**University of Massachusetts, Lowell**
Department of Mechanical Engineering


# Finite Difference Poisson Solver Over a Bilinearly Mapped Domain

**Patrick Heng**

MECH 5200

Prof. D.J. Willis

02/26/25

## Abstract

In the following, we explore a MATLAB based finite difference solver for the Poisson problem over a general quadrilateral domain. A bilinear mapping will be used to transform a unit square computational mesh into the quadrilateral physical domain. The transformation introduces a mapped Laplacian that will be approximated to second order and whose error will be studied in the $L_2$ and $L_\infty$ norms. To test the solver, a simple model of electrolocation will be used as a case study.

## Academic Integrity

I hereby affirm that the following work submitted is my own and that I have not received help from others outside of this class. Furthermore, I shall reference any resources used outside of class lectures and notes.

– P.V. Heng

# Contents

**1 Bilinear Mapping** **2**

**2 Finite Difference Formulas** **3**

**3 Discretized Governing Equations** **3**

**4 Boundary Conditions** **5**

**5 Postprocessing - E Field** **8**

**6 Pseudo Code** **9**

**7 Programming the Solver** **10**

**8 Solution** **11**

**9 Electrolocation Sensor** **13**

**References** **15**

**Appendix A - Functions** **16**

**Appendix B - Test Scripts** **28**

**Appendix C - Extra** **39**

# 1 Bilinear Mapping

We assume that the physical coordinates, $(x, y)$, of our domain may be mapped from the unit square computational domain, $(\xi, \eta)$, as follows,

$$x(\xi, \eta) = R_x\xi\eta + S_x\xi + T_x\eta + U_x,$$
$$y(\xi, \eta) = R_y\xi\eta + S_y\xi + T_y\eta + U_y,$$

for some constant coefficients, $R_x$, $S_x$, ... etc. Since we are considering a quadrilateral shape, the four corner points of the physical domain apply enough constraints to solve for the 8 coefficients. A possible system that may solved for $x(\xi, \eta)$ mapping is,

$$\begin{pmatrix} 0 & 0 & 0 & 1 \\ 0 & 1 & 0 & 1 \\ 1 & 1 & 1 & 1 \\ 0 & 0 & 1 & 1 \end{pmatrix} \begin{pmatrix} R_x \\ S_x \\ T_x \\ U_x \end{pmatrix} = \begin{pmatrix} -4 \\ 4 \\ 4 \\ -4 \end{pmatrix},$$

which gives upon inversion,

$$\begin{pmatrix} R_x \\ S_x \\ T_x \\ U_x \end{pmatrix} = \begin{pmatrix} 0 \\ 8 \\ 0 \\ -4 \end{pmatrix}.$$

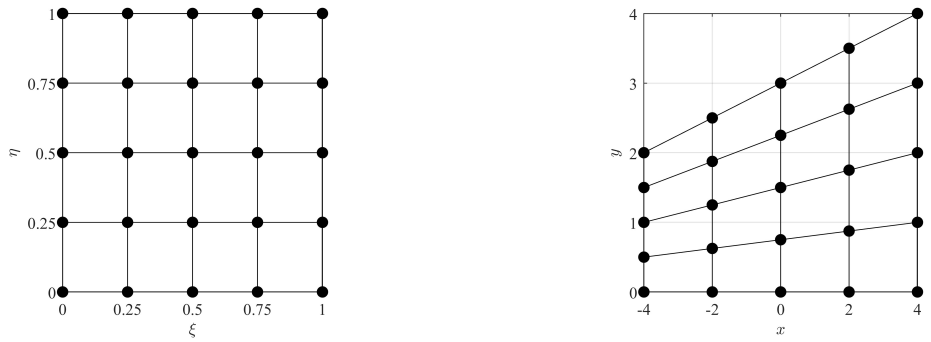
Likewise, the system solved for $y(\xi, \eta)$ mapping is,

$$\begin{pmatrix} 0 & 0 & 0 & 1 \\ 0 & 1 & 0 & 1 \\ 1 & 1 & 1 & 1 \\ 0 & 0 & 1 & 1 \end{pmatrix} \begin{pmatrix} R_y \\ S_y \\ T_y \\ U_y \end{pmatrix} = \begin{pmatrix} 0 \\ 0 \\ h_b \\ h_c \end{pmatrix},$$

which gives upon inversion,

$$\begin{pmatrix} R_y \\ S_y \\ T_y \\ U_y \end{pmatrix} = \begin{pmatrix} h_b - h_c \\ 0 \\ h_c \\ 0 \end{pmatrix}.$$

The 'bilinear_map' function was developed to calculate these mapping coefficients and also transform a $(\xi, \eta)$ meshgrid to an $(x, y)$ meshgrid. For example, the following figure demonstrates the transformation of a $5 \times 5$ uniform grid on the unit square being transformed to the physical domain through the previously defined mapping.




**(a)** Computational grid **(b)** Physical grid

**Figure 1:** Bilinear mapping between computational and physical grids with $(h_b, h_c) = (4, 2)$ using bilinear_map(xi,eta,[-4,4,4,-4],[0,0,4,2]), where the 'xi' and 'eta' are meshgrid variables and the first vector is the $x$ coordinates of $[d, a, b, c]$ and the second vector is the corresponding $y$ coordinates. See the documentation in Appendix A for a full explanation.

## 2 Finite Difference Formulas

Let $u$ be a three times continuously differentiable function. Furthermore, let $i$ denote the index of the $\xi_i$ nodes and $j$ denote the index of the $\eta_j$ nodes. By applying a central difference formula to the second derivatives, we find,

$$\frac{\partial^2 u}{\partial \xi^2} = \frac{u_{i+1,j} - 2u_{i,j} + u_{i-1,j}}{\Delta \xi^2} + \mathcal{O}\left(\Delta \xi^2\right),$$

$$\frac{\partial^2 u}{\partial \eta^2} = \frac{u_{i,j+1} - 2u_{i,j} + u_{i,j-1}}{\Delta \eta^2} + \mathcal{O}\left(\Delta \eta^2\right),$$

$$\frac{\partial^2 u}{\partial \xi \partial \eta} = \frac{u_{i+1,j+1} - u_{i+1,j-1} - u_{i-1,j+1} + u_{i-1,j-1}}{4\Delta \xi \Delta \eta} + \mathcal{O}\left(\max\{\Delta \xi, \Delta \eta\}^2\right).$$

By applying a central difference formula to the first derivatives, we have,

$$\frac{\partial u}{\partial \xi} = \frac{u_{i+1,j} - u_{i-1,j}}{2\Delta \xi} + \mathcal{O}\left(\Delta \xi^2\right),$$

$$\frac{\partial u}{\partial \eta} = \frac{u_{i,j+1} - u_{i,j-1}}{2\Delta \eta} + \mathcal{O}\left(\Delta \eta^2\right).$$

Since we wish to impose Neumann boundary conditions, we require second order forward and backward difference formulas in $\xi$ and $\eta$. The forward differences are,

$$\frac{\partial u}{\partial \xi} = \frac{-3u_{i,j} + 4u_{i+1,j} - u_{i+2,j}}{2\Delta \xi} + \mathcal{O}\left(\Delta \xi^2\right),$$

$$\frac{\partial u}{\partial \eta} = \frac{-3u_{i,j} + 4u_{i,j+1} - u_{i,j+2}}{2\Delta \eta} + \mathcal{O}\left(\Delta \eta^2\right),$$

similarly for the backward differences,

$$\frac{\partial u}{\partial \xi} = \frac{3u_{i,j} - 4u_{i-1,j} + u_{i-2,j}}{2\Delta \xi} + \mathcal{O}\left(\Delta \xi^2\right),$$

$$\frac{\partial u}{\partial \eta} = \frac{3u_{i,j} - 4u_{i,j-1} + u_{i,j-2}}{2\Delta \eta} + \mathcal{O}\left(\Delta \eta^2\right).$$

## 3 Discretized Governing Equations

From the lecture notes, the Poisson equation (at internal nodes) may be expressed in the computational coordinate system by,

$$-J^{-2}\left(a\frac{\partial^2 u}{\partial \xi^2} - 2b\frac{\partial^2 u}{\partial \xi \partial \eta} + c\frac{\partial^2 u}{\partial \eta^2} + d\frac{\partial u}{\partial \eta} + e\frac{\partial u}{\partial \xi}\right) = f,$$

where $f = 0$ in our example and we adopt the notation that the electric potential, $\phi$, is interchangeable with $u$. The $(J, a, b, c, d, e)$ coefficients may depend on $\xi$ and $\eta$ in general. They are defined such that,

$$a = x_\eta^2 + y_\eta^2, \quad b = x_\eta x_\xi + y_\eta y_\xi, \quad c = x_\xi^2 + y_\xi^2,$$

$$e = J^{-1}\left(x_\eta \beta - y_\eta \alpha\right), \quad d = J^{-1}\left(y_\xi \alpha - x_\xi \beta\right),$$

$$J = x_\xi y_\eta - x_\eta y_\xi,$$

$$\alpha = a x_{\xi\xi} - 2b x_{\xi\eta} + c x_{\eta\eta}, \quad \beta = a y_{\xi\xi} - 2b y_{\xi\eta} + c y_{\eta\eta},$$

where subscripts denote partial differentiation. For a general bilinear map,

$$x_\xi = R_x \eta + S_x, \quad x_\eta = R_x \xi + T_x,$$

$$y_\xi = R_y \eta + S_y, \quad y_\eta = R_y \xi + T_y,$$

which also simplifies $\alpha$ and $\beta$,

$$\alpha = -2bR_x, \quad \beta = -2bR_x.$$

By applying central difference formulas for the first and second derivatives, we have,

$$\begin{aligned} f_{i,j} \;=\; & -\frac{1}{J_{i,j}^2}\frac{a_{i,j}}{\Delta\xi^2}\left(u_{i+1,j} - 2u_{i,j} + u_{i-1,j}\right) \\ & -\frac{1}{J_{i,j}^2}\frac{-b_{i,j}}{2\Delta\xi\Delta\eta}\left(u_{i+1,j+1} - u_{i+1,j-1} - u_{i-1,j+1} + u_{i-1,j-1}\right) \\ & -\frac{1}{J_{i,j}^2}\frac{c_{i,j}}{\Delta\eta^2}\left(u_{i,j+1} - 2u_{i,j} + u_{i,j-1}\right) \\ & -\frac{1}{J_{i,j}^2}\frac{d_{i,j}}{2\Delta\eta}\left(u_{i,j+1} - u_{i,j-1}\right) \\ & -\frac{1}{J_{i,j}^2}\frac{e_{i,j}}{2\Delta\xi}\left(u_{i+1,j} - u_{i-1,j}\right), \end{aligned}$$

where the $(J, a, b, c, d, e)$ coefficients are evaluated at the central node. By grouping nodal points,

$$\begin{aligned} f_{i,j} \;=\; & \underbrace{-\frac{1}{J_{i,j}^2}\left(\frac{a_{i,j}}{\Delta\xi^2} + \frac{e_{i,j}}{2\Delta\xi}\right)}_{E_{i,j}} u_{i+1,j} + \underbrace{\frac{2}{J_{i,j}^2}\left(\frac{a_{i,j}}{\Delta\xi^2} + \frac{c_{i,j}}{\Delta\eta^2}\right)}_{P_{i,j}} u_{i,j} + \underbrace{-\frac{1}{J_{i,j}^2}\left(\frac{a_{i,j}}{\Delta\xi^2} - \frac{e_{i,j}}{2\Delta\xi}\right)}_{W_{i,j}} u_{i-1,j} \\ & + \underbrace{-\frac{1}{J_{i,j}^2}\left(\frac{c_{i,j}}{\Delta\eta^2} + \frac{d_{i,j}}{2\Delta\eta}\right)}_{N_{i,j}} u_{i,j+1} + \underbrace{-\frac{1}{J_{i,j}^2}\left(\frac{c_{i,j}}{\Delta\eta^2} - \frac{d_{i,j}}{2\Delta\eta}\right)}_{S_{i,j}} u_{i,j-1} \\ & + \underbrace{-\frac{1}{J_{i,j}^2}\left(\frac{-b_{i,j}}{2\Delta\xi\Delta\eta}\right)}_{NE_{i,j}} u_{i+1,j+1} + \underbrace{\frac{1}{J_{i,j}^2}\left(\frac{-b_{i,j}}{2\Delta\xi\Delta\eta}\right)}_{SE_{i,j}} u_{i+1,j-1} \\ & + \underbrace{\frac{1}{J_{i,j}^2}\left(\frac{-b_{i,j}}{2\Delta\xi\Delta\eta}\right)}_{NW_{i,j}} u_{i-1,j+1} + \underbrace{-\frac{1}{J_{i,j}^2}\left(\frac{-b_{i,j}}{2\Delta\xi\Delta\eta}\right)}_{SW_{i,j}} u_{i-1,j-1} \end{aligned}$$

Visually, these stencil coefficients correspond to the grid points in the following figure and table.

