Chapter 4

Introduction to numerical methods for partial differential equations

4.1 Introduction

Numerical methods involve transforming continuous analytical problems into discrete numerical ones, and there exists a number of discretization techniques for any given equation. In this chapter, we introduce the finite difference method, which is widely used for approximating partial differential equations (PDEs) using a computer. The finite difference method is a common technique for finding approximate solutions to partial differential equations. It involves solving a system of relations (numerical scheme) that connects the values of unknown functions at points that are sufficiently close to each other. At first glance, this method seems to be the simplest to implement, as it proceeds in two steps: first, the discretization of the differentiation operators using finite differences, and second, the convergence of the resulting numerical scheme as the distance between the points decreases.

4.2 Approximating the derivatives of a function by finite differences

In previous courses, you were introduced to the notion of the derivative for a differentiable function and the Taylor's formula.

Definition 4.2.1 (**Derivative of a function and Taylor's formula).** *Assume that the function* $f : \mathbb{R} \to \mathbb{R}$ *is differentiable, then*

$$f'(x) = \lim_{\Delta x \to 0} \frac{f(x + \Delta x) - f(x)}{\Delta x} \tag{4.1}$$

If we assume that the function can be differentiated many times then Taylor's formula is given by:

$$f(x + \Delta x) = f(x) + \Delta x f'(x) + \frac{\Delta x^2}{2!} f''(x) + \frac{\Delta x^3}{3!} f^{(3)}(x) + \dots \tag{4.2}$$

or, with $-\Delta x$ *instead of* $+\Delta x$ *:*

$$f(x - \Delta x) = f(x) - \Delta x f'(x) + \frac{\Delta x^2}{2!} f''(x) - \frac{\Delta x^3}{3!} f^{(3)}(x) + \dots \tag{4.3}$$

Three basic types of finite difference methods are commonly considered: forward, backward, and central finite differences.

Definition 4.2.2 (Forward difference). *On a computer, derivatives are approximated by finite difference expressions; rearranging (4.2) gives the forward difference approximation*

$$\frac{f(x+\Delta x)-f(x)}{\Delta x} = f'(x) + O(\Delta x), \tag{4.4}$$

where $O(\Delta x)$ means 'terms of order Δx ', i.e. terms which have size similar to or smaller than Δx when Δx is small. Technically, a term or function $E(\Delta x)$ is $O(\Delta x)$ if

$$\lim_{\Delta x \to 0} \frac{E(\Delta x)}{\Delta x} = \text{ constant.}$$

*So the expression on the left approximates the derivative of f at x, and has an error of size Δx; the approximation is said to be '***first order accurate***'.*

Definition 4.2.3 (Backward difference). *Rearranging (4.3) similarly gives the backward difference approximation*

$$\frac{f(x)-f(x-\Delta x)}{\Delta x} = f'(x) + O(\Delta x) \tag{4.5}$$

which is also first order accurate, since the error is of order Δx.

Definition 4.2.4 (Central difference). *Combining (4.2) and (4.3) gives the central difference approximation*

$$\frac{f(x+\Delta x)-f(x-\Delta x)}{2\Delta x} = f'(x) + O\left(\Delta x^2\right) \tag{4.6}$$

*which is '***second order accurate***', because the error this time is of order Δx^2.*

Definition 4.2.5 (Second derivative, central difference). *Adding (4.2) and (4.3) gives*

$$f(x+\Delta x) + f(x-\Delta x) = 2f(x) + \Delta x^2 f''(x) + \frac{\Delta x^4}{12} f^{(4)}(x) + \dots \tag{4.7}$$

Rearranging this therefore gives the central difference approximation to the second derivative:

$$\frac{f(x+\Delta x) - 2f(x) + f(x-\Delta x)}{\Delta x^2} = f''(x) + O\left(\Delta x^2\right) \tag{4.8}$$

which is second order accurate.

