CMIS Hand-in 1: Finite Difference Methods 1

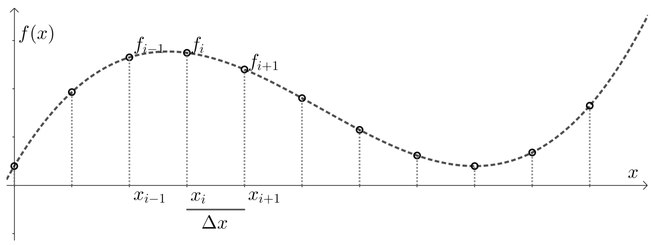
Nikolai Plambech Nielsen
lpk331@alumni.ku.dk
Niels Bohr Institute, University of Copenhagen

1 THE FINITE DIFFERENCE METHOD

When solving differential equations on a computer, one has several options. In this hand-in the finite difference method (FDM) is explored.

In general we are interested in calculating the evolution of a system with time over some domain, as it is governed by a partial differential equation. The domain is the "space" over which we solve the system, for example a line, surface or volume. We then sample this domain at a finite set of points, called the nodes of the system. All nodes then need to be updated for the system to evolve in time. The computational mesh is then the collection of all nodes in our domain.

For this hand in we are dealing with a regular mesh. This means that we sample the system at regular intervals in space, with a spacing of Δx:



With the x-values given by:

$$x_i = x_0 + i\Delta x \tag{1}$$

where x_0 is usually taken as the origin, for convenience. The sampled values of the system $f(x_i)$ are then denoted in shorthand by

$$f_i \equiv f(x_i) \tag{2}$$

This of course generalises to higher dimensional systems. For two dimensions we have $y_j = y_0 + j\Delta y$, and $f_{i,j} \equiv f(x_i, y_j)$, and more indices for even higher dimensions.

2 L1A

The idea behind the finite difference method is to replace all derivatives in the equations with differences instead. This can be seen as not letting the spacing h in the differential quotient go all the way to zero, but just to some small value. Since we are working with regular grids, a convenient value is the spacing between points Δx:

$$\frac{df(x_i)}{dx} = \lim_{h\to 0} \frac{f(x_i+h) - f(x_i)}{(x_i+h) - x_i} \to \frac{f(x_i+\Delta x) - f(x_i)}{\Delta x} \tag{3}$$

In this particular case, the finite difference is called the "forward difference" (FD) since it uses the point of interest x_i and the one next to it $x_{i+1} = x_i + \Delta x$. In the compact notation we have

$$\frac{df_i}{dx} \approx \frac{f_{i+1} - f_i}{\Delta x} \tag{4}$$

There is also the "backwards difference" (BD):

$$\frac{df_i}{dx} \approx \frac{f_i - f_{i-1}}{\Delta x} \tag{5}$$

And the "central difference" (CD):

$$\frac{df_i}{dx} \approx \frac{f_{i+1} - f_{i-1}}{2\Delta x} \tag{6}$$

These can be shown explicitly from the Taylor Polynomials for $f(x)$ around x_i. There are of course also higher order differences. In particular we use the second order central difference when studying the heat equation, as this includes a second derivative:

$$\frac{d^2 f_i}{dx^2} \approx \frac{f_{i-1} - 2f_i + f_{i+1}}{\Delta x^2} \tag{7}$$

This, however, only takes care of calculating derivatives. What we want is to follow the evolution of the system with time. This means we have to do some sort of numerical time integration as well. But that is the subject of next weeks hand-in.

There are some considerations we need to make when designing the simulation of the system. In general we have some restrictions in the form of time, memory and accuracy.

These restrictions boil down to choosing the parameters of the system (for example Δx and Δy so as to get an acceptable trade-off between the restrictions).

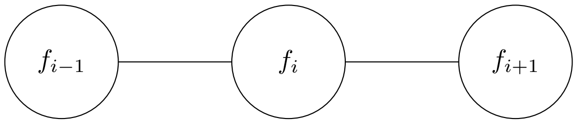
The first concern is accuracy. We set out to solve some partial differential equation. This is all for naught, if we do not calculate an accurate result. The easiest way of increasing the accuracy is to decrease the spacing between the nodes. This of course results in more accurate approximations of the derivatives (with analytical results as $\Delta x \to 0$), but also increases both computation time and memory cost (if we keep the domain constant).

One might then think, given infinite time and memory - could we just keep decreasing the grid spacing Δx to get a more accurate result? It turns out we cannot, as the computer cannot resolve the difference between values accurately, as they approach each other.

To be more specific, the computer stores the numbers as floating points numbers, which in Numpy has a spacing of $2.2204 \cdot 10^{-16}$ (called the machine epsilon) for double precision floats (accessed by `numpy.finfo(float).eps`). Two numbers which differ by less than this value will be identical to the computer, and their difference will thus be 0. Trying to then divide by this difference (for example) will lead to crashing of the simulation.

3 CONSTRUCTING THE SIMULATION

When constructing the simulator it is useful to get a high-level overview of which values are needed to update each node. This is where the stencil comes in. This is is a schematic representation of which nodes are involved in each calculation. If, for example we want to calculate the second order derivative of a function at the point x_i, using the second order centred difference, we will need the nodes x_i, x_{i+i} and x_{i-1}. This can be graphically shown as



4 L1C