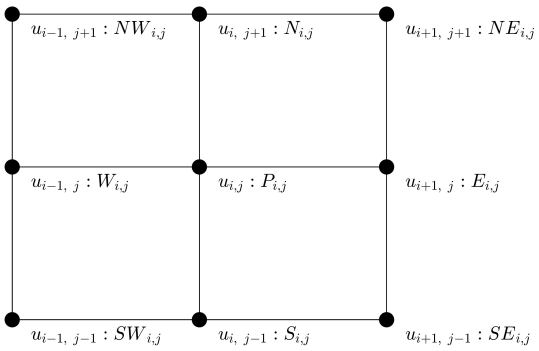



**Figure 2:** 9 point stencil coefficients for node $u_{i,j}$ over the computational mesh

**Table 1:** 9 Point Stencil Coefficients for Second Order Mapped Laplacian

| | | |
|---|---|---|
| $NW_{i,j} = \frac{1}{J_{i,j}^2}\left(\frac{-b_{i,j}}{2\Delta\xi\Delta\eta}\right)$ | $N_{i,j} = -\frac{1}{J_{i,j}^2}\left(\frac{c_{i,j}}{\Delta\eta^2} + \frac{d_{i,j}}{2\Delta\eta}\right)$ | $NE_{i,j} = -\frac{1}{J_{i,j}^2}\left(\frac{-b_{i,j}}{2\Delta\xi\Delta\eta}\right)$ |
| $W_{i,j} = -\frac{1}{J_{i,j}^2}\left(\frac{a_{i,j}}{\Delta\xi^2} - \frac{e_{i,j}}{2\Delta\xi}\right)$ | $P_{i,j} = \frac{2}{J_{i,j}^2}\left(\frac{a_{i,j}}{\Delta\xi^2} + \frac{c_{i,j}}{\Delta\eta^2}\right)$ | $E_{i,j} = -\frac{1}{J_{i,j}^2}\left(\frac{a_{i,j}}{\Delta\xi^2} + \frac{e_{i,j}}{2\Delta\xi}\right)$ |
| $SW_{i,j} = -\frac{1}{J_{i,j}^2}\left(\frac{-b_{i,j}}{2\Delta\xi\Delta\eta}\right)$ | $S_{i,j} = -\frac{1}{J_{i,j}^2}\left(\frac{c_{i,j}}{\Delta\eta^2} - \frac{d_{i,j}}{2\Delta\eta}\right)$ | $SE_{i,j} = \frac{1}{J_{i,j}^2}\left(\frac{-b_{i,j}}{2\Delta\xi\Delta\eta}\right)$ |

If we let 'A' denote the finite difference matrix for the Laplacian, then the coefficients,

$$[\text{SW},\ \text{S},\ \text{SE},\ \text{W},\ \text{P},\ \text{E},\ \text{NW},\ \text{N},\ \text{NE}],$$

are placed on the,

$$\text{diag\_idx} = [-\text{N}-1,\ -\text{N},\ -\text{N}+1,\ -1,\ 0,\ 1,\ \text{N}-1,\ \text{N},\ \text{N}+1],$$

diagonals and off diagonals of the A matrix. This can done be with iterated for-loops, but to take advantage of MATLAB's vectorized commands, we will evaluate these coefficients using the $N \times M$ meshgrid variables. Then, the resulting $N \times M$ nodal coefficients will be reshaped into $(N * M) \times 1$ column vectors. These vectors are then placed side by side to form a matrix of coefficients. That is, we define a 'diag' variable as,

$$\text{diag} = [\text{SW},\ \text{S},\ \text{SE},\ \text{W},\ \text{P},\ \text{E},\ \text{NW},\ \text{N},\ \text{NE}],$$

which is a dense $(N * M) \times 9$ matrix. Constructing the bulk of the A matrix now simply becomes,

$$\text{A} = \text{spdiags}(\text{diag},\ \text{diag\_idx},\ \text{N} * \text{M},\ \text{N} * \text{M} + 1),$$

where we have generated a $(N*M)\times(N*M+1)$ matrix with the columns of the diag matrix on the diag_idx diagonals. To get the correct indexing for the coefficients on the diagonals, we must generate A with an extra column (see [2] for documentation [1]). The extra column is then removed with,

$$\text{A} = \text{A}(:, 1 : \text{N} * \text{M}).$$

This generates the A matrix at all of the internal nodes, but with incorrect boundary conditions, which may be fixed afterwards.

# 4 Boundary Conditions

## Neumman Conditions

Since we wish to impose Neumann conditions on all boundaries, the normal derivatives and gradients must be computed in the computational domain. For the boundaries of constant $\xi$, the unit normal, $\mathbf{n}_\xi$, is given by [2],

$$\mathbf{n}_\xi = (n_\xi^x, n_\xi^y) = \frac{1}{\sqrt{x_\eta^2 + y_\eta^2}}(y_\eta, -x_\eta),$$

similarly, for boundaries of constant $\eta$,

$$\mathbf{n}_\eta = (n_\eta^x, n_\eta^y) = \frac{1}{\sqrt{x_\xi^2 + y_\xi^2}}(-y_\xi, x_\xi).$$

[1] This took me a while to figure out, but I think it's quite an elegant 'MATLABism'!

[2] The normal vectors may flip signs depending on the boundary, but since there is no flux, the sign doesn't matter.

From this, the normal derivative along a boundary of constant $\xi$ may be given as,

$$\nabla u \cdot \mathbf{n}_\xi = \underbrace{J^{-1}\left[y_\eta n_\xi^x - x_\eta n_\xi^y\right]}_{N_{\xi\xi}} \frac{\partial u}{\partial \xi} + \underbrace{J^{-1}\left[-y_\xi n_\xi^x + x_\xi n_\xi^y\right]}_{N_{\xi\eta}} \frac{\partial u}{\partial \eta},$$

similarly, for boundaries of constant $\eta$,

$$\nabla u \cdot \mathbf{n}_\eta = \underbrace{J^{-1}\left[y_\eta n_\eta^x - x_\eta n_\eta^y\right]}_{N_{\eta\xi}} \frac{\partial u}{\partial \xi} + \underbrace{J^{-1}\left[-y_\xi n_\eta^x + x_\xi n_\eta^y\right]}_{N_{\eta\eta}} \frac{\partial u}{\partial \eta}.$$

Above, we have adopted a 'stress tensor-like' notation where the first index of $N_{(\cdot,\cdot)}$ denotes the normal direction and the second index denotes the derivative direction. The $N_{(\cdot,\cdot)}$ coefficients may be evaluated on the appropriate boundaries (they will be $N\times 1$ or $M\times 1$ vectors). The discretized boundary conditions simply come from using forward or backward differencing on the derivatives. For example, on the left boundary,

$$\left.\frac{\partial u}{\partial n}\right|_{left} = \frac{N_{\xi\xi}}{2\Delta\xi}\left(-3u_{i,j} + 4u_{i+1,j} - u_{i+2,j}\right) + \frac{N_{\xi\eta}}{2\Delta\eta}\left(u_{i,j+1} - u_{i,j-1}\right),$$

which corresponds to a 5 point stencil. In the A matrix, the rows corresponding to the left boundary would have,

$$\text{diag} = \left[-\frac{N_{\xi\eta}}{2\Delta\eta}, \frac{-3N_{\xi\xi}}{2\Delta\xi}, \frac{4N_{\xi\xi}}{2\Delta\xi}, \frac{-N_{\xi\xi}}{2\Delta\xi}, \frac{N_{\xi\eta}}{2\Delta\eta}\right],$$

on the [−N, 0, 1, 2, N] diagonals. Similarly, the rows corresponding to the right boundary have,

$$\text{diag} = \left[-\frac{N_{\xi\eta}}{2\Delta\eta}, \frac{N_{\xi\xi}}{2\Delta\xi}, \frac{-4N_{\xi\xi}}{2\Delta\xi}, \frac{3N_{\xi\xi}}{2\Delta\xi}, \frac{N_{\xi\eta}}{2\Delta\eta}\right],$$

on the [−N, −2, −1, 0, N] diagonals of the A matrix. The discretized bottom boundary has,

$$\left.\frac{\partial u}{\partial n}\right|_{btm} = \frac{N_{\eta\xi}}{2\Delta\xi}\left(u_{i+1,j} - u_{i-1,j}\right) + \frac{N_{\eta\eta}}{2\Delta\eta}\left(3u_{i,j} - 4u_{i,j-1} + u_{i,j-2}\right).$$

implying that the rows corresponding to the bottom boundary would have,

$$\text{diag} = \left[\frac{N_{\eta\eta}}{2\Delta\eta}, \frac{-4N_{\eta\eta}}{2\Delta\eta}, \frac{-N_{\eta\xi}}{2\Delta\xi}, \frac{3N_{\eta\eta}}{2\Delta\eta}, \frac{N_{\eta\xi}}{2\Delta\xi}\right],$$

on the [−2N, −N, −1, 0, 1] diagonals. Similarly, for the top nodes, the A matrix would have,

$$\text{diag} = \left[\frac{-N_{\eta\xi}}{2\Delta\xi}, \frac{-3N_{\eta\eta}}{2\Delta\eta}, \frac{N_{\eta\xi}}{2\Delta\xi}, \frac{4N_{\eta\eta}}{2\Delta\eta}, \frac{-N_{\eta\eta}}{2\Delta\eta}\right],$$

on the [−1, 0, 1, N, 2N] diagonals. To apply these boundary conditions for any stencil and any diagonal indices, the following code outline holds. k denotes either 1, N, or, M and may switch with i depending on the boundary. 'NN(·,·)' denotes the node numbering function.

**Algorithm 1** Application of Neumman boundary conditions

1: Find the nodes corresponding to the boundary: idx = NN(·,·)
2: Calculate the stencil coefficients along the boundary using the above formulas: diag = ⋯
3: Store the corresponding diagonal indices: diag_idx = ⋯
4: Replace the rows of A corresponding to the boundary with the corresponding rows of the diag matrix:
5: **for** i = 2 : size(idx, 1) − 1 **do**
6: A(NN(i, k), :) = 0; ▷ Clear row of A
7: A(NN(i, k), NN(i, k) + diag_idx) = diag(i, :); ▷ Place coefficients on the *correct* diagonals
8: f(NN(i, k)) = ⋯; ▷ Set the desired flux (0 for no flux)
9: **end for**

The 4 corner nodes are unique and require double forward and backwards differencing; the formulas are similar to the others. Refer to the 'impose_BCs', 'normals_xi', and 'normals_eta' functions in Appendix A for exact implementation. Additionally, since the corner nodes do not have a unique normal vector, the two normal vectors at the corners were averaged.

## Dirichlet Conditions

The Dirichlet conditions may be imposed by applying logical indexing on the bottom boundary. For example, let x denote the vector of physical nodal points along the bottom boundary. An if-statement would be applied to find the indices of the x vector where $-2 < \text{x} < 0$, let us denote these indices by 'idx_left'. Another if-statement is used to find the indices where $0 < \text{x} < 2$, which we may denote by 'idx_right'. The rows of the A matrix corresponding to 'idx_left' would then be replaced by the corresponding rows of the identity matrix. Similarly, the rows of A corresponding to 'idx_right' would have their rows replaced by the approriate rows of the identity. If the vectorized forcing function ($f = 0$ in our example) is represented by f, then the rows of f corresponding to 'idx_left' would be set to $-1$ while the rows corresponding to 'idx_right' would be set to 1.

The above scheme would work in a general language, but again, we wish to take advantage of MATLAB's logical indexing syntax. Therefore, generation of the indices becomes,

$$\text{indices} = 1 : \text{n},$$

$$\text{idx\_left} = \text{indices}(-2 <= \text{x} \ \& \ \text{x} <= 0),$$

$$\text{idx\_right} = \text{indices}(0 <= \text{x} \ \& \ \text{x} <= 2).$$

If we let I denote the identity matrix, then imposing the boundary conditions simply becomes,

$$\text{A}(\text{NN}(\text{idx\_left}, \text{M}), :) = \text{I}(\text{NN}(\text{idx\_left}, \text{M}), :),$$

$$\text{A}(\text{NN}(\text{idx\_right}, \text{M}), :) = \text{I}(\text{NN}(\text{idx\_right}, \text{M}), :),$$

$$\text{f}(\text{NN}(\text{idx\_left}, \text{M})) = -1,$$

$$\text{f}(\text{NN}(\text{idx\_right}, \text{M})) = 1.$$

## General Considerations

Implementation of all the boundary conditions and stencil coefficients leads to an A matrix with the following pattern.

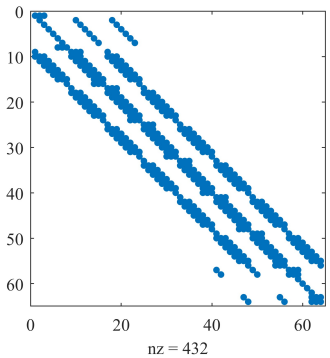



**Figure 3:** Sparsity plot of the A matrix on a $(N, M) = (8, 8)$ grid. The top rows of the matrix represent the top boundary while the bottom rows represent the bottom. The periodic 'breaks' in the diagonals are due to the forwards/backwards differencing on the left and right boundaries. The Dirichlet condition can be seen near the bottom of the matrix where the only non-zero entries are on the main diagonal.