How many boundary conditions are needed to solve a PDE?

Typically, for a PDE, to obtain a unique solution, we need one condition (either boundary or initial) for each derivative in each variable. For instance:

- **The heat equation:**

$$u_t = u_{xx},$$

involves one time derivative and two spatial derivatives, meaning we require:

- One initial condition (IC)
- Two boundary conditions (BCs)

- **The wave equation:**

$$u_{tt} = u_{xx}$$

involves two time derivatives and two spatial derivatives, meaning we require:

- Two initial conditions (ICs)
- Two boundary conditions (BCs)

- **Laplace's equation:**

$$u_{xx} + u_{yy} = 0$$

involves two spatial derivatives in both the x and y directions, meaning we require:

- Four boundary conditions (BCs)

4.3 Solving the heat equation using the method of finite differences

4.3.1 Dirichlet boundary conditions

Methodology 4.3.1 (Dirichlet boundary conditions). *To find a numerical solution to the heat equation:*

$$\text{PDE}: \quad \frac{\partial u}{\partial t} = \alpha^2 \frac{\partial^2 u}{\partial x^2}, \quad 0 < x < L, \ t > 0 \tag{4.9a}$$

$$\text{BC}: \quad u(0,t) = A, \quad u(L,t) = B \tag{4.9b}$$

$$\text{IC}: \quad u(x,0) = f(x) \tag{4.9c}$$

we approximate the time derivative using forward differences, and the spatial derivative using central differences;

$$\frac{u(x,t+\Delta t) - u(x,t)}{\Delta t} = \alpha^2 \frac{u(x+\Delta x,t) - 2u(x,t) + u(x-\Delta x,t)}{\Delta x^2} + O\left(\Delta t, \Delta x^2\right). \tag{4.10}$$

This approximation is second order accurate in space and first order accurate in time. The use of the forward difference means the method is explicit, because it gives an explicit formula for $u(x,t+\Delta t)$ depending only on the values of u at time t.

Divide the interval $0 < x < L$ into $N+1$ evenly spaced points, with spacing Δx; ie. $x_n = n\Delta x$, for $n = 0, 1, \ldots, N$, and divide the time interval $[0, T]$ into $M+1$ equal time levels $t_k = k\Delta t$ for $k = 0, 1, \ldots, M$. Then seek the solution by finding the discrete values

$$u_n^k = u\left(x_n, t_k\right)$$

From (4.10), these satisfy the equations

$$\frac{u_n^{k+1} - u_n^k}{\Delta t} = \alpha^2 \frac{u_{n+1}^k - 2u_n^k + u_{n-1}^k}{\Delta x^2}$$

or, rearranging,

$$u_n^{k+1} = u_n^k + \frac{\alpha^2 \Delta t}{\Delta x^2}\left(u_{n+1}^k - 2u_n^k + u_{n-1}^k\right) \tag{4.11}$$

If the values of u_n at time step k are known, this formula gives all the values at time step $k+1$, and it can then be iterated again and again. The initial condition gives

$$u_n^0 = f\left(x_n\right), \tag{4.12}$$

for all $n \in \{0, \ldots, N\}$, and the boundary conditions require

$$u_0^k = A, \quad u_N^k = B \tag{4.13}$$

for all $k \in \{0, \ldots, M\}$ (Equation (4.11) only has to be solved for $1 \le n \le N-1$).

