Math 257: Finite difference methods

1 Finite Differences

Remember the definition of a derivative

$$f'(x) = \lim_{\Delta x \to 0} \frac{f(x+\Delta x) - f(x)}{\Delta x} \tag{1}$$

Also recall Taylor's formula:

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

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

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

Forward difference

On a computer, derivatives are approximated by *finite difference* expressions; rearranging (2) gives the *forward difference* approximation

$$\frac{f(x+\Delta x) - f(x)}{\Delta x} = f'(x) + O(\Delta x), \tag{4}$$

where $O(\Delta x)$ means 'terms of order Δx', *ie.* terms which have size similar to or smaller than Δx when Δx is small.[1] 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'.

[1]The strict mathematical definition of $O(\delta)$ is to say that it represents terms y such that

$$\lim_{\delta \to 0} \left(\frac{y}{\delta} \right) \quad \text{is finite.}$$

So δ^2 and δ are $O(\delta)$, but $\delta^{1/2}$ or 1, for instance, are not. Another commonly used notation is $o(\delta)$, which represents terms y for which

$$\lim_{\delta \to 0} \left(\frac{y}{\delta} \right) = 0.$$

In other words, $o(\delta)$ represents terms that are *strictly* smaller than δ as $\delta \to 0$. So δ^2 is $o(\delta)$, but δ is not.

Backward difference

Rearranging (3) similarly gives the *backward difference* approximation

$$\frac{f(x) - f(x - \Delta x)}{\Delta x} = f'(x) + O(\Delta x), \tag{5}$$

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

Centered difference

Combining (2) and (3) gives the *centered difference* approximation

$$\frac{f(x + \Delta x) - f(x - \Delta x)}{2\Delta x} = f'(x) + O(\Delta x^2), \tag{6}$$

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

Second derivative, centered difference

Adding (2) and (3) gives

$$f(x + \Delta x) + f(x - \Delta x) = 2f(x) + \Delta x^2 f''(x) + \frac{\Delta x^4}{12} f^{(4)}(x) + \dots \tag{7}$$

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

$$\frac{f(x + \Delta x) - 2f(x) + f(x - \Delta x)}{\Delta x^2} = f''(x) + O(\Delta x^2), \tag{8}$$

which is second order accurate.

2 Heat Equation

Dirichlet boundary conditions

To find a numerical solution to the heat equation

$$\frac{\partial u}{\partial t} = \alpha^2 \frac{\partial^2 u}{\partial x^2}, \quad 0 < x < L, \tag{9}$$

$$u(0, t) = A, \quad u(L, t) = B, \quad u(x, 0) = f(x), \tag{10}$$

approximate the time derivative using forward differences, and the spatial derivative using centered differences;

$$\frac{u(x, t + \Delta t) - u(x, t)}{\Delta t} = \alpha^2 \frac{u(x + \Delta x, t) - 2u(x, t) + u(x - \Delta x, t)}{\Delta x^2} + O(\Delta t, \Delta x^2). \tag{11}$$