The matrix assembly sequence would be to generate the A matrix with the incorrect boundary conditions, apply the Neumman conditions on all boundaries, then correct the bottom boundary with the Dirichlet conditions.

# 5 Postprocessing - E Field

The electric field, $\mathbf{E}$, is given by the negative gradient of the potential,

$$\mathbf{E} = -\left(\frac{\partial u}{\partial x}, \frac{\partial u}{\partial y}\right),$$

which may be converted to computational coordinates by the chain rule,

$$\mathbf{E} = -\left(J^{-1}\left[\frac{\partial u}{\partial \xi} y_\eta - \frac{\partial u}{\partial \eta} y_\xi\right], J^{-1}\left[-\frac{\partial u}{\partial \xi} x_\eta + \frac{\partial u}{\partial \eta} x_\xi\right]\right).$$

By applying central difference formulas, we find,

$$(E_x)_{i,j} = \frac{1}{2J}\left[y_\eta \frac{u_{i+1,j} - u_{i-1,j}}{\Delta\xi} - y_\xi \frac{u_{i,j+1} - u_{i,j-1}}{\Delta\eta}\right],$$

$$(E_y)_{i,j} = \frac{1}{2J}\left[-x_\eta \frac{u_{i+1,j} - u_{i-1,j}}{\Delta\xi} + x_\xi \frac{u_{i,j+1} - u_{i,j-1}}{\Delta\eta}\right],$$

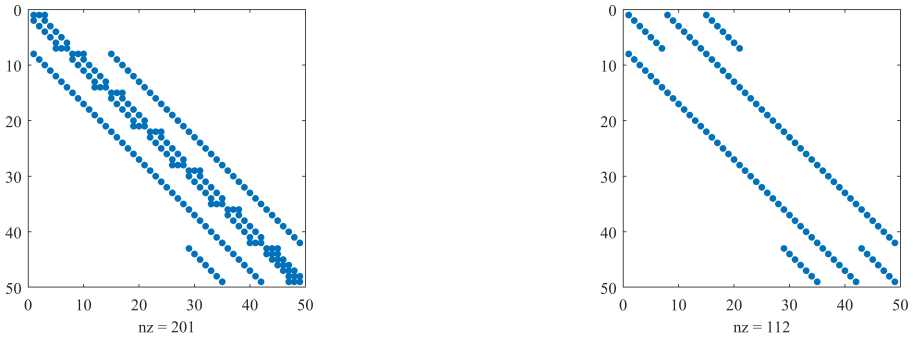
where $J$, $x_\xi$, $x_\eta$, etc. are evaluated at the central node. These formulas can be used to compute $\mathbf{E}$ on the internal nodes. This may be done with iterated for loops, but instead, we will again construct finite difference matrices such that multiplying the matrix by the solution vector, $u_k$, yields the (vectorized) $E_x$ or $E_y$ field. This is done essentially the same way as the construction of the Laplacian A matrix, but the diagonal coefficients and positions of the diagonals change. For example, to obtain the $\partial/\partial x$ matrix, $\mathrm{D_x}$, we collect the nodal coefficients from central differencing,

$$\text{diag\_x} = \frac{1}{2J_{i,j}}\left[\frac{-(y_\xi)_{i,j}}{\Delta\eta}, \frac{-(y_\eta)_{i,j}}{\Delta\xi}, \frac{(y_\eta)_{i,j}}{\Delta\xi}, \frac{(y_\xi)_{i,j}}{\Delta\eta}\right],$$

which is placed on the $[-\mathrm{N}, -1, 1, \mathrm{N}]$ diagonals of the $\mathrm{D_x}$ matrix. Similarly, for the $\mathrm{D_y}$ matrix,

$$\text{diag\_y} = -\frac{1}{2J_{i,j}}\left[\frac{-(x_\xi)_{i,j}}{\Delta\eta}, \frac{-(x_\eta)_{i,j}}{\Delta\xi}, \frac{(x_\eta)_{i,j}}{\Delta\xi}, \frac{(x_\xi)_{i,j}}{\Delta\eta}\right],$$

is placed on the $[-\mathrm{N}, -1, 1, \mathrm{N}]$ diagonals.




**(a)** $\mathrm{D_x}$ **(b)** $\mathrm{D_y}$

**Figure 4:** Sparsity plots of the $\mathrm{D_x}$ and $\mathrm{D_y}$ matrices to demonstrate the diagonal positioning of the stencil coefficients; $(N, M) = (7, 7)$. Observe that the main diagonal is missing for pure central differencing on the internal nodes.

The majority of the $D_x$ and $D_y$ matrices are then assembled by,

$$\text{Dx} = \text{spdiags(diag\_x, [}-\text{N, } -1, \ 1, \ \text{N], nodes, nodes} + 1); \quad \text{Dx} \ = \ \text{Dx(:, nodes);,}$$

$$\text{Dy} = \text{spdiags(diag\_y, [}-\text{N, } -1, \ 1, \ \text{N], nodes, nodes} + 1); \quad \text{Dy} \ = \ \text{Dy(:, nodes);.}$$

Correction of the derivatives at the boundary nodes again involves forward and backwards differencing. For example, the forward difference gives,

$$(E_x)_{i,j} = \frac{1}{2J}\left[y_\eta \frac{-3u_{i,j} + 4u_{i+1,j} - u_{i+2,j}}{\Delta\xi} - y_\xi \frac{-3u_{i,j} + 4u_{i,j+1} - u_{i,j+2}}{2\Delta\eta}\right],$$

$$(E_y)_{i,j} = \frac{1}{2J}\left[-x_\eta \frac{-3u_{i,j} + 4u_{i+1,j} - u_{i+2,j}}{\Delta\xi} + x_\xi \frac{-3u_{i,j} + 4u_{i,j+1} - u_{i,j+2}}{\Delta\eta}\right].$$

For a full implementation these corrections, refer to 'grad_field.m' in Appendix A. However, once these corrections are made, $E_x$ and $E_y$ are simply found by,

$$(E_x)_k = \text{D}_\text{x} u_k, \quad (E_y)_k = \text{D}_\text{y} u_k,$$

and reshaping the resulting vector back into the meshgrid shape.

# 6 Pseudo Code

**Algorithm 2** Finite difference Poisson solver over a transformed domain (test script)

1: Define mesh and discretization parameters
   N = val; M = val; hb = val; hc = val; nodes = N*M;
2: xi = linspace(0, 1, N); eta = linspace(0, 1, M);
3: Create a $N \times M$ uniform computational grid of $(\xi_i, \eta_j)$ nodes
   [Xi, Eta] = meshgrid(xi,eta)
4: Generate the bilinear mapping coefficients $(R_x, S_x, ...)$ based on $h_b$ and $h_c$ (Solve the system used in 'Bilinear Mapping', essentially just an x = A\b problem). Compute the physical domain [X, Y] from these coefficients
   X = Rx*Xi.*Eta + Sx*Xi + Tx*Eta + Ux;
   Y = Ry*Xi.*Eta + Sy*Xi + Ty*Eta + Uy;
5: Compute the mapping derivatives $(x_\xi, x_\eta, ...)$ at every point in the grid (using meshgrid variables). For example: x_xi = Rx*Eta + Tx
6: Compute the resulting mapping coefficients $(a, b, c, ...)$ at every point in the grid. Use the previously calculated x_xi, x_eta, ... variables along with element wise operations
7: Compute the forcing function, f, over the entire domain, in our example, f = zeros(nodes, 1)
8: Perform A matrix assembly and impose boundary conditions on A and f (see following code block)
9: Solve the linear system: u = A\f
10: Compute the **E** field using central differencing on the internal nodes of $u$ field and mixed backwards and forwards differencing on the boundaries. i.e. construct finite difference matrices, 'Dx' and 'Dy' (see 'Postprocessing' for stencil coefficients and code outline)
11: Reshape the u, Ex, and, Ey vectors back into $M \times N$ grids: u = flip(reshape(u, N, M)')
12: Plot results (in the physical domain): surf(X, Y, u), quiver(X, Y, Ex, Ey)

**Algorithm 3** A matrix assembly

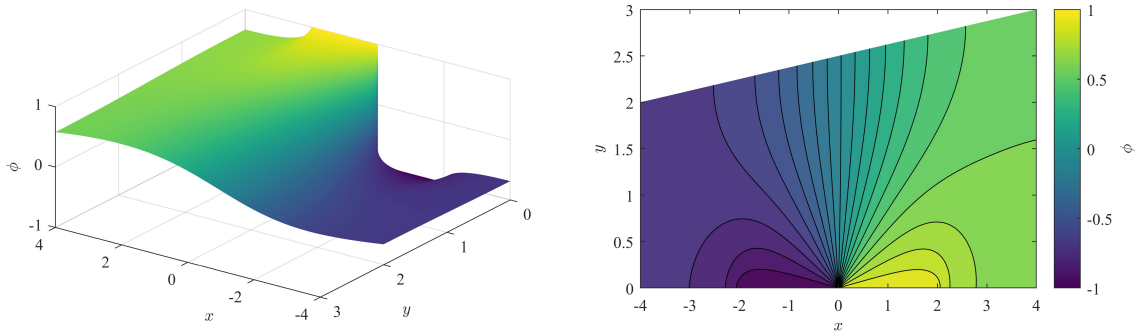
1: Get the mapping derivatives and coefficients previously computed over the mesh (see previous code block)
2: Compute the stencil coefficients (W, P, E ...) as defined in 'Discretized Governing Equations' using the meshgrid variables (J, a, b ...) and component-wise operations ($.*$, $./$); each stencil coefficient will be a $N \times M$ matrix
3: Vectorize the stencil coefficients such that lines of constant $\eta$ are stacked on top of each other, for example, using the SE coefficient, SE = reshape(SE', nodes, 1)
4: Create a dense (nodes $\times$ 9) stencil matrix by placing the vectorized stencil coefficients side by side: diag = [SW, S, SE, W, P, E, NW, N, NE]
5: Define the diagonal indices: diag_idx = [-N-1, -N, -N+1, -1, 0, 1, N-1, N, N+1]
6: Use spdiags to generate the A matrix at the internal nodes
A = spdiags(diag, diag_idx, nodes, nodes + 1); A = A(:, nodes)
7: Compute the normal derivative coefficients, $N_{(\cdot,\cdot)}$, on the appropriate boundaries
8: Impose a Neumann condition on the top and right boundaries of the A matrix using backward differencing; use forward differencing on the top and left boundaries (see 'Boundary Conditions' for code outline)
9: Impose Dirichlet conditions using logical indexing to find the appropriate nodes in the computational domain. If 'k' denotes the nodes with Dirichlet conditions, then set A(k, :) = eye(k, :)
10: Build the f vector, set all boundary nodes to 0, then correct the Dirichlet nodes by setting them to 1 or -1 (see 'Boundary Conditions' for code outline): f(k, :) = -1 or f(k, :) = 1

# 7 Programming the Solver

In an attempt to generalize the code, the solver was broken into many functions and test scripts. Appendix A contains the more general functions, including 'stencil_coefficients' which generates the diagonals of the Laplacian A matrix for any mapping (with a non-singular Jacobian). Also included are 'normals_xi' and 'normals_eta' functions which generate the stencil coefficients needed to impose normal derivative boundary conditions in the computational domain. Finally, 'grad_field' can compute the gradient field of any vector, $u$, given a general mapping. The remaining functions are more specific to the bilinear mapping and boundary conditions given in the problem. Appendix B contains the test scripts to call the previous functions. Much of the code in Appendix B is dedicated to generating and saving numerical data and using that data to generate the plots seen in the report.

# 8 Solution

The solution of the potential field with $(h_b, h_c) = (3, 2)$ yields the plots in the following section where the finest grid solution with $(N, M) = (1024, 1024)$ is presented. The 'HW_1_test.m' script found in Appendix B was used to run the solution and generate the plots.