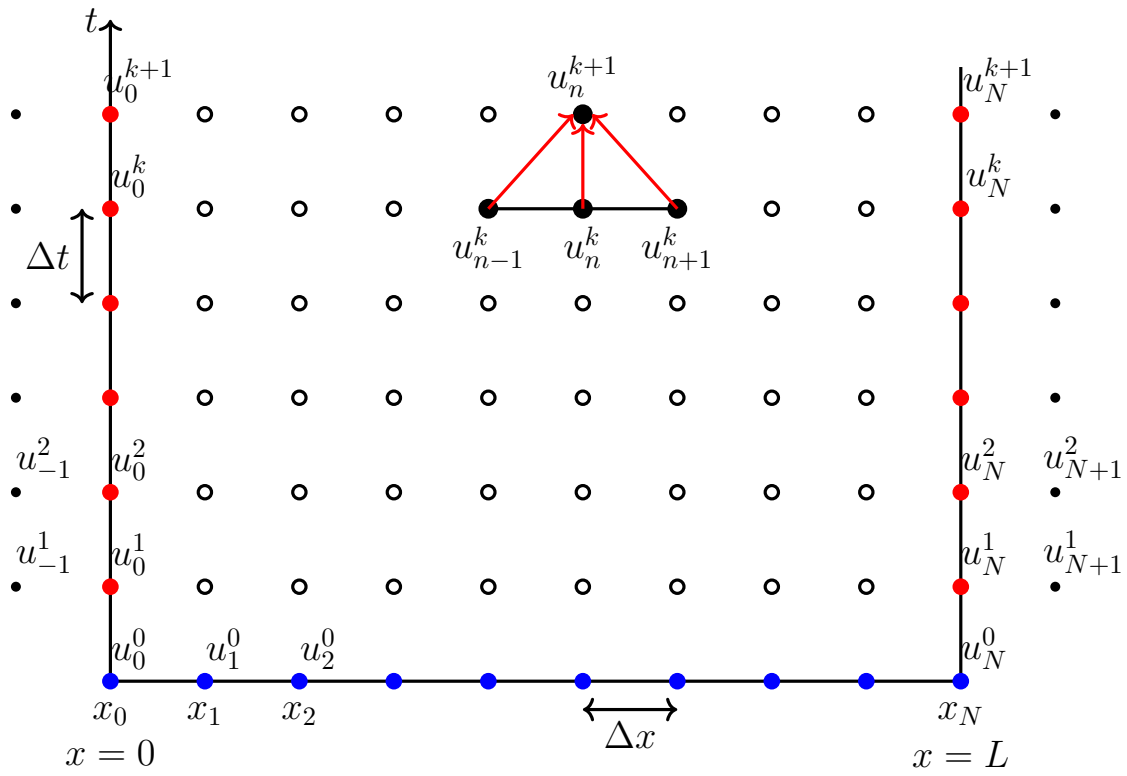

Figure 4.1: Mesh points and finite difference stencil for the heat equation. Blue points are prescribed by the initial condition, red points by the boundary conditions.

Remark 4.3.1. *In Figure 4.1, in order to use Neumann boundary conditions, fictional points (ghost nodes) at $x = -\Delta x$ and $x = L + \Delta x$ can be used to facilitate the method.*

4.3.2 Neumann boundary conditions

Methodology 4.3.2 (Neumann boundary conditions). *To apply the boundary condition*

$$\frac{\partial u}{\partial x}(0,t) = C \tag{4.14}$$

instead of $u(0,t) = A$ in (4.9b), notice that the central difference version of this boundary condition would be

$$\frac{u(0+\Delta x,t) - u(0-\Delta x,t)}{2\Delta x} = C \tag{4.15}$$

and therefore

$$u(-\Delta x,t) = u(\Delta x,t) - 2\Delta x C \quad \left(\textit{ i.e, } u_{-1}^k = u_1^k - 2\Delta x C\right) \tag{4.16}$$

$x = -\Delta x$ is outside the domain of interest, but knowing the value of $u(-\Delta x,t)$ there allows the discretised Equation (4.11) to be used also for $x = 0$ ($n = 0$), and it becomes

$$u_0^{k+1} = u_0^k + \frac{\alpha^2 \Delta t}{\Delta x^2}\left(2u_1^k - 2u_0^k - 2\Delta x C\right) \tag{4.17}$$

So, in this case we must solve (4.11) for $1 \leq n \leq N-1$, and (4.17) for $n = 0$ at each time step. If there is a derivative condition at $x = L$ the same procedure is followed and an equation similar to (4.17) must be solved for $n = N$ too. A handy way to implement this type of boundary condition, which enables the same formula (4.11) to be used for all points, is to introduce 'fictional' mesh points for $n = -1$ and $n = N+1$, and to prescribe the value u_{-1}^k given by (4.16) (or the equivalent for u_{N+1}^k) at those points.

