of order $O(h^2)$ and $O(k)$, respectively, we would expect that the error in the solution behaves in the same way and depends quadratically on h and linearly on k . Thus, if we want to quarter the numerical approximation error, we have to quarter the step length k , while we only have to halve the grid size h . Although we do not have any stability problems, this means in practice that the step length k should again behave like h^2 .

The idea of the *Crank-Nicolson method* is to combine the updates for the explicit and the implicit Euler methods. For instance, in the case of Neuman boundary conditions, the explicit and implicit Euler methods read as

$$\begin{aligned}\vec{U}^{n+1} &= (I_{M+1} + rA)\vec{U}^n + 2rh\vec{a}^n, \\ (I_{M+1} - rA)\vec{U}^{n+1} &= \vec{U}^n + 2rh\vec{a}^{n+1},\end{aligned}$$

respectively. For the Crank-Nicolson method, we take the average of these two updates and obtain the new equation

$$\left(I_{M+1} - \frac{r}{2}A\right)\vec{U}^{n+1} = \left(I_{M+1} + \frac{r}{2}A\right)\vec{U}^n + rh(\vec{a}^n + \vec{a}^{n+1}).$$

One can show that the numerical error for this method is of order $O(h^2 + k^2)$.

Dirichlet conditions: In the case of Dirichlet boundary conditions, the method reads as:

- Choose step sizes $h = L/M$ and $k > 0$, define $x_i = ih$ for $i = 1, \dots, M - 1$, and define $t_n = nk$ for $n = 1, 2, \dots$
- Define the matrix B as in (11).
- For $i = 1, \dots, M - 1$ define $U_i^0 := f(x_i)$.
- For $n = 0, 1, 2, \dots$ do:

Compute \vec{b}^n and \vec{b}^{n+1} as in (12).

Compute the solution \vec{U}^{n+1} of the equation

$$\left(I_{M-1} - \frac{r}{2}B\right)\vec{U}^{n+1} = \left(I_{M-1} + \frac{r}{2}B\right)\vec{U}^n + \frac{r}{2}(\vec{b}^n + \vec{b}^{n+1}).$$

Neumann conditions: For Neuman boundary conditions, the algorithm becomes:

- Choose step sizes $h = L/M$ and $k > 0$, define $x_i = ih$ for $i = 0, \dots, M$, and define $t_n = nk$ for $n = 1, 2, \dots$
- Define the matrix A as in (8).
- For $i = 0, \dots, M$ define $U_i^0 := f(x_i)$.
- For $n = 0, 1, 2, \dots$ do:
 - Compute \bar{a}^n and \bar{a}^{n+1} as in (13).
 - Compute the solution \vec{U}^{n+1} of the equation

$$\left(I_{M+1} - \frac{r}{2}A\right)\vec{U}^{n+1} = \left(I_{M+1} + \frac{r}{2}A\right)\vec{U}^n + rh(\bar{a}^n + \bar{a}^{n+1}).$$

Example. We use again the same example as for both the explicit Euler method and the implicit Euler method.

In the jupyter version of this note, you can find a comparison of the analytical solution with the numerical result obtained with the algorithm above. The result shows that the solution is stable even though we have chosen a relatively large step length k compared to the grid size h . Also, the result is much more accurate for the same grid size and step length than that for the implicit Euler method, although the computational cost is only slightly larger.