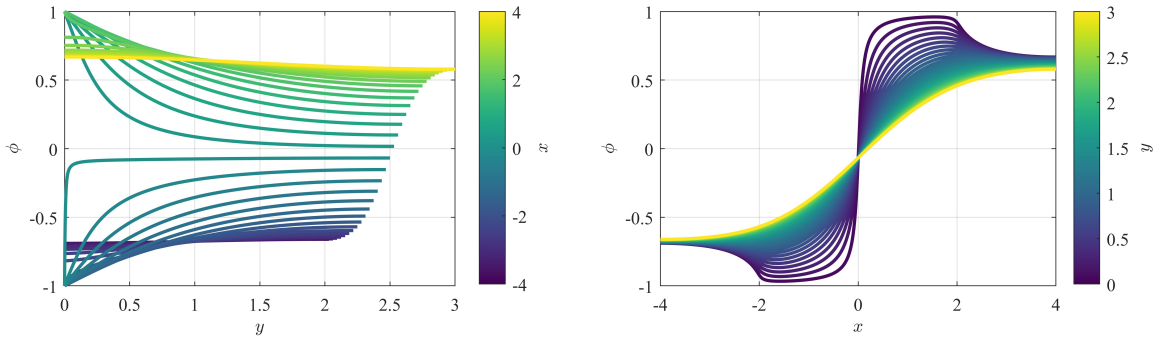



**(a)** Surface plot **(b)** Contour plot

**Figure 5:** Solution plots for the potential field, $\phi$, over a $(N, M) = (1024, 1024)$ grid with $(h_b, h_c) = (3, 2)$

Upon visual inspection, the solution satisfies the Dirichlet conditions at the bottom boundary and satisfies the maximum principle for the Poisson problem [4]. Additionally, the domain is mapped correctly as the physical quadrilateral has the correct corner nodes.

To verify the Neumann conditions, consider slices of the solution surface in lines of constant $x$ and $y$.



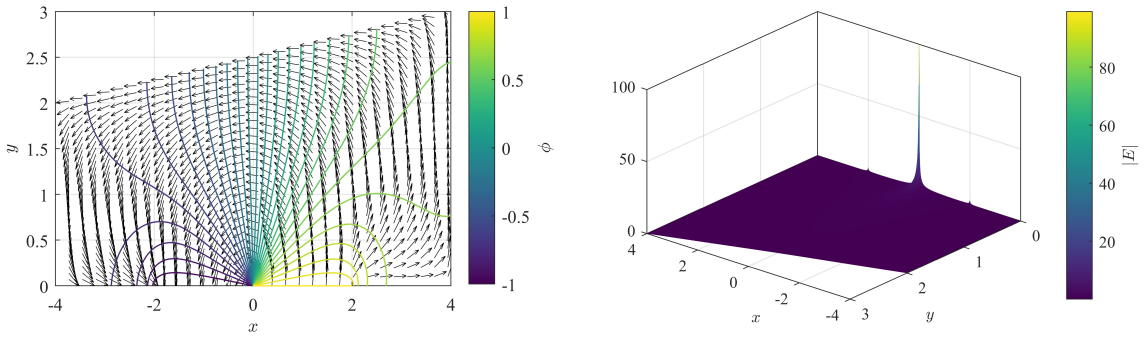

**(a)** Slices of constant $x$ **(b)** Slices of constant $y$

**Figure 6:** Surface projection plots on lines of constant $x$ and $y$ over a $(N, M) = (1024, 1024)$ grid with $(h_b, h_c) = (3, 2)$

From this, we see that the along the edges of the projection plots, the derivatives become 0 and the curves level out. This implies that the Neumman condition is being satisfied.

To verify the solution of the electric field, we present a quiver plot of gradient of the potential along with equipotential lines [3]. The **E** vectors have been normalized to unit length to prevent clutter. Also shown is a surface plot of the **E** field to demonstrate its singularity at the origin.

[3] Sorry for the quality of this figure, I can never get quiver plots to look good!




**(a)** Normalized **E** field with equally spaced equipotential lines

**(b)** Surface plot of the magnitude of the **E** field

**Figure 7:** **E** field over a $(N, M) = (1024, 1024)$ grid with $(h_b, h_c) = (3, 2)$

Visually, the **E** vectors are perpendicular to the lines of constant potential, which must be true by the definition of the gradient. This observation also further verifies that the computation of the mapping is correct. Additionally, the vectors never pierce the boundaries, which physically makes sense since the boundaries are impermeable. The vectors also emanate from the source at the right half of the domain and circulate into the sink at the left half of the domain.

## $L_2$ and $L_\infty$ Convergence

Let $h = \max(\Delta\xi, \Delta\eta)$ and let $u^*_{i,j}$ be the numerical solution on a highly refined mesh. By using central differencing and second order backwards and forward differencing, we expect that the error of less refined meshes to converge to the refined mesh on the order of $\mathcal{O}\left(h^2\right)$. To this end, we define the approximate error, $e_{i,j}$, as,

$$e_{i,j} = u^*_{i,j} - u_{i,j}.$$

The $L_2$ norm error may then be approximated as,

$$||e||_2 \approx \left( \sum_{j=1}^{M} \sum_{i=1}^{N} |e_{i,j}|^2 \, \Delta\xi\Delta\eta \right)^{1/2},$$

where $u^*_{i,j}$ is evaluated at every $k$ steps to make it the same size as $u_{i,j}$. For example, if the most refined mesh is $16 \times 16$, then it would be evaluated at every 4 steps to compare it to the solution on a $4 \times 4$ grid. In MATLAB syntax, this is related to the Frobenius ('Fro') norm of $e$ [3],

$$||e||_2 = ||e||_{Fro}\sqrt{\Delta\xi\Delta\eta},$$

which is coded by,

$$\text{e\_L2} = \text{norm}(\text{u\_star}(1:\text{k}:\text{end}, 1:\text{k}:\text{end}) - \text{u}, \text{"Fro"}) * \text{sqrt}(\text{dxi} * \text{deta}).$$

As an absolute error bound, the $L_\infty$ norm may be calculated by,

$$||e||_\infty = \max(|e_{i,j}|)$$

which is coded as,

$$\text{e\_inf} = \max(\text{abs}(\text{u\_star}(1:\text{k}:\text{end}, 1:\text{k}:\text{end}) - \text{u}), [\,], \text{"all"}).$$

The finest mesh tested was $(N, M) = (1024, 1024)$ [4] with $h =$ 9.77e-4. Successive meshes had $N$ and $M$ halved each time until the coarsest mesh with $(N, M) = (4, 4)$ was tested. These errors may be plotted on a log-log plot and compared to $\mathcal{O}(h)$ and $\mathcal{O}\left(h^2\right)$ which appear as straight lines.

[4] $1048576 \times 1048576$ A matrix, massive! This took ~37 s to solve on my laptop.

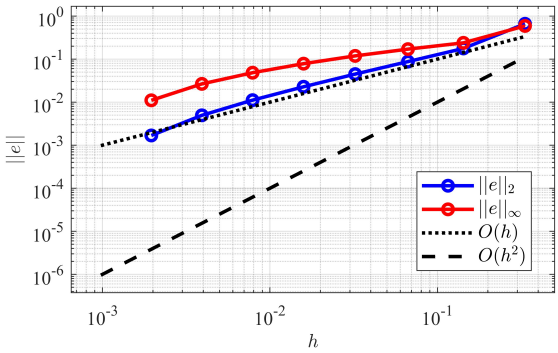



**Figure 8:** Mesh convergence test in the $L_2$ and $L_\infty$ norms

The plot suggests that the truncation error is *not* of order $h^2$, but rather of order $h$ or worse. This may be expected of the $L_\infty$ norm since the solution is discontinuous on the bottom boundary while the derivation of the second order truncation error assumes a continuous third derivative. Therefore, we cannot expect second order accuracy since this boundary error will dominate in the $L_\infty$ and $L_2$ norms. The $L_2$ norm encompasses more information of the solution over the entire domain which reflected with the errors generally being smaller than the $L_\infty$ errors. This implies that the error is smaller in the rest of the domain, away from the discontinuities and sharp changes.

# 9 Electrolocation Sensor

The electrolocation study may be performed by varying the angle, $\theta$, of the top wall relative to the bottom and varying the normal distance, $h_n$, of the wall from the source (at the origin).

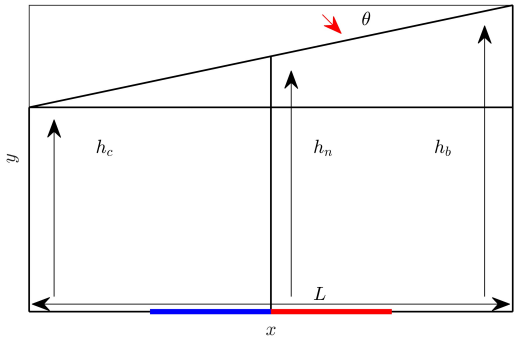



**Figure 9:** Diagram of proposed method to test the response of the electric field with varying wall distances and angles

For the electric field, we expect that there should be spike at the origin ('delta function') since loosely speaking, we are taking the derivative of a step function at the bottom boundary, which leads to a singular **E** field. We are interested in the change in **E** normal to the bottom boundary, thus, we need only measure the $E_y$ component. We may estimate the electric flux, $\Phi$, at the origin by taking the difference between the maximum and minimum values of $E_y$ along the bottom boundary [5]. We may then see how the boundary effects the flux by measuring $\Phi$ as we change $h_n$ and $\theta$.

[5] By the divergence theorem, we are essentially estimating the integral of $\nabla \cdot \mathbf{E}$ over a small volume around the origin (I could be wrong, I have never formally learned EM before!).

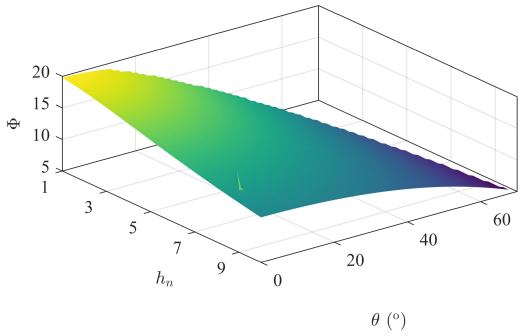
To this end, we must calculate $h_b$ and $h_c$ from $h_n$ and $\theta$. By trigonometry, the following relations hold,

$$\tan(\theta) = \frac{h_b - h_c}{L},$$

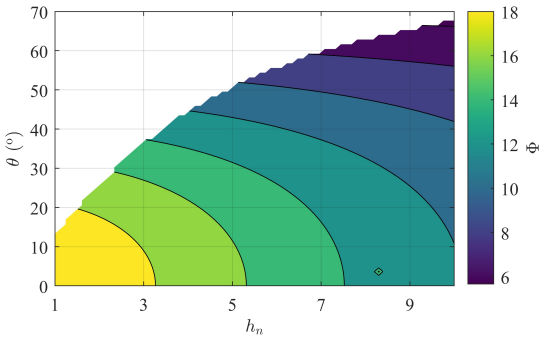
$$h_b = h_n + \frac{L}{2}\tan(\theta).$$

$$h_c = h_n - \frac{L}{2}\tan(\theta).$$

While the above is true, one must be careful that $h_c$ is not negative or else the bilinear mapping ceases to apply. With the above relations in mind, we may use a double for-loop to run the solver from the $h_b$ and $h_c$ values calculated from the desired $\theta$ and $h_n$ vectors. An if-statement should be used to check for a positive $h_c$ value and if the condition does not hold, then the stored value of $\Phi$ is set to NaN. If $h_c > 0$, then $\Phi$ is calculated from the minimum and maximum values of $E_y$ along the bottom. For a complete implementation of this code, refer to Appendix B, 'HW_1_electrolocation.m'. The test was performed using a $(N, M) = (64, 64)$ grid with $h_n$ varying from 1 to 10 and $\theta$ varying between 0 and 89 degrees. However, high angles often led to negative $h_c$ values and thus, $\theta$ did not exceed 70 degrees.




**(a)** Surface plot - observe the numerical anomaly at $(h_n, \theta) \sim (9, 5°)$




**(b)** Contour plot

**Figure 10:** Surface and contour plots of $\Phi$ at the origin with varying $\theta$ and $h_n$

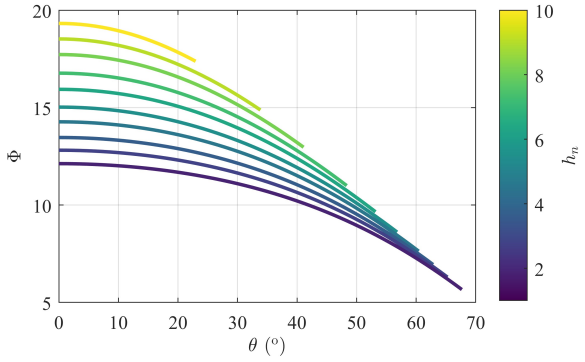



**Figure 11:** Surface projection plot of $\Phi$ on lines of constant $h_n$

From this, as $h_n$ increases, $\Phi$ decreases, which is logical since at small $h_n$ values, the electric field is almost directly reflected back into the source. For larger $h_n$ values, the electric field has more room to circulate, and thus, less of it returns normal to the bottom boundary (smaller $E_y$, larger $E_x$). This also explains why large $\theta$ values decrease $\Phi$ since large angles create a domain that is essentially triangular. This implies that the **E** vectors sharply turn at the $h_c$ vertex which causes a very large $E_x$ component and smaller $E_y$ component.