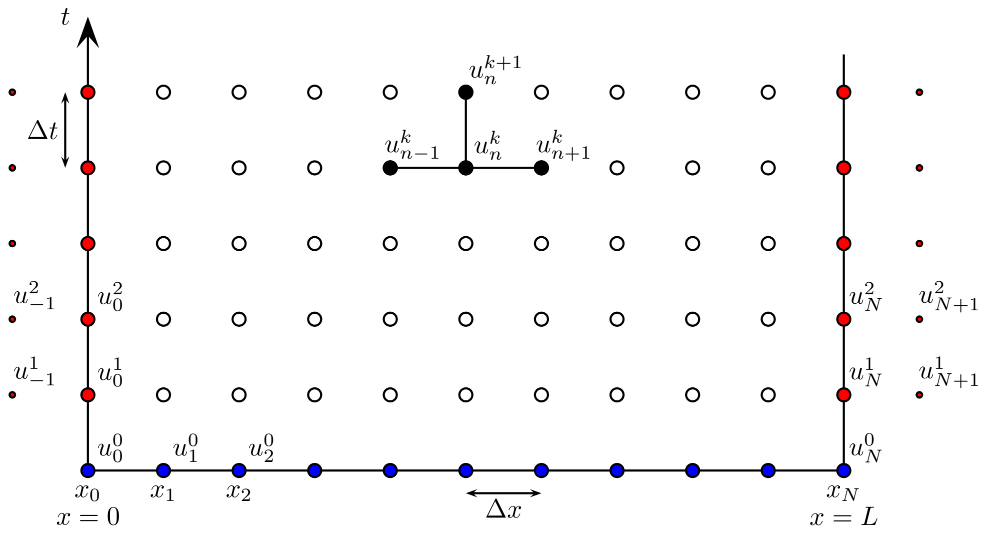

Figure 1: Mesh points and finite difference stencil for the heat equation. Blue points are prescribed the initial condition, red points are prescribed by the boundary conditions. For Neumann boundary conditions, fictional points at $x = -\Delta x$ and $x = L + \Delta x$ can be used to facilitate the method.

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 time into discrete levels $t_k = k\Delta t$ for $k = 0, 1, \ldots$. Then seek the solution by finding the discrete values

$$u_n^k = u(x_n, t_k). \tag{12}$$

From (11), these satisfy the equations

$$\frac{u_n^{k+1} - u_n^k}{\Delta t} = \alpha^2 \frac{u_{n+1}^k - 2u_n^k + u_{n-1}^k}{\Delta x^2}, \tag{13}$$

or, rearranging,

$$u_n^{k+1} = u_n^k + \frac{\alpha^2 \Delta t}{\Delta x^2}\left(u_{n+1}^k - 2u_n^k + u_{n-1}^k\right). \tag{14}$$

If the values of u_n at timestep k are known, this formula gives all the values at timestep $k + 1$, and it can then be iterated again and again to march forward in time.

The initial condition gives

$$u_n^0 = f(x_n), \tag{15}$$

for all n, and the boundary conditions require

$$u_0^k = A, \quad u_N^k = B, \tag{16}$$

for all values of $k > 0$ (equation (14) only has to be solved for $1 \leq n \leq N-1$).

Stability

Note, this method will only be *stable*, provided the condition

$$\frac{\alpha^2 \Delta t}{\Delta x^2} \leq \frac{1}{2}, \tag{17}$$

is satisfied; otherwise it will not work.

Neumann boundary conditions

To apply

$$\frac{\partial u}{\partial x}(0,t) = C, \tag{18}$$

instead of (10), notice that the centered difference version of this boundary condition would be

$$\frac{u(0+\Delta x, t) - u(0-\Delta x, t)}{2\Delta x} = C, \tag{19}$$

and therefore

$$u(-\Delta x, t) = u(\Delta x, t) - 2\Delta x C. \tag{20}$$

$x = -\Delta x$ is outside the domain of interest, but knowing the value of $u(-\Delta x, t)$ there allows the discretised equation (14) to be used also for $x = 0$ ($n = 0$), and it becomes

$$u_0^{k+1} = u_0^k + \frac{\alpha^2 \Delta t}{\Delta x^2}\left(2u_1^k - 2u_0^k - 2\Delta x C\right). \tag{21}$$

So in this case we must solve (14) for $1 \leq n \leq N-1$, and (21) for $n = 0$ at each timestep. If there is a derivative condition at $x = L$ the same procedure is followed and an equation similar to (21) must be solved for $n = N$ too.

A handy way to implement this type of boundary condition, which enables the same formula (14) 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 (20) (or the equivalent for u_{N+1}^k) at those points.

