Derivation of Relaxation Algorithm

January 2020

1 Discretising a function

Given the Laplace equation $\nabla^2\phi = 0$, how do we discretise the equation for use in a numerical scheme?

Consider the Taylor expansion of a function $f(x)$:

$$f(x) = f(x_0) + (x - x_0)f'(x_0) + \frac{(x-x_0)^2}{2!}f''(x_0) + ...$$

To discretise this equation we set $f(x) = f_{i+1}$, $f(x_0) = f_i$ and $h = x - x_0$ giving:

$$f_{i+1} = f_i + hf'_i + \frac{h^2}{2!}f''_i + ...$$

$$hf'_i = f_{i+1} - f_i - \frac{h^2}{2!}f''_i - ...$$

$$f'_i = \frac{f_{i+1}-f_i}{h} - \frac{h^2}{2!}f''_i - ...$$

Now we discard the non-dominant terms and represent the error in our approximation using big O notation. $O(h)$ is used to represent some error term that is directly proportional to h i.e. it tells us how big an improvement in accuracy (i.e. how much smaller the error term can get) when changing h. The higher the power of h our error depends on, the greater the effect reducing h has on the error. Here, h specifically refers to our mesh spacing.

$$f'_i = \frac{f_{i+1}-f_i}{h} + O(h)$$

This expression is called the **two point formula** and is the most basic discretisation method for the derivative of a function. However we can easily expand $f_i + 1$ and $f_i - 1$ to give $O(h^2)$:

$$f_{i+1} = f_i + hf_i' + \frac{h^2}{2!}f_i'' - \frac{h^3}{3!}f_i^{(3)} + \ldots, \quad f_{i-1} = f_i - hf_i' + \frac{h^2}{2!}f_i'' + \frac{h^3}{3!}f_i^{(3)} + \ldots$$

$$f_{i+1} - f_{i-1} = 2hf_i' - \frac{2h^3}{3!}f_i^{(3)} + \ldots$$

$$f_i' = \frac{f_{i+1} - f_{i-1}}{h} + \frac{2h^2}{3!}f_i^{(3)} + \ldots$$

$$f_i' = \frac{f_{i+1} - f_{i-1}}{h} + O(h^2)$$

However if we want to find the second derivative using a three point formula we need to cancel out the first and third derivatives:

$$f_{i+1} + f_{i-1} = 2f_i + \frac{2h^2}{2!}f_i'' + O(h^4)$$

$$h^2 f_i'' = f_{i+1} + f_{i-1} - 2f_i + O(h^4)$$

$$f_i'' = \frac{f_{i+1} + f_{i-1} - 2f_i}{h^2} + O(h^2)$$

2 Generalising to multivariate functions

This can then easily be generalised to partial derivatives for a multivariate function $f(x_1, x_2)$:

$$\frac{\partial^2 f_{i,j}}{\partial x_1^2} = \frac{f_{i+1,j} + f_{i-1,j} - 2f_{i,j}}{h_{x1}^2} + O(h_{x1}^2)$$

$$\nabla^2 f_{i,j} = \frac{f_{i+1,j} + f_{i-1,j} - 2f_{i,j}}{h_{x1}^2} + \frac{f_{i,j+1} + f_{i,j-1} - 2f_{i,j}}{h_{x2}^2} + O(h_{x1}^2 + h_{x2}^2)$$

We can now go back to the original problem, substitute for $\nabla^2\phi = 0$, and rearrange to give the discretised $\phi_{i,j}$ in terms of its neighbouring points:

$$\frac{\phi_{i+1,j} + \phi_{i-1,j} - 2\phi_{i,j}}{h_x^2} + \frac{\phi_{i,j+1} + \phi_{i,j-1} - 2\phi_{i,j}}{h_y^2} + O(h_x^2 + h_y^2) = 0$$

$$\frac{2(h_x^2 + h_y^2)\phi_{i,j}}{h_x^2 h_y^2} = \frac{h_y^2(\phi_{i+1,j} + \phi_{i-1,j}) + h_x^2(\phi_{i,j+1} + \phi_{i,j-1})}{h_x^2 h_y^2} + O(h_x^2 + h_y^2)$$

$$2(h_x^2 + h_y^2)\phi_{i,j} = h_y^2(\phi_{i+1,j} + \phi_{i-1,j}) + h_x^2(\phi_{i,j+1} + \phi_{i,j-1}) + O(h_x^2 h_y^2(h_x^2 + h_y^2))$$

let $\alpha = \frac{h_x^2}{h_y^2}$

$$2(\alpha + 1)\phi_{i,j} = \phi_{i+1,j} + \phi_{i-1,j} + \alpha(\phi_{i,j+1} + \phi_{i,j-1}) + O(h_x^2(h_x^2 + h_y^2))$$

$$\phi_{i,j} = \frac{1}{2(\alpha+1)}[\phi_{i+1,j} + \phi_{i-1,j} + \alpha(\phi_{i,j+1} + \phi_{i,j-1})] + O(h_x^2 \frac{(h_x^2 + h_y^2)}{(1 + \frac{h_x^2}{h_y^2})})$$

$$\phi_{i,j} = \frac{1}{2(\alpha+1)}[\phi_{i+1,j} + \phi_{i-1,j} + \alpha(\phi_{i,j+1} + \phi_{i,j-1})] + O(h_x^2 h_y^2) \tag{1.0}$$

3 The Relaxation Algorithm

With this method for finding a discrete point's potential in terms of its surrounding point's potential we can now implement a scheme for finding the potentials at each point given some boundary condition. This scheme, **relaxation**, iteratively moves the potential at each non-boundary point towards the correct value using this recurrence relation for the kth iteration:

$$\phi_{i,j}^{(k+1)} = \phi_{i,j}^{(k)} - \Delta\phi_{i,j} = \phi_{i,j}^{(k)} - p(\phi_{i,j}^{(k)} - \phi_{i,j}), \;\; p > 0$$

The requirement for a positive p arises because we have to take away difference between the current estimate and the prediction from Eq.(1.0) in order to move the potential closer to the correct value. As we don't allow the boundary values to change, these are the values that determine the value of the equation at all points on the grid - their effect on neighbouring points being propagated to all other points in the grid given enough iterations. In the code we use the somewhat simplified expression for updating:

$$\phi_{i,j}^{(k+1)} = (1-p)\phi_{i,j}^{(k)} + p\phi_{i,j}, \;\; p \in [0,2]$$

As p is is the step size for changes we make to the potential the optimal value will depend on the geometry of potential surface. Rather than determine the optimal p analytically, in practice it turns out to be more efficient to choose a p by quickly testing the convergence rates of multiple different p's and choosing the best one. We do this by running 10 iterations on values in [0,2] at 0.1 intervals and choose the value with the minimum error.