A General 2D Electrostatic Boundary Problem Solver

Christopher Docherty

***Abstract*— In creating this software package the performance of 3 different relaxation algorithms (Jacobi, Gauss Seidel and Successive Over Relaxation) was analysed. As the optimal parameter for Taylor expansions on a square mesh has been determined analytically for SOR[1] it was shown to converge magnitudes faster than Jacobi and Gauss Seidel. The program is intended for use in closed systems but in cases where the boundaries do not enclose the region of interest two solutions are offered: interpolation of outside points and weighting of edge values.**

I. Introduction

This project aimed to produce software that could solve any arbitrary 2D electrostatic boundary value problem. This is achieved by serving a GUI through which the user can create their BVP, and have returned a plot of potential.

II. Relaxation

The final program contains implementations of three different relaxation methods: Jacobi, Gauss-Seidel and Successive Over Relaxation (hereafter referred to as SOR). All of these algorithms require a way to differentiate between boundary and non-boundary points and this is implemented by simply using another boolean matrix which has the same dimension as the mesh. This matrix indicates whether the point is on a boundary or not and therefore if it should be relaxed or passed over by the algorithm.

The algorithm will loop over all of the points in the mesh, continuing to relax any non-boundary points, until either the supplied maximum number of iterations is reached or the change from one iteration to the next is less than a user determined minimum.

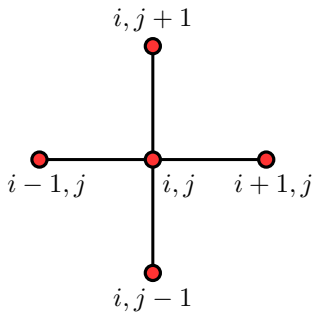

Fig. 1: The 5 point stencil used in Taylor expansion.

All of the algorithms use an equation which is the result of a Taylor expansion using a 5 point stencil (Fig. 1), around the point being updated. Given the discretisation of the second partial derivative of the potential (denoted by ϕ) with respect to a coordinate variable:

$$\frac{\partial^2 \phi_{i,j}}{\partial x^2} = \frac{\phi_{i+1,j} + \phi_{i-1,j} - 2\phi_{i,j}}{h^2} + O(h^2) \tag{1}$$

An expression for the Laplace equation can be derived (where $O(h^2)$ indicates that the error term grows with the square of h).

$$0 = \frac{\phi_{i+1,j} + \phi_{i-1,j} + \phi_{i,j+1} + \phi_{i,j-1} - 4\phi_{i,j}}{h^2} + O(h^2) \tag{2}$$

Rearranging to make $\phi_{i,j}$ the subject gives:

$$\phi_{i,j} = \frac{1}{4}[\phi_{i+1,j} + \phi_{i-1,j} + \phi_{i,j+1} + \phi_{i,j-1}] + O(h^4) \tag{3}$$

Where $\phi_{i,j}$ is the potential at a point with coordinates i, j on the mesh and h is the mesh spacing. Having an error term that grows with the 4th power of the mesh spacing is beneficial in that accuracy can be significantly improved just by decreasing the mesh spacing.

A. Relaxation methods

The least optimal method, in terms of both computation time and memory management, is the Jacobi method. On every relaxation iteration the algorithm first saves a copy of the mesh and then uses the values in the copy for the left hand side of Eq. 3, looping over all mesh points.

Gauss-Seidel is a simple improvement to the Jacobi scheme: where the mesh is relaxed in place meaning no extra memory is used to store a copy but, crucially, that updated mesh values can instantly be used for relaxing points further ahead or below.

SOR improves Gauss-Seidel by introducing a relaxation parameter ω which can be tuned to a specific problem. The relaxation scheme updates the mesh as a weighted average of the previous value and the updated value given by the Gauss-Seidel method. The update scheme for SOR is:

$$\phi_{i,j}^{(k+1)} = (1-\omega)\phi_{i,j}^{(k)} + \omega\phi_{i,j}, \;\; \omega \in [0, 2] \tag{4}$$

Where $\phi_{i,j}$ is from Eq.(3) and $\phi_{i,j}^{(k)}$ is the value of the mesh point from the previous iteration. The relaxation parameter has been shown to belong in the range $[0, 2]$ for all cases in literature and the optimal value is typically chosen by testing the convergence of multiple different values in the valid range. However, in the case of 2D Taylor expansion over the points on a square mesh, the optimal relaxation parameter has been determined analytically[1] and is shown to only depend on the mesh spacing h :

$$\omega_{optimal} = \frac{2}{1 + sin(\pi h)} \tag{5}$$

In the program the inexpensive calculation of the optimal parameter is carried out once and the result used in the main relaxation loop.

B. Theory behind relaxation

The previous section derived the relaxation update schema but it fails to convincingly prove why the algorithm solves the system. In order to offer an explanation as to why the algorithm actually converges on the true solution the problem must be restated in terms of matrices. Here the matrix form will be derived for a 1D system for the sake of brevity but the 2D case is very similar and the resulting matrix has the same properties as in the 1D case.