3 Wave equation

For the wave equation,

$$\frac{\partial^2 u}{\partial t^2} = c^2 \frac{\partial^2 u}{\partial x^2}, \quad 0 < x < L, \tag{22}$$

$$u(0,t) = 0, \quad u(L,t) = 0, \tag{23}$$

$$u(x,0) = f(x), \quad \frac{\partial u}{\partial t}(x,0) = g(x), \tag{24}$$

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

$$\frac{u_n^{k+1} - 2u_n^k + u_n^{k-1}}{\Delta t^2} = c^2 \frac{u_{n+1}^k - 2u_n^k + n_{n-1}^k}{\Delta x^2} + O(\Delta x^2, \Delta t^2). \tag{25}$$

This can be rearranged to give

$$u_n^{k+1} = 2u_n^k - u_n^{k-1} + \frac{c^2 \Delta t^2}{\Delta x^2} \left(u_{n+1}^k - 2u_n^k + n_{n-1}^k \right), \tag{26}$$

which gives an explicit method to calculate u_n^{k+1} in terms of the values at the previous two timesteps k and $k-1$. This method is second order accurate in space and time - it is sometimes referred to as the ‘leap-frog’ method.

Boundary conditions

To apply Dirichlet boundary conditions (23), 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.

If the boundary conditions are Neumann, on the other hand:

$$\frac{\partial u}{\partial x}(0,t) = 0, \quad \frac{\partial u}{\partial x}(L,t) = 0, \tag{27}$$

then (26) can still be solved for $n = 0$ and $n = N$ if we use the centered 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 (27) is

$$\frac{u_{-1}^k - u_1^k}{2\Delta x} = 0, \tag{28}$$

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

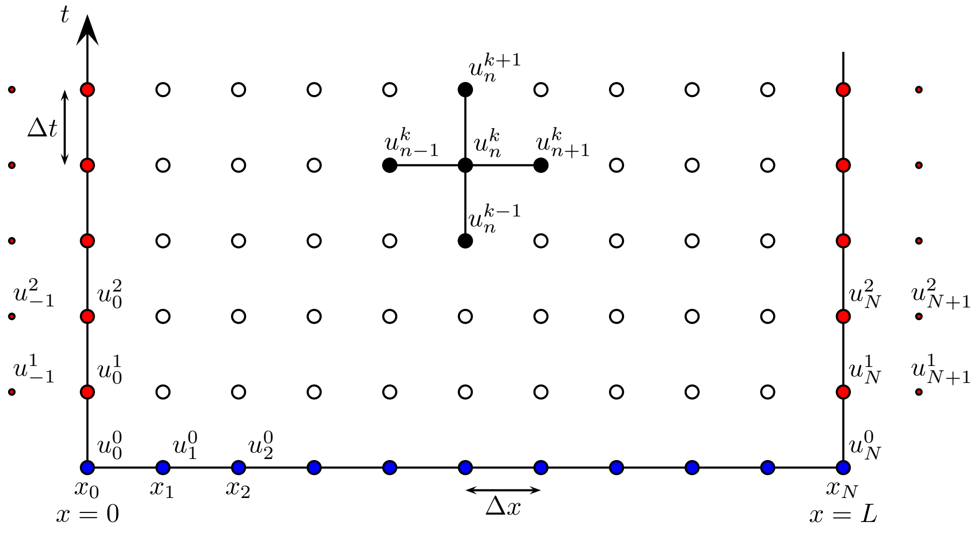

Figure 2: Mesh points and finite difference stencil for the wave equation.

Initial conditions

To apply the initial conditions, note that to use (26) for the first timestep $k = 1$, 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 $u(x, 0)$;

$$u_n^0 = f(x_n). \tag{29}$$

The values of u_n^{-1} come from considering the centered difference approximation to the derivative condition (24);

$$\frac{u(x, 0 + \Delta t) - u(x, 0 - \Delta t)}{2\Delta t} = g(x), \tag{30}$$

from which

$$u_n^{-1} = u_n^1 - 2\Delta t g(x_n). \tag{31}$$

Substituting this into the discrete equation (26), and rearranging, gives the formula

$$u_n^1 = u_n^0 + \frac{1}{2}\frac{c^2\Delta t^2}{\Delta x^2}\left(u_{n+1}^0 - 2u_n^0 + n_{n-1}^0\right) + \Delta t g(x_n), \tag{32}$$

for the first timestep. Once the solution has been initialised in this way, all subsequent timesteps can be made using (26).

