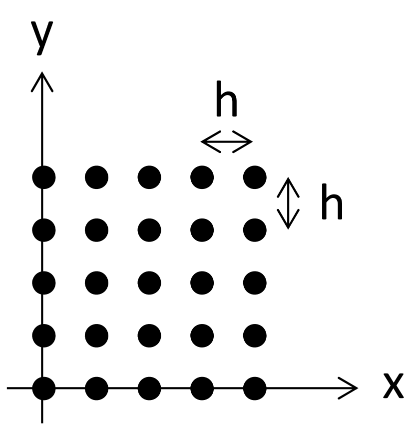
Numerical Solution to Laplace Equation: Finite Difference Method

[Note: We will illustrate this in 2D. Extension to 3D is straightforward.]

Suppose seek a solution to the Laplace Equation subject to Dirichlet boundary conditions :

$$\vec{\nabla}^2\Phi(x,y) = \frac{\partial^2\Phi(x,y)}{\partial x} + \frac{\partial^2\Phi(x,y)}{\partial y} = 0 \qquad \textit{subject to } \Phi \textit{ specified on the boundary}$$

We discretize the (x,y) region of interest into a grid, with equal Δx = Δy = h grid spacings.



As we showed on HW #1, centered difference approximations to the partial derivatives at a grid point (i,j) are :

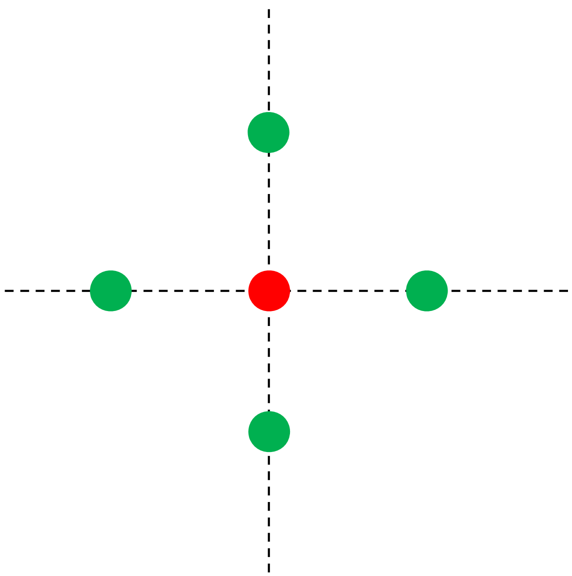
$$\left.\frac{\partial^2\Phi(x,y)}{\partial x^2}\right|_{(i,j)} \approx \frac{\Phi_{i+1,j} - 2\Phi_{i,j} + \Phi_{i-1,j}}{h^2} \qquad \left.\frac{\partial^2\Phi(x,y)}{\partial y^2}\right|_{(i,j)} \approx \frac{\Phi_{i,j+1} - 2\Phi_{i,j} + \Phi_{i,j-1}}{h^2}$$

Thus, the discretized approximation to the Laplace Equation becomes :

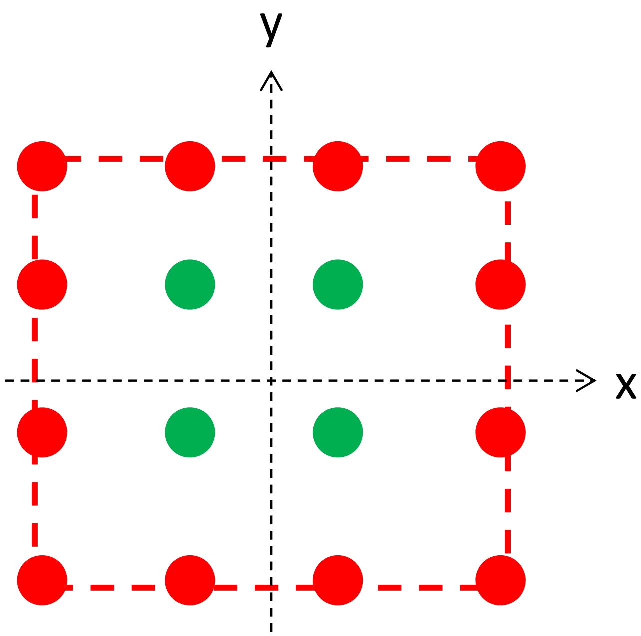
$$\left(\frac{\Phi_{i+1,j} - 2\Phi_{i,j} + \Phi_{i-1,j}}{h^2}\right) + \left(\frac{\Phi_{i,j+1} - 2\Phi_{i,j} + \Phi_{i,j-1}}{h^2}\right) \approx 0$$

$$\Rightarrow \Phi_{i,j} \approx \frac{1}{4}\left[\Phi_{i+1,j} + \Phi_{i-1,j} + \Phi_{i,j+1} + \Phi_{i,j-1}\right]$$

Thus, we see that under this approximation the value of Φ at grid point (i,j) depends on the values of Φ at its four nearest neighboring grid points.