Stability

Theorem 4.3.1 (Stability condition). *Note that, this method will only be stable, provided the condition*

$$\frac{\alpha^2 \Delta t}{\Delta x^2} \leq \frac{1}{2} \tag{4.18}$$

is satisfied; otherwise it will not work.

4.4 4.4 Solving the Wave equation using the method of finite differences

4.4.1 Dirichlet boundary conditions

Methodology 4.4.1 (Dirichlet boundary conditions). *For the wave equation,*

$$PDE: \quad \frac{\partial^2 u}{\partial t^2} = c^2 \frac{\partial^2 u}{\partial x^2}, \quad 0 < x < L, \quad t > 0 \tag{4.19a}$$

$$BC: \quad u(0,t) = 0, \quad u(L,t) = 0 \tag{4.19b}$$

$$IC: \quad u(x,0) = f(x), \quad \frac{\partial u}{\partial t}(x,0) = g(x) \tag{4.19c}$$

discretise x into $N+1$ evenly spaced mesh points $x_n = n\Delta x$, discretise the time interval $[0,T]$ into $M+1$ equal time levels $t_k = k\Delta t$ for $k = 0, 1, \ldots, M$, and seek the solution at these mesh points; $u_n^k = u(x_n, t_k)$. Using central difference approximations to the two second derivatives, the discrete equation is

$$\frac{u_n^{k+1} - 2u_n^k + u_n^{k-1}}{\Delta t^2} = c^2 \frac{u_{n+1}^k - 2u_n^k + n_{n-1}^k}{\Delta x^2} + O\left(\Delta x^2, \Delta t^2\right). \tag{4.20}$$

This can be rearranged to give

$$u_n^{k+1} = 2u_n^k - u_n^{k-1} + \frac{c^2 \Delta t^2}{\Delta x^2}\left(u_{n+1}^k - 2u_n^k + n_{n-1}^k\right) \tag{4.21}$$

which gives an explicit method to calculate u_n^{k+1} in terms of the values at the previous two time steps k and $k-1$. This method is second order accurate in space and time - it is sometimes referred to as the 'leap-frog' method.
To apply Dirichlet boundary conditions given in (4.19b), the values of u_0^{k+1} and u_N^{k+1} are simply prescribed to be 0 ; there is no need to solve an equation for these end points.

4.4.2 Neumann Boundary conditions

Methodology 4.4.2 (Neumann boundary conditions). *If the boundary conditions are Neumann, on the other hand:*

$$\frac{\partial u}{\partial x}(0,t) = 0, \quad \frac{\partial u}{\partial x}(L,t) = 0 \tag{4.22}$$

then (4.21) can still be solved for $n = 0$ and $n = N$ if we use the central difference approximations to the boundary conditions in order to determine the 'fictional' values u_{-1}^k and u_{N+1}^k respectively. At $x = 0$, for instance, the discretised condition (4.22) is

$$\frac{u_{-1}^k - u_1^k}{2\Delta x} = 0 \tag{4.23}$$

and therefore $u_{-1}^k = u_1^k$. Similarly $u_{N+1}^k = u_{N-1}^k$.