Stability

This method is stable provided

$$\frac{c\Delta t}{\Delta x} \leq 1, \tag{33}$$

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

4 Laplace's equation

Laplace's equation on a square domain is

$$\frac{\partial^2 u}{\partial x^2} + \frac{\partial^2 u}{\partial y^2} = 0, \quad 0 < x < L, \quad 0 < y < L, \tag{34}$$

with boundary conditions, for instance,

$$u(0, y) = 0, \quad u(L, y) = 0, \quad u(x, 0) = f(x), \quad u(x, L) = 0. \tag{35}$$

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 $n = 0, 1, \dots N$, and $y_m = m\Delta y$ for $m = 0, 1, \dots M$. Then seek solution values $u_{n\,m} = u(x_n, y_m)$. Using second order accurate centered differences for the two derivatives, the discrete equation is

$$\frac{u_{n+1\,m} - 2u_{n\,m} + u_{n-1\,m}}{\Delta x^2} + \frac{u_{n\,m+1} - 2u_{n\,m} + u_{n\,m-1}}{\Delta y^2} = O(\Delta x^2, \Delta y^2), \tag{36}$$

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

$$u_{n\,m} = \frac{1}{4}\left(u_{n+1\,m} + u_{n-1\,m} + u_{n\,m+1} + u_{n\,m-1}\right) \tag{37}$$

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.

Boundary conditions

The boundary conditions (35) determine the values $u_{n\,m}$ on each of the four boundaries, so (37) does not have to be solved at these mesh points:

$$u_{0\,m} = 0, \quad u_{N\,m} = 0, \quad u_{n\,0} = f(x_n), \quad u_{n\,M} = 0. \tag{38}$$

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

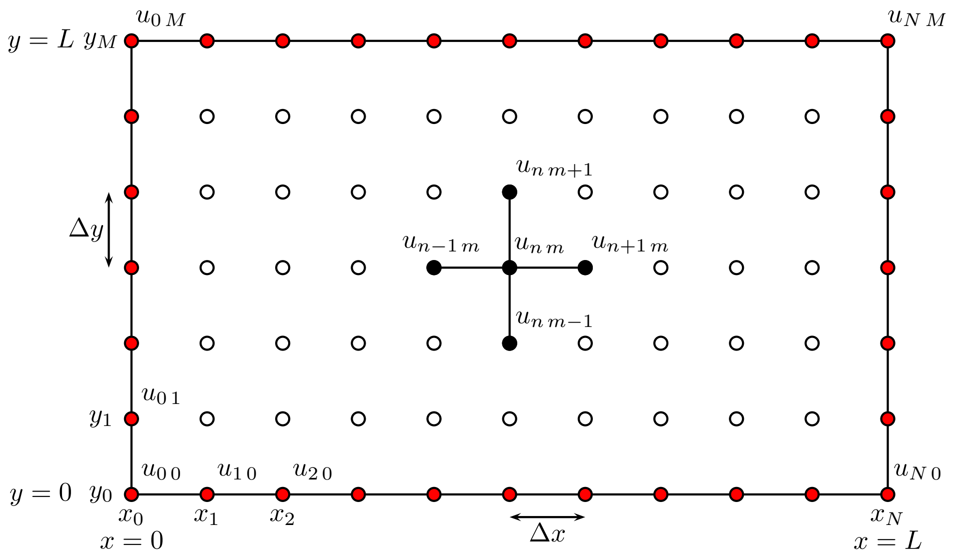

Figure 3: Mesh points and finite difference stencil for Laplace's equation.

Jacobi Iteration

Note (37) 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.* $u_{n\,m} = 0$ everywhere), and then go through each of the mesh points updating the solution according to (37) 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 $u^{(0)}_{n\,m}$, the successive iterations $u^{(1)}_{n\,m}, \ldots,\ u^{(k)}_{n\,m},\ u^{(k+1)}_{n\,m}, \ldots$, are given by

$$u^{(k+1)}_{n\,m} = \frac{1}{4}\left(u^{(k)}_{n+1\,m} + u^{(k)}_{n-1\,m} + u^{(k)}_{n\,m+1} + u^{(k)}_{n\,m-1}\right). \tag{39}$$

Here the superscript (k) denotes the values for the kth iteration. The process is continued until the change between successive iterations is less than some desired tolerance.