# References

[1] Saad, T., *The Finite Volume Method*. University of Utah, 2019.
https://www.youtube.com/watch?v=4n3DPwcoy4E&ab_channel=ProfessorSaadExplains

[2] https://www.mathworks.com/help/matlab/ref/spdiags.html

[3] https://www.mathworks.com/help/matlab/ref/norm.html

[4] Haberman, R., *Applied Partial Differential Equations with Fourier Series and Boundary Value Problems,* 3rd Ed., Prentice Hall, 1997.

## Appendix A - Functions

```
% ------------------------------------------------------------------
% Patrick Heng
% 7 Feb. 2025
% Function to generate bilinear map coefficients from a unit square
% to a general affine grid.
% Inputs: Xi, Eta: Meshgrid variables defined over the unit square
%         x, y: Vector of x and y coordinates of verticies of transformed
%               coordinates. Specified in the order (d,a,b,c)
%               c ----- b
%               |       |
%               |       |
%               d ----- a
% Outputs: X, Y: Meshgrid variables defined over the physical domain
%          coeffs_x, coeffs_y vectors of transformation coefficients
%          specified such that
%               x(xi,eta) = coeffs_x'*[xi.*eta; xi; eta; 1]
%          and similarly for y(xi,eta)
% ------------------------------------------------------------------

function [X,Y,coeffs_x,coeffs_y] = bilinear_map(Xi,Eta,x,y)

    % Make sure x, y are column vectors
    x = reshape(x,4,1);
    y = reshape(y,4,1);

    % Get size of mesh
    sz = size(Xi);
    N = numel(Xi);

    % Vectorize the mesh, calculate the points needed for mapping
    xi = reshape(Xi,N,1);
    eta =  reshape(Eta,N,1);
    xi_eta = reshape(Xi.*Eta,N,1);

    % Place vectorized mesh into matrix
    bilin =  [xi_eta, xi, eta, ones(N,1)]';

    % Transfer matrix to find mapping coefficients
    trans_mtx = [0,0,0,1; 0,1,0,1; 1,1,1,1; 0,0,1,1];

    % Solve for mapping coefficients
    coeffs_x = trans_mtx\x;
    coeffs_y = trans_mtx\y;

    % Calculate new coordinates
    coords(1,:) = coeffs_x'*bilin;
    coords(2,:) = coeffs_y'*bilin;

    % Return X and Y back as in the same style as meshgrid
    X = reshape(coords(1,:),sz(1),sz(2));
    Y = reshape(coords(2,:),sz(1),sz(2));
end
```

```
% ------------------------------------------------------------------
% Patrick Heng
% 7 Feb 2025
% 2D transformation coefficients for Laplacian and derivatives for a
% bilinear map. Used for finite differences.
% ------------------------------------------------------------------

function [x_xi,x_eta,y_xi,y_eta,x_xi_eta,...
            y_xi_eta,J,a,b,c,d,e,alpha,beta] = ...
            bilinear_map_derivatives(Xi,Eta,coeffs_x,coeffs_y)

    % Use coefficients as defined in class for better readability
    Rx = coeffs_x(1); Sx = coeffs_x(2); Tx = coeffs_x(3);
    Ry = coeffs_y(1); Sy = coeffs_y(2); Ty = coeffs_y(3);

    % Compute first derivatives of x,y over the entire mesh
    x_xi = Rx*Eta + Sx;
    x_eta = Rx*Xi + Tx;

    y_xi = Ry*Eta + Sy;
    y_eta = Ry*Xi + Ty;

    % Compute mixed derivatives of x,y over the entire mesh
    x_xi_eta = Rx;
    y_xi_eta = Ry;

    % Compute Jacobian determinant
    J = x_xi.*y_eta - x_eta.*y_xi;

    % Compute mapping coefficients for the Laplacian over the
    % entire domain (notation as defined in class)
    a = x_eta.^2 + y_eta.^2;
    b = x_eta.*x_xi + y_eta.*y_xi;
    c = x_xi.^2 + y_xi.^2;

    alpha = -2*b.*x_xi_eta;
    beta = -2*b.*y_xi_eta;

    e = (x_eta.*beta - y_eta.*alpha)./J;
    d = (y_xi.*alpha - x_xi.*beta)./J;

end
```

```
% ------------------------------------------------------------------
% Patrick Heng
% 7 Feb 2025
% 2D stencil coefficients for mapped Laplacian and derivatives for
% finite differences over a uniform computational domain.
% ------------------------------------------------------------------

function coeffs = stencil_coefficients(J,a,b,c,d,e,dxi,deta,nodes)

    % Compute stencil coefficients
    SW = b/(2*dxi*deta)./(J.^2);
    S = -(c/deta^2 - d/(2*deta))./(J.^2);
    SE = -SW;
    W = -(a/dxi^2 - e/(2*dxi))./(J.^2);
    P = 2*(a/dxi^2+c/deta^2)./(J.^2);
    E = -(a/dxi^2 + e/(2*dxi))./(J.^2);
    NW = -SW;
    N = -(c/deta^2 + d/(2*deta))./(J.^2);
    NE = SW;

    % Vectorize stencil coefficients, vertiacally stack rows
    % of contant eta
    SW = reshape(SW',nodes,1);
    S = reshape(S',nodes,1);
    SE = reshape(SE',nodes,1);
    W = reshape(W',nodes,1);
    P = reshape(P',nodes,1);
    E = reshape(E',nodes,1);
    NW = reshape(NW',nodes,1);
    N = reshape(N',nodes,1);
    NE = reshape(NE',nodes,1);

    % Return coefficients in convenient form for using spdiags
    coeffs = [SW,S,SE,W,P,E,NW,N,NE];
end

% ------------------------------------------------------------------
% Patrick Heng
% 7 Feb 2025
% 2D normal vectors for finite differences over a uniform computational
% domain. Returns normal vector to lines of constant xi given mapping
% derivatives and Jacobian.
% ------------------------------------------------------------------

function [N_xi_xi, N_xi_eta] = normals_xi(x_xi,x_eta,y_xi,y_eta,J)
    % Normalization factor for normal vector
    mag_xi = 1./sqrt(x_eta.^2+y_eta.^2);
    n_xi(:,1) = y_eta.*mag_xi;      % x component
    n_xi(:,2) = -x_eta.*mag_xi;     % y component

    % Normal derivative coefficients, notation defined in report
    N_xi_xi = (y_eta.*n_xi(:,1) - x_eta.*n_xi(:,2))./J;
    N_xi_eta = (-y_xi.*n_xi(:,1) + x_xi.*n_xi(:,2))./J;
end
```

```
% ------------------------------------------------------------------
% Patrick Heng
% 7 Feb 2025
% 2D normal vectors for finite differences over a uniform computational
% domain. Returns normal vector to lines of constant eta given mapping
% derivatives and Jacobian.
% ------------------------------------------------------------------

function [N_eta_xi, N_eta_eta] = normals_eta(x_xi,x_eta,y_xi,y_eta,J)
    % Normalization factor for normal vector
    mag_eta = 1./sqrt(x_xi.^2+y_xi.^2);
    n_eta(:,1) = -y_xi.*mag_eta;    % x component
    n_eta(:,2) = x_xi.*mag_eta;     % y component

    % Normal derivative coefficients, notation defined in report
    N_eta_xi = (y_eta.*n_eta(:,1) - x_eta.*n_eta(:,2))./J;
    N_eta_eta = (-y_xi.*n_eta(:,1) + x_xi.*n_eta(:,2))./J;
end

% ------------------------------------------------------------------
% Patrick Heng
% 8 Feb. 2025
% Function to impose boundary conditions for our specific HW problem.
% Neumann, No flux: Top, Left, Right, Bottom (partial)
% Dirichlet: Bottom over x = [-2,0] and [0,2] in the physcial domain
% Return modified A matrix, f vector
% ------------------------------------------------------------------

function [A,f] = impose_BCs(A,f,x_xi,x_eta,y_xi,y_eta,J,dxi,deta,X)

    % Get domain size
    n = numel(X(1,:));
    m = numel(X(:,1));
    nodes = n*m;

    % ----- NEUMANN CONDITIONS -----

    % --- TOP BOUNDARY ---

    % Compute transformed normal derivative coefficients
    % (notation defined in report)
    idx = NN(1:n,1,n);
    [N_eta_xi, N_eta_eta] = normals_eta(x_xi(idx), x_eta(idx), ...
        y_xi(idx), y_eta(idx), J(idx));

    % Central xi, foward eta derivative stencil and indexing
    diag = [-N_eta_xi/(2*dxi), -3*N_eta_eta/(2*deta), ...
        N_eta_xi/(2*dxi), 4*N_eta_eta/(2*deta), -N_eta_eta/(2*deta)];
    diag_idx = [-1, 0, 1, n, 2*n];

    % Clear rows of A corresponding to the boundary
    A(idx,:) = 0;

    % Apply stencil to A matrix, avoid domain corners
```

```
for i = 2:n-1
    A(NN(i,1,n),NN(i,1,n)+diag_idx) = diag(i,:);
end


% --- BOTTOM BOUNDARY ---

% Compute transformed normal derivative coefficients
idx = NN(1:n,m,n);

[N_eta_xi, N_eta_eta] = normals_eta(x_xi(idx), ...
    x_eta(idx), y_xi(idx), y_eta(idx), J(idx));

% Central xi, backward eta derivative stencil and indexing
diag = [N_eta_eta/(2*deta), -4*N_eta_eta/(2*deta), ...
    -N_eta_xi/(2*dxi), 3*N_eta_eta/(2*deta), N_eta_xi/(2*dxi)];
diag_idx = [-2*n, -n, -1, 0, 1];


% Clear rows of A corresponding to the boundary
A(idx,:) = 0;

% Apply stencil to A matrix, avoid domain corners
for i = 2:n-1
    A(NN(i,m,n),NN(i,m,n)+diag_idx) = diag(i,:);
end

% --- LEFT BOUNDARY ---

% Compute transformed normal derivative coefficients
idx = NN(1,1:m,n);
[N_xi_xi, N_xi_eta] = normals_xi(x_xi(idx), ...
    x_eta(idx),y_xi(idx),y_eta(idx), J(idx));

% Forward xi, central eta derivative stencil and indexing
diag = [-N_xi_eta/(2*deta), -3*N_xi_xi/(2*dxi), ...
    4*N_xi_xi/(2*dxi), -N_xi_xi/(2*dxi), N_xi_eta/(2*deta)];
diag_idx = [-n, 0, 1, 2, n];

% Clear rows of A corresponding to the boundary
A(NN(1,2:m-1,n),:) = 0;

% Apply stencil to A matrix, avoid domain corners
for j = 2:m-1
    A(NN(1,j,n),NN(1,j,n)+diag_idx) = diag(j,:);
end

% --- RIGHT BOUNDARY ---

% Compute transformed normal derivative coefficients
idx = NN(n,1:m,n);
[N_xi_xi, N_xi_eta] = normals_xi(x_xi(idx), ...
    x_eta(idx), y_xi(idx), y_eta(idx), J(idx));
```

```
% Backward xi, central eta derivative stencil and indexing
diag = [-N_xi_eta/(2*deta), N_xi_xi/(2*dxi), -4*N_xi_xi/(2*dxi), ...
    3*N_xi_xi/(2*dxi), N_xi_eta/(2*deta)];
diag_idx = [-n, -2, -1, 0, n];

% Clear rows of A corresponding to the boundary
A(NN(n,2:m-1,n),:) = 0;

% Apply stencil to A matrix, avoid domain corners
for j = 2:m-1
    A(NN(n,j,n),NN(n,j,n)+diag_idx) = diag(j,:);
end


% ----- CORNER POINTS -----
% Mix of foward and backward differencing to maintain second order
% accuracy. F = forwards, B = backwards

% --- TOP LEFT ---

% F xi, F eta
diag = [-3*N_xi_xi(1)/(2*dxi)+3*N_xi_eta(1)/(2*deta), ...
    4*N_xi_xi(1)/(2*dxi), -N_xi_xi(1)/(2*dxi), ...
    -4*N_xi_eta(1)/(2*deta), N_xi_eta(1)/(2*deta)];
diag_idx = [0,1,2,n,2*n];
A(NN(1,1,n),:) = 0;
A(NN(1,1,n), 1 + diag_idx) = diag;


% --- BOTTOM LEFT ---

% F xi, B eta
diag = 0.5*[(N_xi_eta(end)+N_eta_eta(1))/(2*deta), ...
        -4*(N_xi_eta(end)+N_eta_eta(1))/(2*deta), ...
        -3*(N_xi_xi(end)+N_eta_xi(1))/(2*dxi) + ...
         3*(N_xi_eta(end)+N_eta_eta(1))/(2*deta), ...