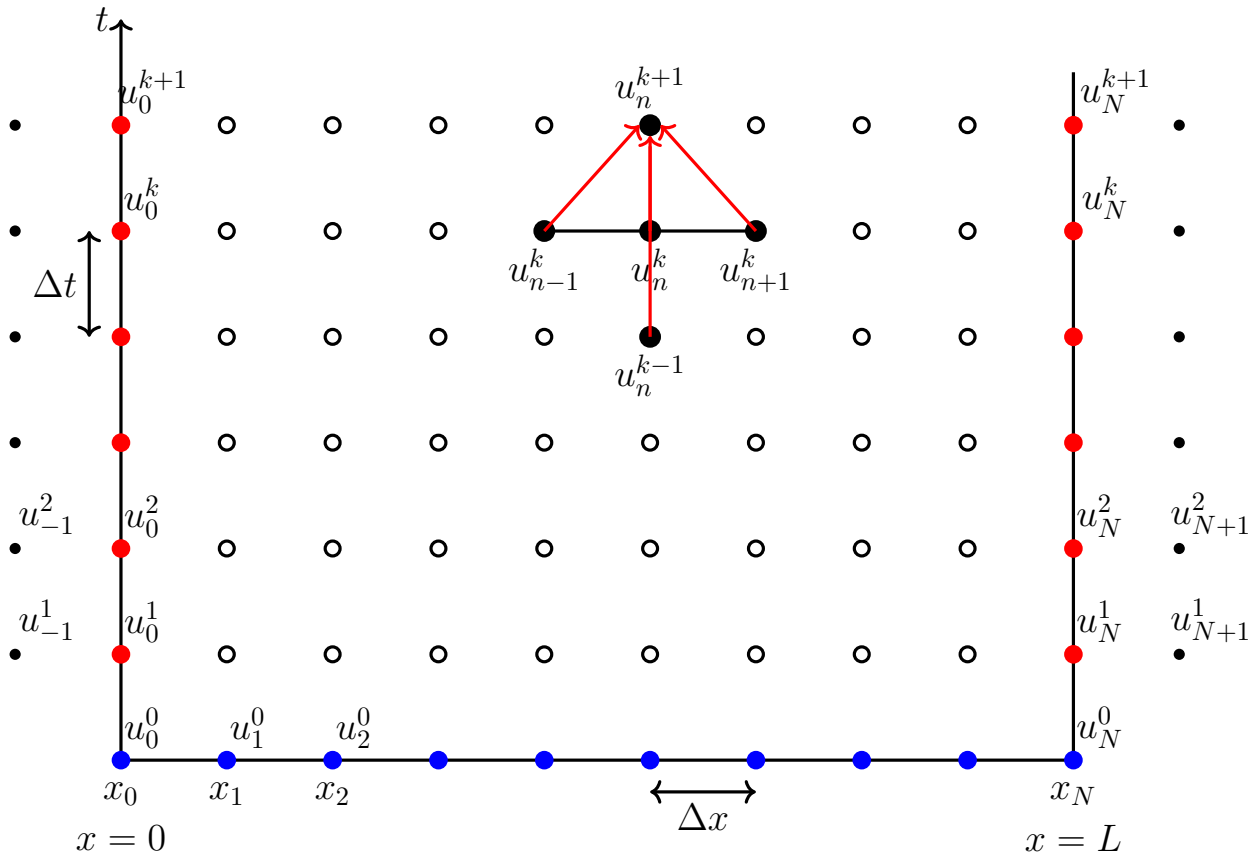

Figure 4.2: Mesh points and finite difference stencil for the wave equation.