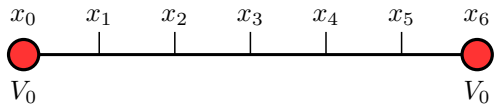

Fig. 2: 1D discretised electrostatic boundary value problem.

By discretising a 1D boundary problem (fig. 2) it is possible to construct a system of linear equations (Eq. 6) using the Taylor expansions about each point in along the line.

$$\begin{aligned} -2x_1 + x_2 &= V_0 \\ x_1 - 2x_2 + x_3 &= 0 \\ x_2 - 2x_3 + x_4 &= 0 \\ x_3 - 2x_4 + x_5 &= 0 \\ x_4 - 2x_5 + x_6 &= 0 \\ x_5 - 2x_6 &= V_0 \end{aligned} \tag{6}$$

This system can easily be converted to the following matrix equation:

$$\begin{bmatrix} -2 & 1 & & & & & \\ 1 & -2 & 1 & & & & \\ & 1 & -2 & 1 & & & \\ & & 1 & -2 & 1 & & \\ & & & 1 & -2 & -1 & \\ & & & & 1 & -2 & 1 \\ & & & & & 1 & -2 \end{bmatrix} \begin{bmatrix} x_0 \\ x_1 \\ x_2 \\ x_3 \\ x_4 \\ x_5 \\ x_6 \end{bmatrix} = \begin{bmatrix} V_0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ V_0 \end{bmatrix} \tag{7}$$

Which, using concise notation, is just:

$$A\boldsymbol{x} = \boldsymbol{b} \tag{8}$$

Now that the system is represented in this form relaxation methods can now be used. Note that its doesn't matter what the specific details of the initial system actually are, as long as the system can be reduced to this form, the relaxation methods can be applied. Thus in the 2D case the Taylor expansions would still be found around every point but the $\boldsymbol{x}$ vector is actually a vertical stack of all the points on the 2D mesh giving an A matrix with dimensions $[N^2, N^2]$ where N is the length of the mesh.

Given Eq. 8, the next step in applying a relaxation scheme is to split the matrix A up into different parts depending on which method is chosen. As it is the simplest, the Jacobi method will be used for the rest of this derivation which calls for splitting A into two parts: a matrix where all of diagonal elements are 0 and the diagonal elements are the same as in A which is denoted by D and another matrix N where all diagonal elements are 0 and all of diagonal elements are 0. Leading on from Eq. 8, this gives:

$$[D + N]\boldsymbol{x} = \boldsymbol{b} \tag{9}$$

Which is then rearranged to give an $\boldsymbol{x}$ on both sides:

$$D\boldsymbol{x} = -N\boldsymbol{x} + \boldsymbol{b} \tag{10}$$

$$\boldsymbol{x} = -D^{-1}N\boldsymbol{x} + D^{-1}\boldsymbol{b} \tag{11}$$

Then, simplifying the terms:

$$\boldsymbol{x} = B\boldsymbol{x} + \boldsymbol{z} \tag{12}$$

Now the equation is in a form where an iterative scheme can be defined by simply defining the $(k+1)th$ iteration's guess for $\boldsymbol{x}$ in terms of the kth iteration's guess.

$$\boldsymbol{x}^{(k+1)} = B\boldsymbol{x}^{(k)} + \boldsymbol{z} \tag{13}$$

At this point it is useful to represent $\boldsymbol{x^{(k)}}$ as the true solution $\boldsymbol{x}$ with an error term $\boldsymbol{e^{(k)}}$ which gives:

$$\boldsymbol{x} + \boldsymbol{e}^{(k+1)} = B[\boldsymbol{x} + \boldsymbol{e}^{(k)}] + \boldsymbol{z} \tag{14}$$

$$\boldsymbol{x} + \boldsymbol{e}^{(k+1)} = (B\boldsymbol{x} + \boldsymbol{z}) + B\boldsymbol{e}^{(k)} \tag{15}$$

From Eq. 12 clearly,

$$\boldsymbol{e}^{(k+1)} = B\boldsymbol{e}^{(k)} \tag{16}$$

Now there is a precise relationship between the error from one iteration to the next which depends only one result of multiplying an arbitrary vector by the matrix B. To understand the effect this will have it is pertinent now to briefly discuss the geometric effect of matrix multiplication.

C. Geometric transformations

For any arbitrary matrix M, the geometric effect of its multiplication is always defined for its eigenvectors, that is vectors which satisfy the equation:

$$M\boldsymbol{v} = \lambda\boldsymbol{v} \tag{17}$$

where λ is the eigenvector's corresponding eigenvalue. From Eq. 17 the geometric effect of multiplying the eigenvector $\boldsymbol{v}$ by M is clearly to scale the length of the vector by a factor of λ - with a possible 180^o change in direction if $\lambda < 0$. For proving the convergence of iterative relaxation

schemes, the behaviour of interest is the eventual behaviour of repeated matrix multiplications and, in particular, the criteria for the vector to tend towards $\mathbf{0}$. For eigenvectors the only required criteria for this behaviour is that the corresponding eigenvalue belongs to the range $\lambda \in (-1, 1)$ as the vector will decrease in magnitude each iteration.