         4*(N_xi_xi(end)+N_eta_xi(1))/(2*dxi), ...
        -(N_xi_xi(end)+N_eta_xi(1))/(2*dxi)];
diag_idx = [-2*n,-n,0,1,2];
A(NN(1,m,n),:) = 0;
A(NN(1,m,n), NN(1,m,n) + diag_idx) = diag;


% --- TOP RIGHT ---

% B xi, F eta
diag = [N_xi_xi(1)/(2*dxi), -4*N_xi_xi(1)/(2*dxi), ...
    3*N_xi_xi(1)/(2*dxi)-3*N_xi_eta(1)/(2*deta), ...
    4*N_xi_eta(1)/(2*deta), -N_xi_eta(1)/(2*deta)];
diag_idx = [-2,-1,0,n,2*n];
A(NN(n,1,n),:) = 0;
A(NN(n,1,n), NN(n,1,n) + diag_idx) = diag;
```

```
    % --- BOTTOM RIGHT ---

    % B xi, B eta
    diag = 0.5*[(N_xi_eta(end)+N_eta_xi(end))/(2*deta), ...
            -4*(N_xi_eta(end)+N_eta_xi(end))/(2*deta), ...
             (N_xi_xi(end)+N_eta_xi(end))/(2*dxi), ...
            -4*(N_xi_xi(end)+N_eta_xi(end))/(2*dxi), ...
             3*(N_xi_xi(end)+N_eta_xi(end))/(2*dxi) + ...
             3*(N_xi_eta(end)+N_eta_xi(end))/(2*deta)];
    diag_idx = [-2*n,-n,-2,-1,0];
    A(NN(n,m,n),:) = 0;
    A(NN(n,m,n), NN(n,m,n) + diag_idx) = diag;


    % ----- DIRICHLET CONDITIONS ----
    % Find the indices of the physical domain where Dirichlet boundaries
    % should be imposed, use logical indexing

    indices = 1:n;

    % Left DBC
    dirich_idx_left = indices(-2 <= X(1,:) & X(1,:) <= 0);
    % Right DBC
    dirich_idx_right = indices(0 <= X(1,:) & X(1,:) <= 2);

    % Replace the rows of the Dirichlet nodes with the corresponding rows
    % of the identity matrix

    I = speye(nodes);
    A(NN(dirich_idx_left,m,n),:) = I(NN(dirich_idx_left,m,n),:);
    A(NN(dirich_idx_right,m,n),:) = I(NN(dirich_idx_right,m,n),:);

    % ----- FORCING FUCNTION -----
    % Neumann conditions for RHS
    f(NN(1:n,1,n)) = 0;     % Top
    f(NN(1:n,m,n)) = 0;     % Bottom
    f(NN(1,1:m,n)) = 0;     % Left
    f(NN(n,1:m,n)) = 0;     % Right

    % Impose Dirichlet conditions for the source
    f(NN(dirich_idx_left,m,n)) = -1;
    f(NN(dirich_idx_right,m,n)) = 1;


    % Node numbering function, i -> xi, j -> eta, n = total xi nodes
    function NN = NN(i,j,n)
        NN = n*(j-1) + i;
    end

end
```

```
% ------------------------------------------------------------------
% Patrick Heng
% 9 Feb. 2025
% Function to evaluate the gradient of a potential field, u, over a
% uniform computational grid given the mapping derivatives to physical
% domain (notation defined in report).
% ------------------------------------------------------------------

function [Ex,Ey] = grad_field(u,x_xi,x_eta,y_xi,y_eta,J,dxi,deta,n,m)

    % Get total number of nodes, n -> xi, m -> eta
    nodes = n*m;

    % Make sure u is a column vector
    u = flip(u,2);

    % C = Central difference, F = Forward difference,
    % B = Backward difference

    % ----- X COMPONENT OF GRAD FIELD -----

    % Generate the main stencil for Ex with central differencing
    diag = 0.5.*(J.^-1).*[y_xi/deta, -y_eta/dxi, y_eta/dxi, -y_xi/deta];
    diag_idx = [-n,-1,1,n];

    % Place the stencil coefficients on the appropriate diagonals
    A = spdiags(diag,diag_idx,nodes,nodes+1);
    A = A(:,1:nodes);

    % Generate the boundary stencils for Ex with mixed central, forward,
    % and backwards differencing

    % --- BOTTOM ---
    idx = NN(1:n,m,n);

    % xi - C, eta - B
    diag = 0.5.*(J(idx).^-1).*[-y_xi(idx)/deta,4*y_xi(idx)/deta, ...
        -y_eta(idx)/dxi,-3*y_xi(idx)/deta,y_eta(idx)/dxi];
    diag_idx = [-2*n,-n,-1,0,1];

    A(idx,:) = 0;   % Clear rows of A corresponding to the boundary

    % Fill the rows corresponding to the boundary
    for i = 2:n-1
        A(NN(i,m,n),NN(i,m,n)+diag_idx) = diag(i,:);
    end

    % --- TOP ---
    idx = NN(1:n,1,n);

    % xi - C, eta - F
    diag = 0.5*(J(idx).^-1).*[-y_eta(idx)/dxi,3*y_xi(idx)/deta, ...
        y_eta(idx)/dxi,-4*y_xi(idx)/deta,y_xi(idx)/deta];
    diag_idx = [-1,0,1,n,2*n];
```

```
A(idx,:) = 0;   % Clear rows of A corresponding to the boundary

% Fill the rows corresponding to the boundary
for i = 2:n-1
    A(NN(i,1,n),NN(i,1,n)+diag_idx) = diag(i,:);
end

% --- LEFT ---
idx = NN(1,1:m,n);

% xi - F, eta - C
diag = 0.5*(J(idx).^-1).*[y_xi(idx)/deta,-3*y_eta(idx)/dxi, ...
    4*y_eta(idx)/dxi,-y_eta(idx)/dxi,-y_xi(idx)/deta];
diag_idx = [-n,0,1,2,n];

A(idx,:) = 0;   % Clear rows of A corresponding to the boundary

% Fill the rows corresponding to the boundary
for i = 2:m-1
    A(NN(1,i,n),NN(1,i,n)+diag_idx) = diag(i,:);
end

% --- RIGHT ---
idx = NN(n,1:m,n);

% xi - B, eta - C
diag = 0.5*(J(idx).^-1).*[y_xi(idx)/deta,y_eta(idx)/dxi, ...
    -4*y_eta(idx)/dxi,3*y_eta(idx)/dxi,-y_xi(idx)/deta];

diag_idx = [-n,-2,-1,0,n];

A(idx,:) = 0;   % Clear rows of A corresponding to the boundary

% Fill the rows corresponding to the boundary
for i = 2:m-1
    A(NN(n,i,n),NN(n,i,n)+diag_idx) = diag(i,:);
end

% --- BOTTOM RIGHT ---
idx = NN(n,m,n);
% xi - B, eta - B
diag = 0.5*(J(idx).^-1).*[-y_xi(idx)/deta,4*y_xi(idx)/deta, ...
        y_eta(idx)/dxi,-4*y_eta(idx)/dxi, ...
        3*y_eta(idx)/dxi-3*y_xi(idx)/deta];
diag_idx = [-2*n,-n,-2,-1,0];

A(idx,:) = 0;   % Clear rows of A corresponding to the boundary
A(idx,idx+diag_idx) = diag;

% --- TOP RIGHT ---
idx = NN(n,1,n);
% xi - B, eta - F
diag = 0.5*(J(idx).^-1).*[y_eta(idx)/dxi,-4*y_eta(idx)/dxi, ...
```

```
        3*y_eta(idx)/dxi+3*y_xi(idx)/deta,-4*y_xi(idx)/deta, ...
        y_xi(idx)/deta];
diag_idx = [-2,-1,0,n,2*n];

A(idx,:) = 0;   % Clear rows of A corresponding to the boundary
A(idx,idx+diag_idx) = diag;

% --- BOTTOM LEFT ---
idx = NN(1,m,n);
% xi - F, eta - B
diag = 0.5*(J(idx).^-1).*[-y_xi(idx)/deta,4*y_xi(idx)/deta, ...
        -3*y_eta(idx)/dxi-3*y_xi(idx)/deta,4*y_eta(idx)/dxi, ...
        -y_eta(idx)/dxi];
diag_idx = [-2*n,-n,0,1,2];

A(idx,:) = 0;   % Clear rows of A corresponding to the boundary
A(idx,idx+diag_idx) = diag;

% --- TOP LEFT ---
idx = NN(1,1,n);
% xi - F, eta F
diag = 0.5*(J(idx).^-1).*[-3*y_eta(idx)/dxi+3*y_xi(idx)/deta, ...
        4*y_eta(idx)/dxi,-y_eta(idx)/dxi,-4*y_xi(idx)/deta, ...
        y_xi(idx)/deta];
diag_idx = [0,1,2,n,2*n];

A(idx,:) = 0;   % Clear rows of A corresponding to the boundary
A(idx,idx+diag_idx) = diag;


% Generate Ex by applying the finite difference matrix
Ex = A*u;

% ----- Y COMPONENT OF GRAD FIELD -----

% Generate the stencil for Ey with central differencing
diag = 0.5.*(J.^-1).*[-x_xi/deta, x_eta/dxi, -x_eta/dxi, x_xi/deta];
diag_idx = [-n,-1,1,n];

% Place the stencil coefficients on the appropriate diagonals
A = spdiags(diag,diag_idx,nodes,nodes+1);
A = A(:,1:nodes);

% Generate the boundary stencils for Ey with mixed central, forward,
% and backwards differencing

% --- BOTTOM ---
idx = NN(1:n,m,n);

% xi - C, eta - B
diag = -0.5.*(J(idx).^-1).*[-x_xi(idx)/deta,4*x_xi(idx)/deta, ...
        -x_eta(idx)/dxi,-3*x_xi(idx)/deta,x_eta(idx)/dxi];
diag_idx = [-2*n,-n,-1,0,1];
```

```
A(idx,:) = 0;   % Clear rows of A corresponding to the boundary

% Fill the rows corresponding to the boundary
for i = 2:n-1
    A(NN(i,m,n),NN(i,m,n)+diag_idx) = diag(i,:);
end

% --- TOP ---
idx = NN(1:n,1,n);

% xi - C, eta - F
diag = -0.5*(J(idx).^-1).*[-x_eta(idx)/dxi,3*x_xi(idx)/deta, ...
        x_eta(idx)/dxi,-4*x_xi(idx)/deta,x_xi(idx)/deta];
diag_idx = [-1,0,1,n,2*n];

A(idx,:) = 0;   % Clear rows of A corresponding to the boundary

% Fill the rows corresponding to the boundary
for i = 2:n-1
    A(NN(i,1,n),NN(i,1,n)+diag_idx) = diag(i,:);
end

% --- LEFT ---
idx = NN(1,1:m,n);

% xi - F, eta - C
diag = -0.5*(J(idx).^-1).*[x_xi(idx)/deta,-3*x_eta(idx)/dxi, ...
        4*x_eta(idx)/dxi,-x_eta(idx)/dxi,-x_xi(idx)/deta];
diag_idx = [-n,0,1,2,n];

A(idx,:) = 0;   % Clear rows of A corresponding to the boundary

% Fill the rows corresponding to the boundary
for i = 2:m-1
    A(NN(1,i,n),NN(1,i,n)+diag_idx) = diag(i,:);
end

% --- RIGHT ---
idx = NN(n,1:m,n);

% xi - B, eta - C
diag = -0.5*(J(idx).^-1).*[x_xi(idx)/deta,x_eta(idx)/dxi, ...
        -4*x_eta(idx)/dxi, 3*x_eta(idx)/dxi,-x_xi(idx)/deta];
diag_idx = [-n,-2,-1,0,n];

A(idx,:) = 0;   % Clear rows of A corresponding to the boundary

% Fill the rows corresponding to the boundary
for i = 2:m-1
    A(NN(n,i,n),NN(n,i,n)+diag_idx) = diag(i,:);
end

% --- BOTTOM RIGHT ---
idx = NN(n,m,n);
```

```
    % xi - B, eta - B
    diag = 0.5*(J(idx).^-1).*[-x_xi(idx)/deta,4*x_xi(idx)/deta, ...
            x_eta(idx)/dxi,-4*x_eta(idx)/dxi, ...
            3*x_eta(idx)/dxi-3*x_xi(idx)/deta];
    diag_idx = [-2*n,-n,-2,-1,0];

    A(idx,:) = 0;   % Clear rows of A corresponding to the boundary
    A(idx,idx+diag_idx) = diag;

    % --- TOP RIGHT ---
    idx = NN(n,1,n);
    % xi - B, eta - F
    diag = 0.5*(J(idx).^-1).*[x_eta(idx)/dxi,-4*x_eta(idx)/dxi, ...
            3*x_eta(idx)/dxi+3*x_xi(idx)/deta, -4*x_xi(idx)/deta, ...
            x_xi(idx)/deta];
    diag_idx = [-2,-1,0,n,2*n];

    A(idx,:) = 0;   % Clear rows of A corresponding to the boundary
    A(idx,idx+diag_idx) = diag;

    % --- BOTTOM LEFT ---
    idx = NN(1,m,n);
    % xi - F, eta - B
    diag = 0.5*(J(idx).^-1).*[-x_xi(idx)/deta,4*x_xi(idx)/deta, ...
            -3*x_eta(idx)/dxi-3*x_xi(idx)/deta,4*x_eta(idx)/dxi, ...
            -x_eta(idx)/dxi];
    diag_idx = [-2*n,-n,0,1,2];

    A(idx,:) = 0;   % Clear rows of A corresponding to the boundary
    A(idx,idx+diag_idx) = diag;

    % --- TOP LEFT ---
    idx = NN(1,1,n);
    % xi - F, eta - F
    diag = 0.5*(J(idx).^-1).*[-3*x_eta(idx)/dxi+3*x_xi(idx)/deta, ...
            4*x_eta(idx)/dxi,-x_eta(idx)/dxi, ...
            -4*x_xi(idx)/deta,x_xi(idx)/deta];
    diag_idx = [0,1,2,n,2*n];

    A(idx,:) = 0;   % Clear rows of A corresponding to the boundary
    A(idx,idx+diag_idx) = diag;

    % Generate Ey by applying the finite difference matrix
    Ey = -A*u;

    % Reshape gradient components to match meshgrid variables
    Ex = flip(reshape(Ex,n,m)');
    Ey = flip(reshape(Ey,n,m)');

    % Node numbering function, i -> xi, j -> eta, n = total xi nodes
    function NN = NN(i,j,n)
        NN = n*(j-1) + i;
    end
end
```