4.4.3 Initial conditions

Methodology 4.4.3 (nitial conditions). *To apply the initial conditions, note that to use (4.21) for the first time step $k = 1$, we need to know the values of u_n^0 and also u_n^{-1}. The values of u_n^0 follow from the initial condition on $u(x, 0)$;*

$$u_n^0 = f(x_n) \tag{4.24}$$

The values of u_n^{-1} come from considering the central difference approximation to the derivative condition given in (4.19c);

$$\frac{u(x, 0 + \Delta t) - u(x, 0 - \Delta t)}{2\Delta t} = g(x),$$

from which we deduce that

$$u_n^{-1} = u_n^1 - 2\Delta t g(x_n)$$

Substituting this into the discrete equation (4.21), and rearranging, gives the formula

$$u_n^1 = u_n^0 + \frac{1}{2}\frac{c^2\Delta t^2}{\Delta x^2}\left(u_{n+1}^0 - 2u_n^0 + n_{n-1}^0\right) + \Delta t g(x_n) \tag{4.25}$$

for the first time step. Once the solution has been initialised in this way, all subsequent time steps can be made using (4.21).

Stability

Theorem 4.4.1 (**Stability condition).** *This method is stable provided*

$$\frac{c\Delta t}{\Delta x} \leq 1 \tag{4.26}$$

which is often called the Courant-Friedrichs-Levy (or CFL) condition.

4.5 Solving the Laplace's equation using the method of finite differences

4.5.1 Dirichlet boundary conditions

Methodology 4.5.1 (**Dirichlet boundary conditions).** *Laplace's equation on a square domain is*

$$\frac{\partial^2 u}{\partial x^2} + \frac{\partial^2 u}{\partial y^2} = 0, \quad 0 < x < L, \quad 0 < y < L \tag{4.27}$$

with boundary conditions,

$$u(0, y) = 0, \quad u(L, y) = 0, \quad u(x, 0) = f(x), \quad u(x, L) = 0 \tag{4.28}$$

Choose a mesh with spacing Δx in the x direction, Δy in the y direction (often it is sensible to choose $\Delta x = \Delta y$), so grid points are $x_n = n\Delta x$ for $n = 0, 1, \ldots N$, and $y_m = m\Delta y$ for $m = 0, 1, \ldots M$. Then seek solution values $u_{nm} = u(x_n, y_m)$. Using second order accurate central differences for the two derivatives, the discrete equation is

$$\frac{u_{n+1m} - 2u_{nm} + u_{n-1m}}{\Delta x^2} + \frac{u_{nm+1} - 2u_{nm} + u_{nm-1}}{\Delta y^2} = O\left(\Delta x^2, \Delta y^2\right) \tag{4.29}$$

and if $\Delta x = \Delta y$ this simplifies to

$$u_{nm} = \frac{1}{4}\left(u_{n+1m} + u_{n-1m} + u_{nm+1} + u_{nm-1}\right) \tag{4.30}$$

Note this equation shows that the solution has the property that the value at each point (x_n, y_m) is the average of the values at its four neighbouring points. This is an important property of Laplace's equation.
The boundary conditions (4.28) determine the values u_{nm} on each of the four boundaries, so (4.30) does not have to be solved at these mesh points:

$$u_{0m} = 0, \quad u_{Nm} = 0, \quad u_{n0} = f(x_n), \quad u_{nM} = 0 \tag{4.31}$$

Neumann boundary conditions

Neumann boundary conditions can be incorporated by calculating values for u_{-1m} (for instance), as for the heat equation.