Using a property of M called the spectral radius (denoted by $\rho(M)$) which is defined as the largest absolute value of M's eigenvalues a more general statement can be made. For any matrix M, if we have that $\rho(M) < 1$ then any arbitrary vector that can be represented as a linear combination of M's egienvectors, will have its magnitude tend to 0 upon repeated multiplication by the matrix M.

D. Diminishing error

Thus, to show that the error in Eq. 16 will diminish as the iterative scheme is used it suffices to show that the error is a linear combination of B's eigenvectors. Fortunately, it has already been proved in literature that for both the 1D and 2D cases the eigenvectors of B in each case span $\mathbb{R}^n$ where n is the dimension of both the solution $\boldsymbol{x}$ and the error $\boldsymbol{e}^{(k)}$. Therefore, no matter what the initial guess $\boldsymbol{x}^{(1)}$ is, the error will diminish with repeated applications and the guess will tend towards the solution to the equation.

Although represented very differently, Eq. 3 and Eq. 13 represent the same schema where the 2D Taylor expansion's B matrix picks out the 4 terms used in the right hand side of Eq. 3.

The Gauss Seidel and SOR methods only differ from Jacobi in how the matrix A in Eq. 8 is decomposed. For Gauss Seidel the matrix is decomposed into its lower triangular L_* and the remaining elements above the diagonal U and has the equivalent to Eq. 11:

$$\boldsymbol{x} = -L_*^{-1}U\boldsymbol{x} + L_*^{-1}\boldsymbol{b} \tag{18}$$

SOR decomposes the matrix into strictly upper triangular U, strictly lower triangular L and diagonal D and introduces a relaxation parameter ω to take Eq. 8 to:

$$\omega[L + D + U]\boldsymbol{x} = \omega\boldsymbol{b} \tag{19}$$

$$[D + \omega L]\boldsymbol{x} = -[\omega U + (\omega - 1)D]\boldsymbol{x} + \omega\boldsymbol{b} \tag{20}$$

Giving the equivalent to Eq. 11 and Eq. 18 of:

$$\boldsymbol{x} = -[D+\omega L]^{-1}[\omega U+(\omega-1)D]\boldsymbol{x}+[D+\omega L]^{-1}\omega\boldsymbol{b} \tag{21}$$

Again the B matrices associated with these methods have been shown to satisfy the sufficient criteria for convergence: $\rho(B) < 1$ and $span(\{\boldsymbol{v}|B\boldsymbol{v} = \lambda\boldsymbol{v}\}) = \mathbb{R}^n$.

E. Edge cases

The optimally determined parameter does not apply for every use case of the program as the relaxation algorithm does not have a criteria for handling non-boundary edges (i.e. edges of our mesh that are not part of a boundary) where a 5 point stencil cannot be made. This results in some points not using the Taylor Expansion and hence the parameter is no longer guaranteed to be optimal (although, in practice it is found to still be near optimal).

However, this limitation must be dealt with in a sensible manner. Two different cases have been identified and two different methods have been implemented to mitigate the limitations of the program (with the user able to choose the most suitable for their situation).

The first scenario occurs when some boundary with a non-zero potential extends beyond the region of interest - as is the case in the problem shown in Fig. **??** where the plates are infinitely long. In this case attempts must be made to reflect the fact that the plate does not end at the edges of the region of interest. The simple solution to this problem is to replace the non existent points in Eq. 3 with the point currently being considered i.e. in relaxing $\phi_{1,0}$, $\phi_{1,-1}$ is a non-existent point so it is replaced with $\phi_{1,0}$ in Eq. 3. This method allows the user to gain an accurate insight into how these types of systems' electric fields and potentials will appear. However, a disparity has been found in the accuracy of the method compared to analytical solutions so it is not recommended to use the algorithm to find the potential of the points near the non-boundary edges. Predictions were still found to be reliable far from these edges and the overall picture may still be useful in some cases.

The second scenario occurs when all boundaries are enclosed in the region of interest and the field would fall off at these non-boundary edges. To handle these cases and capture the nature of the field beyond the boundaries, the virtual, "ghost" points have their potential calculated using Lagrange interpolation with 3 points:

$$\phi_{i,j} = \sum_{n=1}^{3} p_{n,j}(x)\phi_{i,j-n} \tag{22}$$

$$p_{n,j}(x) = \prod_{k!=j}^{n} \frac{x - x_k}{x_j - x_k} \tag{23}$$

This method has been shown to work well in situations such as a discrete point boundary distributions and infinitely extending boundaries. The way these non-boundary edges are handled is left for the user to choose based on which they believe to be most useful in their specific BVP.

REFERENCES

[1] Arieh Iserles (2009). A first course in the numerical analysis of differential equations. Cambridge: Cambridge University Press.