Numerical Analysis: "five-point stencil"



"Boundary value problem" becomes :

[1] Φ "boundary values" specified on grid points on the boundary

[2] Want to be able to numerically calculate the values of Φ at the interior grid points.

Because Φ at grid point (i,j) depends on its four neighbors, we can iteratively solve for Φ at the interior grid points via the following iterative scheme ("relaxation") :

[0a] Fix the initial values of Φ on the grid boundaries subject to the boundary values.

[0b] Set/choose initial values for the interior grid points.

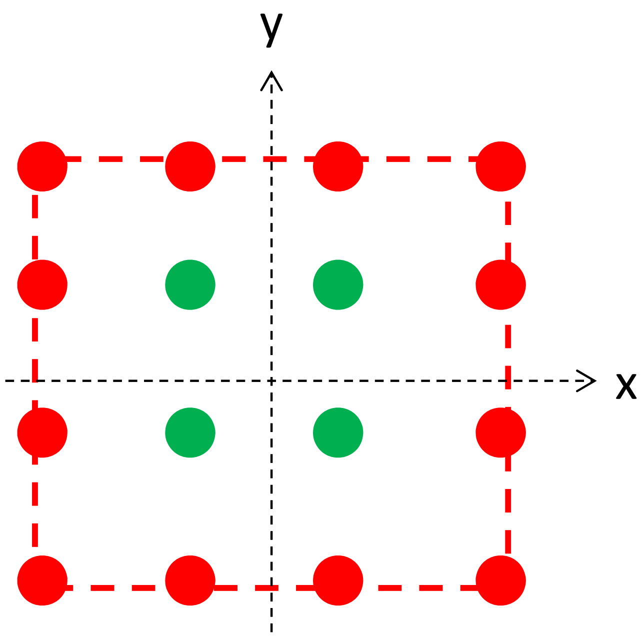
[1] Successively sweep through all of the interior grid points, where on the (m+1)th sweep (iteration) through all of the grid points :

$$\Phi_{i,j}^{m+1} = \Phi_{i,j}^{m} + \frac{1}{4}\left[\left(\Phi_{i+1,j}^{m} + \Phi_{i-1,j}^{m}\right) + \left(\Phi_{i,j+1}^{m} + \Phi_{i,j-1}^{m}\right) - 4\Phi_{i,j}^{m}\right]$$

$$= \Phi_{i,j}^{m} + \Delta\Phi_{i,j}^{m}$$

value from previous iteration

residual of mth iteration



Basic "Jacobi Iteration" scheme :

Step 0: Fix the boundary values; choose initial guesses for the interior points.

Iteration 1: Calculate Φ at all of the interior grid points according to the formulas below. The values of Φ at all of the interior grid points have been "updated".

Iteration 2: Re-calculate Φ at all of the interior grid points using the "updated" values from Iteration 1. Again update the values of Φ at each grid point.

Continue N times …

$$
\begin{aligned}
\Phi_{i,j}^{m+1} &= \Phi_{i,j}^{m} + \frac{1}{4}\left[\left(\Phi_{i+1,j}^{m} + \Phi_{i-1,j}^{m}\right) + \left(\Phi_{i,j+1}^{m} + \Phi_{i,j-1}^{m}\right) - 4\Phi_{i,j}^{m}\right] \\
&= \Phi_{i,j}^{m} + \Delta\Phi_{i,j}^{m}
\end{aligned}
$$

value from previous iteration

residual of m^{th} iteration

After many iterations, the residual will become small (i.e., the solution will have "relaxed" to its true value).

Notes :

[1] Under this scheme, the values of Φ on the boundary are not modified.

[2] The accuracy of the solution will, in general, depend on the grid spacing h.

[3] The accuracy will, in general, improve with the number of iterations N, but is subject to [2]. (The CPU time will also, in general, scale with N.)

Extension to 3D is straightforward. Have a 7-point stencil, where the value of Φ at (i,j,k) depends on its six neighboring points:

(i+1,j,k), (i-1,j,k), (i,j+1,k), (i,j-1,k), (i,j,k+1), (i,j,k-1)

Jacobi iteration is the simplest/most-elementary approach to a numerical solution of the Laplace Equation via relaxation.

More sophisticated methods (e.g., Gauss-Seidel, Successive Overrelaxation, Multigrid Methods, etc.) exist which improve both the accuracy and speed towards convergence.

See, for example, *Numerical Recipes in C++*.