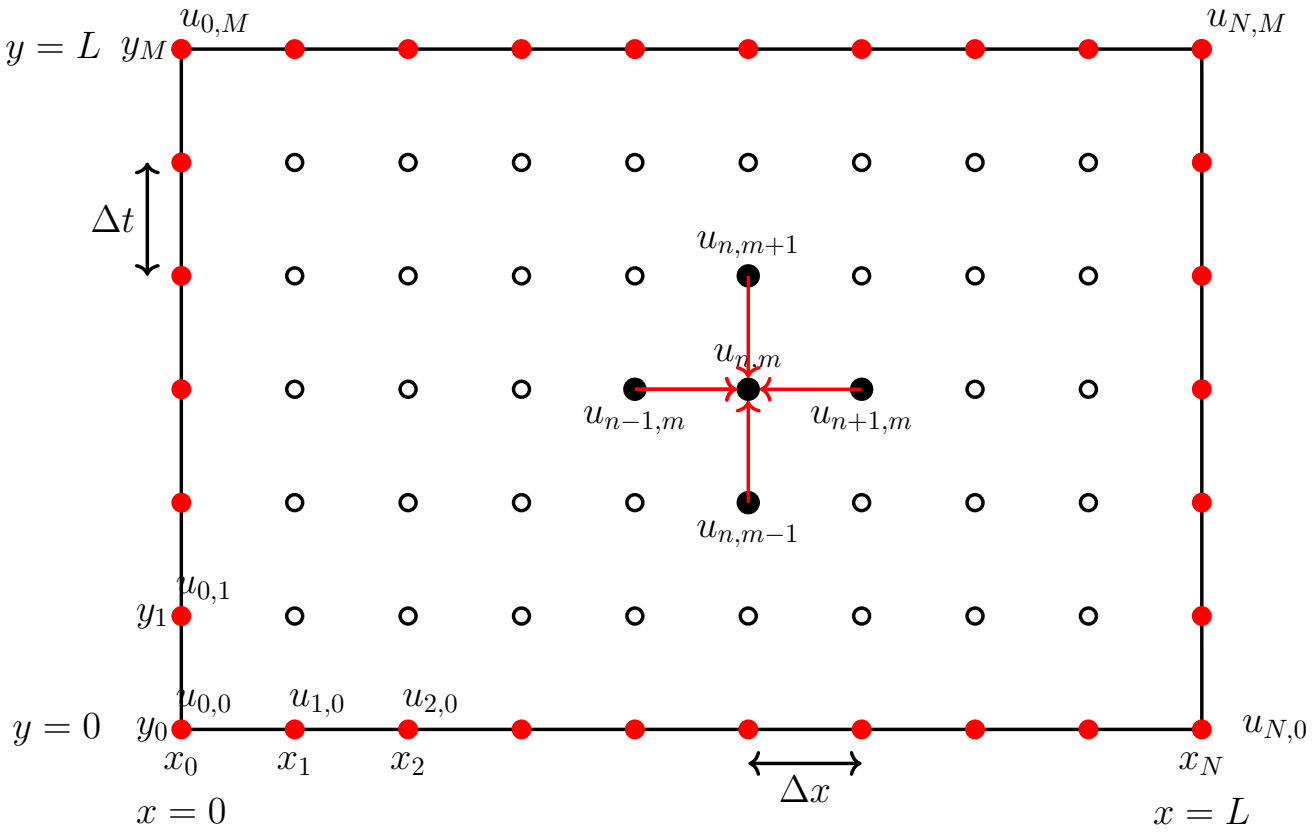

Figure 4.3: Mesh points and finite difference stencil for Laplace's equation

Jacobi Iteration

Note that (4.30) is not an explicit formula, since the solution at each point depends on the unknown values at other points, and it is therefore harder to solve than the previous explicit methods. One way to do it, however, is to take a guess at the solution (eg. $u_{nm} = 0$ everywhere), and then go through each of the mesh points updating the solution according to (4.30) by taking the values of the previous guess on the right hand side. Iterating this process many times, the successive approximations will hopefully change by less and less, and the values they converge to provide the solution. Given an initial guess $u_{nm}^{(0)}$, the successive iterations $u_{nm}^{(1)}, \ldots, u_{nm}^{(k)}, u_{nm}^{(k+1)}, \ldots$, are given by

$$u_{nm}^{(k+1)} = \frac{1}{4}\left(u_{n+1m}^{(k)} + u_{n-1m}^{(k)} + u_{nm+1}^{(k)} + u_{nm-1}^{(k)}\right)$$

Here the superscript (k) denotes the values for the k$^{\text{th}}$ iteration. The process is continued until the change between successive iterations is less than some desired tolerance.

IMPLEMENTATION!

We will look at some matlab implementations of the above schemes.