## Appendix B - Test Scripts

```
% ------------------------------------------------------------------------
% 'HW_1_test.m'
% Patrick Heng
% 23 Feb. 2025
% Test script to run the example from our homework. Essentially just
% calling all of the previously developed functions.
% ------------------------------------------------------------------------
close all; clear all; clc;

% Colormaps from '200 Colormaps' on the MATLAB file exchange
% SWITCH THIS TO A DEFAULT COLORMAP ('parula') IF YOU DO NOT HAVE THE
% FILE INSTALLED
cmap=slanCM(1);

% ------------------------------------------------------------------------
% ----- INPUTS -----
tic
% Forcing function as a function of the physical meshgrid variables,
% 0 for the Laplace equation
F = @(X,Y) 0*X;

% Define mesh shape
h_b = 3;
h_c = 2;

% Number of nodes in the computational domain, n -> x, m -> y
n = 128;
m = 128;
% ------------------------------------------------------------------------
% ----- SOLVER -----

% Total number of nodes
nodes = n*m;

% Generate computational mesh in xi, eta
xi = linspace(0,1,n);
eta = linspace(0,1,m);
[Xi,Eta] = meshgrid(xi,eta);

% Finite differences
dxi = xi(2) - xi(1);
deta = eta(2) - eta(1);

% Get bilinear map coefficients, generate physical mesh (X,Y)
[X,Y,coeffs_x,coeffs_y] = bilinear_map(Xi,Eta,[-4,4,4,-4],[0,0,h_b,h_c]);

% Compute derivatives of the coordinate transformation
[x_xi,x_eta,y_xi,y_eta,x_xi_eta,...
            y_xi_eta,J,a,b,c,d,e,alpha,beta] = ...
            bilinear_map_derivatives(Xi,Eta,coeffs_x,coeffs_y);

% Generate 9 point stencil for the main diagonals of the A matrix
```

```
% (discrete Laplacian)
diag = stencil_coefficients(J,a,b,c,d,e,dxi,deta,nodes);
diag_idx = [-n-1, -n, -n+1, -1, 0, 1, n-1, n , n+1];

% Compute the A matrix over the internal nodes. Initially generate the A
% matrix with one extra column to get the correct indexing for the
% stencil diagonals (see spdiags documentation)
A = spdiags(diag,diag_idx,nodes,nodes+1);

% Remove the extra column
A = A(:,1:nodes);

% Vectorize transformation derivatives
x_xi = reshape(x_xi',nodes,1);
x_eta = reshape(x_eta',nodes,1);
y_xi = reshape(y_xi',nodes,1);
y_eta = reshape(y_eta',nodes,1);
J = reshape(J',nodes,1);

% Impose boundary conditions on the A matrix, generate the corresponding
% forcing vector, f
f = reshape(F(X,Y)',nodes,1);

[A,f] = impose_BCs(A,f,x_xi,x_eta,y_xi,y_eta,J,dxi,deta,X);

% Linear solve for nodes
u = A\f;

% Generate the electric field by taking the negative gradient
% of the potential
[Ex,Ey] = grad_field(u,x_xi,x_eta,y_xi,y_eta,J,dxi,deta,n,m);
Ex = -Ex; Ey = -Ey;

% Magnitude of the E field
E = sqrt(Ex.^2+Ey.^2);

% Reshape u to match meshgrid variables
u = flip(reshape(u,n,m)');

toc
% ---------------------------------------------------------------------
% ----- PLOTTING -----

% Sparsity plot of A
figure
spy(A)

% Potential field solution in the physical domain
figure
surf(X,Y,u,linestyle='none')
colormap(cmap)
box on; grid on

xlabel('$x$', interpreter='latex'); ylabel('$y$', interpreter='latex');
```

```
zlabel('$\phi$', interpreter='latex');

fontsize(16,'points'); fontname('Serif');

% Contour plot of the potential field in the physical domain
figure
contourf(X,Y,u,20,FaceAlpha=0.95)
colormap(cmap)

cb = colorbar();
xlabel('$x$', interpreter='latex'); ylabel('$y$', interpreter='latex');
ylabel(cb,'$\phi$',interpreter='latex')
clim([-1,1])
box on

fontsize(16,'points'); fontname('Serif');

% Surface plot of the magnitude of the electric field
figure
colormap(cmap)
surf(X,Y,E,linestyle ='none')

cb = colorbar();
xlabel('$x$', interpreter='latex'); ylabel('$y$', interpreter='latex');
ylabel(cb,'$|E|$',interpreter='latex')
box on; grid on

fontsize(16,'points'); fontname('Serif');

% Contour plot of potential field
figure
hold on
sc = 3;        % Arrow density for quiver plot, higher sc = less arrows
colormap(cmap)

% Plot normalized E vector field with set arrow density
quiver(X(1:sc:end,1:sc:end),Y(1:sc:end,1:sc:end), ...
        Ex(1:sc:end,1:sc:end)./E(1:sc:end,1:sc:end), ...
        Ey(1:sc:end,1:sc:end)./E(1:sc:end,1:sc:end),color='k', ...
        linewidth=0.3)

% Plot 30 equipotential lines
levels = linspace(min(u,[],'all'),max(u,[],'all'),30);
contour(X,Y,u,levels,linewidth=0.8)
xlim([-4,4]); ylim([0,max([h_b,h_c])]);
xlabel('$x$', interpreter='latex'); ylabel('$y$', interpreter='latex');

cb = colorbar;
ylabel(cb,'$\phi$',interpreter='latex')

box on; grid on
fontsize(16,'points'); fontname('Serif');

% Surface plot projected on lines of constant x
```

```
figure
colormap(cmap);
levels = 32;    % Number of slices to plot
x = X(1,:);

for i = 1:levels
    k = floor(i*size(x,2)/levels);
    plot(x,u(k,:),color=flip(cmap(floor(256*i/levels),:),1),linewidth=2)
    hold on
end

xlabel('$x$',interpreter='latex'); ylabel('$\phi$',interpreter='latex')

cb = colorbar;
clim([0,max([h_b,h_c])])
ylabel(cb,'$y$',interpreter='latex')

box on; grid on
fontsize(16,'points'); fontname('Serif');

% Surface plot projected on lines of constant y
figure
levels = 32;    % Number of slices to plot
colormap(cmap);

for i = 1:levels
    k = floor(i*size(Y(:,1),1)/levels);
    y = Y(:,k);
    plot(y,u(:,k),color=cmap(floor(256*i/levels),:),linewidth=2)
    hold on
end

xlabel('$y$',interpreter='latex'); ylabel('$\phi$',interpreter='latex')

cb = colorbar;
clim([-4,4])
ylabel(cb,'$x$',interpreter='latex')

box on; grid on
fontsize(16,'points'); fontname('Serif');

% ------------------------------------------------------------------
% IF YOU DESIRE TO SAVE THE SOLUTION: UNCOMMENT, RENAME
%save('Refined_mesh_1024_1024.mat','X','Y','Xi','Eta','u','Ex','Ey','E')

% ------------------------------------------------------------------
% 'HW_1_Err_Conv.m'
% Patrick Heng
% 23 Feb. 2025
% Test script to run error convergence from our homework. Essentially
% just calling all of the previously developed functions in a BIG loop.
% ------------------------------------------------------------------
close all; clear all; clc;
```

```
% ---------------------------------------------------------------------
% ----- INPUTS -----
tic
% Forcing function as a function of the physical meshgrid variables,
% 0 for the Laplace equation
F = @(X,Y) 0*X;

% Define mesh shape
h_b = 3;
h_c = 2;

% ---------------------------------------------------------------------
% ----- SOLVER -----
% BIG loop to save mesh data to study error convergence.
% Grab a coffee, this may take a while to run unless you have a super
% computer
for n = 2.^(2:10)

    % Number of nodes in the computational domain, n -> x, m -> y
    m = n;

    % Total number of nodes
    nodes = n*m;

    % Generate computational mesh in xi, eta
    xi = linspace(0,1,n);
    eta = linspace(0,1,m);
    [Xi,Eta] = meshgrid(xi,eta);

    % Finite differences
    dxi = xi(2) - xi(1);
    deta = eta(2) - eta(1);

    % Get bilinear map coefficients, generate physical mesh (X,Y)
    [X,Y,coeffs_x,coeffs_y] = bilinear_map(Xi,Eta,[-4,4,4,-4], ...
                                [0,0,h_b,h_c]);

    % Compute derivatives of the coordinate transformation
    [x_xi,x_eta,y_xi,y_eta,x_xi_eta,...
                y_xi_eta,J,a,b,c,d,e,alpha,beta] = ...
                bilinear_map_derivatives(Eta,Xi,coeffs_x,coeffs_y);

    % Generate 9 point stencil for the main diagonals of the A matrix
    % (discrete Laplacian)
    diag = stencil_coefficients(J,a,b,c,d,e,dxi,deta,nodes);
    diag_idx = [-n-1, -n, -n+1, -1, 0, 1, n-1,n , n+1];

    % Compute the A matrix over the internal nodes. Initially generate the
    % A matrix with one extra column to get the correct indexing for the
    % stencil diagonals (see spdiags documentation)
    A = spdiags(diag,diag_idx,nodes,nodes+1);

    % Remove the extra column
    A = A(:,1:nodes);
```

```
    % Vectorize transformation derivatives
    x_xi = reshape(x_xi',nodes,1);
    x_eta = reshape(x_eta',nodes,1);
    y_xi = reshape(y_xi',nodes,1);
    y_eta = reshape(y_eta',nodes,1);
    J = reshape(J',nodes,1);

    % Impose boundary conditions on the A matrix, generate the
    % corresponding forcing vector, f
    f = reshape(F(X,Y),nodes,1);

    [A,f] = impose_BCs(A,f,x_xi,x_eta,y_xi,y_eta,J,dxi,deta,X);

    % Linear solve for nodes
    u = A\f;

    % Reshape u to match meshgrid variables
    u = flip(reshape(u,n,m)');

    % Save the physical mesh and solution
    save(['u_mesh_' num2str(n) '_' num2str(m) '.mat'],'X','Y','u')

end
toc

% ------------------------------------------------------------------------
% 'HW_1_Err_Conv_Plots.m'
% Patrick Heng
% 23 Feb. 2025
% Test script plot error convergence from our homework. MUST run
% 'HW_1_Err_Conv.m' (previous script) first to generate the data.
% ------------------------------------------------------------------------
close all; clc;

% Maximum exponent of the refined mesh (h = 2^k_max)
k_max = 10;

% Load maximum refined mesh and store for comparison
temp = load('u_mesh_1024_1024.mat','u');
u_star = temp.u;

% Preallocate error vectors
e_L2 = zeros(k_max-1,1);
e_inf = zeros(k_max-1,1);

% Loop through meshes, starting with the most refined and moving to
% the coarsest mesh
i = 1;
N = 2.^(k_max:-1:2);

for n = N
    % Load the coarser mesh
    temp = load(['u_mesh_' num2str(n) '_' num2str(n) '.mat']);
```

```
    u = temp.u;

    % Calculate the error by subtracting the refined mesh evaluated at
    % the coarse mesh grid points. Since the refinement halves each time
    % this means evaluating the u_star at every 'step'.
    % (I think something might be wrong here)
    step = 2^(i-1);
    e = u_star(1:step:end,1:step:end)-u;
    e = e(1:end,:);

    % Calculate normalized L2 error over the domain
    e_L2(i) = norm(e,'fro')*(1/(n-1));
    % Calculate the absolute maximum error
    e_inf(i) = max(e,[],'all');

    % Increment loop counter
    i = i + 1;

end

% Plot error on a log-log plot, compare to O(h) and O(h^2) growth
figure
% L2 error
loglog((N-1).^-1, e_L2,marker='o',color='b',linewidth=2)

hold on
% Maximum error
loglog((N-1).^-1, e_inf,marker='o',color='r',linewidth=2)

% O(h)
loglog((N-1).^-1,(N-1).^-1,linestyle=':',color='k',linewidth=2)
% O(h^2)
loglog((N-1).^-1,(N-1).^-2,linestyle='--',color='k',linewidth=2)

% Pretty plot parameters
grid on
box on
axis padded
yticks(10.^(-6:1:0))
xlabel('$h$',interpreter='latex')
ylabel('$||e||$',interpreter='latex')

legend('$||e||_{2}$', '$||e||_{\infty}$', '$O(h)$', '$O(h^2)$',
   interpreter='latex', location='best')
fontname('Serif'); fontsize(16,'points');

% ------------------------------------------------------------------
% 'HW_1_electrolocation.m'
% Patrick Heng
% 23 Feb. 2025
% Test script to run the electrolocation study. Essentially just
% calling all of the previously developed functions in a BIG loop.
% ------------------------------------------------------------------
close all; clear all; clc;
```

```
% Generate the normal distances and angles to test
H_n = linspace(1,10,75);
theta = linspace(0,pi/2-0.01,75);

[HN,THETA] = meshgrid(H_n,theta);

% Preallocate min/max values of E field
Ey_max = NaN(size(H_n,2),size(theta,2));
Ey_min = NaN(size(H_n,2),size(theta,2));

% ------------------------------------------------------------------
% ----- INPUTS -----
tic
% Forcing function as a function of the physical meshgrid variables,
% 0 for the Laplace equation
F = @(X,Y) 0*X;
% Number of nodes in the computational domain, n -> x, m -> y
n = 64;
m = 64;

for i = 1:size(H_n,2)
    for j = 1:size(theta,2)
        % Define mesh shape
        h_b = H_n(i) + 4*tan(theta(j));
        h_c = H_n(i) - 4*tan(theta(j));
        if h_c < 0
            Ey_max(i,j) = NaN;
            break
        end


        % ----------------------------------------------------------
        % ----- SOLVER -----

        % Total number of nodes
        nodes = n*m;

        % Generate computational mesh in xi, eta
        xi = linspace(0,1,n);
        eta = linspace(0,1,m);
        [Xi,Eta] = meshgrid(xi,eta);

        % Finite differences
        dxi = xi(2) - xi(1);
        deta = eta(2) - eta(1);

        % Get bilinear map coefficients, generate physical mesh (X,Y)
        [X,Y,coeffs_x,coeffs_y] = bilinear_map(Xi,Eta, ...
                                               [-4,4,4,-4],[0,0,h_b,h_c]);

        % Compute derivatives of the coordinate transformation
        [x_xi,x_eta,y_xi,y_eta,x_xi_eta,...
                    y_xi_eta,J,a,b,c,d,e,alpha,beta] = ...
```

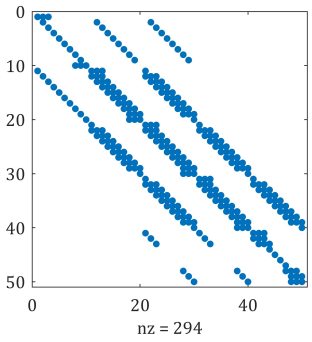
```
                    bilinear_map_derivatives(Xi,Eta,coeffs_x,coeffs_y);

        % Generate 9 point stencil for the main diagonals of the A matrix
        % (discrete Laplacian)
        diag = stencil_coefficients(J,a,b,c,d,e,dxi,deta,nodes);
        diag_idx = [-n-1, -n, -n+1, -1, 0, 1, n-1, n , n+1];

        % Compute the A matrix over the internal nodes. Initially
        % generate the A matrix with one extra column to get the correct
        % indexing for the stencil diagonals (see spdiags documentation)
        A = spdiags(diag,diag_idx,nodes,nodes+1);

        % Remove the extra column
        A = A(:,1:nodes);

        % Vectorize transformation derivatives
        x_xi = reshape(x_xi',nodes,1);
        x_eta = reshape(x_eta',nodes,1);
        y_xi = reshape(y_xi',nodes,1);
        y_eta = reshape(y_eta',nodes,1);
        J = reshape(J',nodes,1);

        % Impose boundary conditions on the A matrix, generate
        % the corresponding forcing vector, f
        f = reshape(F(X,Y)',nodes,1);

        [A,f] = impose_BCs(A,f,x_xi,x_eta,y_xi,y_eta,J,dxi,deta,X);

        % Linear solve for nodes
        u = A\f;

        % Generate the electric field by taking the negative gradient
        % of the potential
        [Ex,Ey] = grad_field(u,x_xi,x_eta,y_xi,y_eta,J,dxi,deta,n,m);
        Ex = -Ex; Ey = -Ey;

        Ey_max(i,j) = max(Ey(1,:),[],'all');
        Ey_min(i,j) = min(Ey(1,:),[],'all');
    end
end
toc

dEy = Ey_max-Ey_min;

surf(HN,THETA*180/pi,dEy',linestyle='none')

% ------------------------------------------------------------------------
% IF YOU DESIRE TO SAVE THE SOLUTION: UNCOMMENT, RENAME
%save('Electrolocation_study_data.mat','HN','THETA','dEy',
%       ... 'Ey_max','Ey_min')
```

```
% ------------------------------------------------------------------
% 'HW_1_electrolocation_plots.m'
% Patrick Heng
% 24 Feb. 2025
% Test script to PLOT data for numerical electrolocation case study.
% MUST run 'HW_1_electrolocation.m' script beforehand to generate the
% data.
% ------------------------------------------------------------------

% Colormaps from '200 Colormaps' on the MATLAB file exchange
% SWITCH THIS TO A DEFAULT COLORMAP ('parula') IF YOU DO NOT HAVE THE
% FILE INSTALLED
cmap=slanCM(1);

load('Electrolocation_study_data.mat')

DEG = THETA*180/pi;

% Contour plot
figure

colormap(cmap)
contourf(HN,DEG,dEy')
xlim([1,10])
xticks(1:2:10)
ylim([0,70])

xlabel('$h_n$', interpreter='latex')
ylabel('$\theta\ (^\mathrm{o})$', interpreter='latex')

cb = colorbar;
ylabel(cb,'$\Phi$',interpreter='latex')

box on; grid on
fontsize(16,'points'); fontname('Serif');

% 3D surface plot
figure

colormap(cmap)
surf(HN,DEG,dEy',linestyle='none')
xlim([1,10])
xticks(1:2:10)
ylim([0,70])

xlabel('$h_n$', interpreter='latex')
ylabel('$\theta\ (^\mathrm{o})$',interpreter='latex')
zlabel('$\Phi$', interpreter='latex')

box on; grid on
fontsize(16,'points'); fontname('Serif');
```

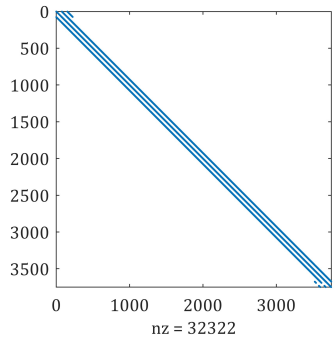
```
% Surface plot projected on lines of constant x
figure
colormap(cmap);
levels = 10;    % Number of slices to plot
hn = HN(1,:);

for i = 1:levels
    k = floor(i*size(hn,2)/levels);
    plot(hn,dEy(:,k),color=cmap(floor(256*(levels-i+1)/levels),:), ...
        linewidth=2)
    hold on
end

xlim([1,10])

xlabel('$h_n$',interpreter='latex')
ylabel('$\Phi$',interpreter='latex')

cb = colorbar;
clim([0,70])
ylabel(cb,'$\theta\ (^\mathrm{o})$',interpreter='latex')

box on; grid on
fontsize(16,'points'); fontname('Serif');

% Surface plot projected on lines of constant y
figure
levels = 10;    % Number of slices to plot
colormap(cmap);
deg = DEG(:,1);

for i = 1:levels
    k = floor(i*size(deg,1)/levels);
    plot(deg,dEy(k,:),color=cmap(floor(256*(levels-i+1)/levels),:),
       linewidth=2)
    hold on
end

xlim([0,70])

xlabel('$\theta\ (^\mathrm{o})$',interpreter='latex')
ylabel('$\Phi$',interpreter='latex')

cb = colorbar;
ylabel(cb,'$h_n$',interpreter='latex')
clim([1,10])

box on; grid on
fontsize(16,'points'); fontname('Serif');
```

# Appendix C - Extra

This appendix consists of extra codes and figures for future reference for myself. This section may be skipped for grading purposes.




**(a)** $(N, M) = (10, 5)$




**(b)** $(N, M) = (75, 50)$

**Figure 12:** Sparsity patterns of the A matrix. The 9 point stencil leads to a 9 main bands on the diagonals centered about the main diagonal. There are also varying bands imposed by the boundary conditions due to the forward and backward differencing.

```
% ------------------------------------------------------------------------
% Patrick Heng
% 21 Feb. 2025
% Script to generate diagram for 9-point stencil coefficients.
% ------------------------------------------------------------------------
close all; clear all; clc;

% Generate 3x3 grid
x = 1:3;
y = 1:3;
h = 0.1;
[X,Y] = meshgrid(x,y);

% Plot nodal points and gridlines
hold on
plot(X,Y,marker='.',markersize=30,color='k')
affine_grid(X,Y);

% Add labels to the nodal points
% Top
text(1+h,3-h,texlabel('$u_{i-1,\ j+1}: NW_{i,j}$'),interpreter='latex')
text(2+h,3-h,texlabel('$u_{i,\ j+1}: N_{i,j}$'),interpreter='latex')
text(3+h,3-h,texlabel('$u_{i+1,\ j+1}: NE_{i,j}$'),interpreter='latex')

% Center
text(1+h,2-h,texlabel('$u_{i-1,\ j}: W_{i,j}$'),interpreter='latex')
text(2+h,2-h,texlabel('$u_{i,j}: P_{i,j}$'),interpreter='latex')
text(3+h,2-h,texlabel('$u_{i+1,\ j}: E_{i,j}$'),interpreter='latex')

% Bottom
text(1+h,1-h,texlabel('$u_{i-1,\ j-1}: SW_{i,j}$'),interpreter='latex')
```

```
text(2+h,1-h,texlabel('$u_{i,\ j-1}: S_{i,j}$'),interpreter='latex')
text(3+h,1-h,texlabel('$u_{i+1,\ j-1}: SE_{i,j}$'),interpreter='latex')

% Pretty plot parameters
pos = get(gca, 'Position');
pos(1) = 0.05; pos(3) = 0.6;
set(gca, 'Position', pos)
fontsize(16,'points')

xlabel(''); ylabel(''); xticks([]); yticks([])

% Plot gridlines for transformed coordinates
function affine_grid(X,Y)

    % Horizontal gridlines
    for i = 1:numel(X(1,:))
        line([X(1,1),X(1,end)],[Y(i,1),Y(i,end)],color='k')
    end

    % Vertical gridlines
    for i = 1:numel(Y(1,:))
        line([X(1,i),X(end,i)],[Y(1,i),Y(end,i)],color='k')
    end

end

% ------------------------------------------------------------------------
% Patrick Heng
% 24 Feb. 2025
% Script to plot diagram for electrolocation case study.
% ------------------------------------------------------------------------

close all; clear all; clc;

% Normal distance and angle of top boundary
theta = atan2(2,8);
h_n = 2 + 2*tan(theta);

figure
hold on

% Plot domain
line([-4,-4],[0,2], linewidth=1,color='k')
line([4,4],[0,3], linewidth=1,color='k')
line([-4,4],[0,0], linewidth=1,color='k')
line([-4,4],[2,3], linewidth=1,color='k')
line([-4,4],[2,2], linewidth=1,color='k')
line([0,0],[0,h_n], linewidth=1,color='k')

% Source
line([-2,0],[0,0],linewidth=3,color='b')
line([0,2],[0,0],linewidth=3,color='r')

% Varical arrows
```

```
text(1.5,2.85,'$\theta$',interpreter='latex')
text(0.7,1.6,'$h_n$',interpreter='latex')
text(-2.9,1.6,'$h_c$',interpreter='latex')
text(2.7,1.6,'$h_b$',interpreter='latex')
annotation('arrow',[0.6 0.63],[0.9 0.85],colo='r')
annotation('arrow',[.55 .55],[0.15 .75])
annotation('arrow',[.17 .17],[0.15 .62])
annotation('arrow',[.86 .86],[0.15 .87])

% Bottom arrow
text(0.7,0.16,'$L$',interpreter='latex')
annotation('arrow',[0.2 0.9],[0.13 0.13])
annotation('arrow',[0.9 0.135],[0.13 0.13])


xticks([]); yticks([])
xlabel('$x$',interpreter='latex'); ylabel('$y$',interpreter='latex')
fontsize(16,'points'); fontname('Serif'); box on; grid on;
```