Queen's UNIVERSITY | FACULTY OF ENGINEERING AND APPLIED SCIENCE

Lab 5: Partial Differential Equations

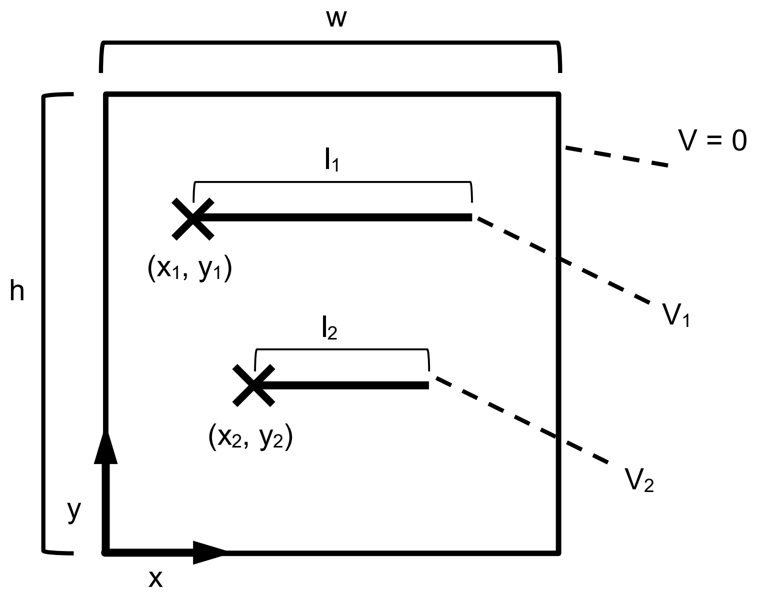
The objective of this laboratory is to give you some practice writing code to solve partial differential equations and also working with different boundary conditions. The problem that you will solve in this lab is Laplace's equation ($\nabla^2 V = 0$), so that given a fixed *V* at set points (boundary conditions), *V* can be found at all points.

Before You Start

Remember that MATLAB can only use files, functions, and data within the current directory. **Make sure** you make a new folder for each lab and that it is sest as your current folder before you start.

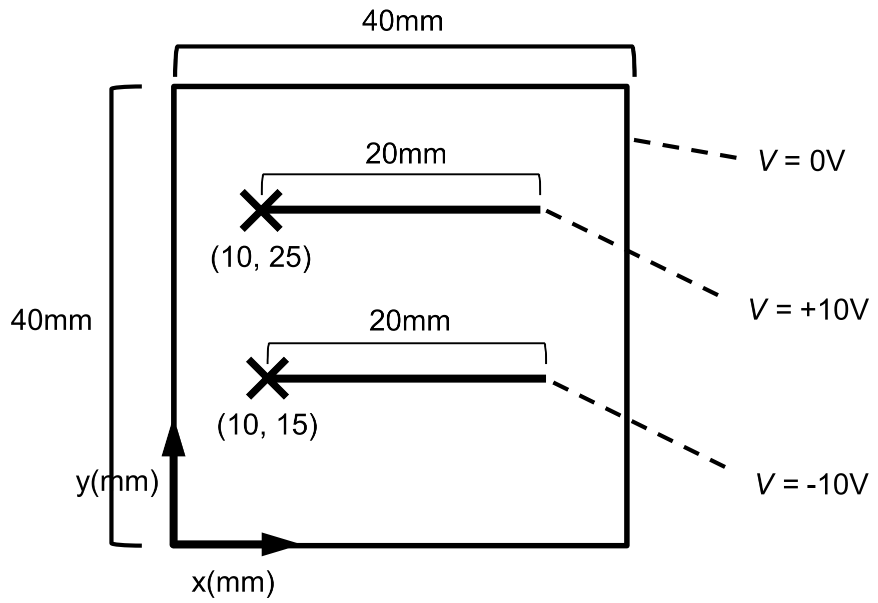
Part A – Due at end of the lab period

The geometry of the problem you will solve consists of a box with width, *w,* and height, *h*. You can imagine the bottom-left corner of the box being at the origin *(0, 0)*. The box is grounded, so its walls are at *V=0*. Within the box are two plates, each are horizontal (|| to the x-axis) and are described by a position *(x, y)* of the left-end of the plate, and a plate length, *l.* Each plate can be at a different voltage *V.* All units are in mm. This geometry is shown below:



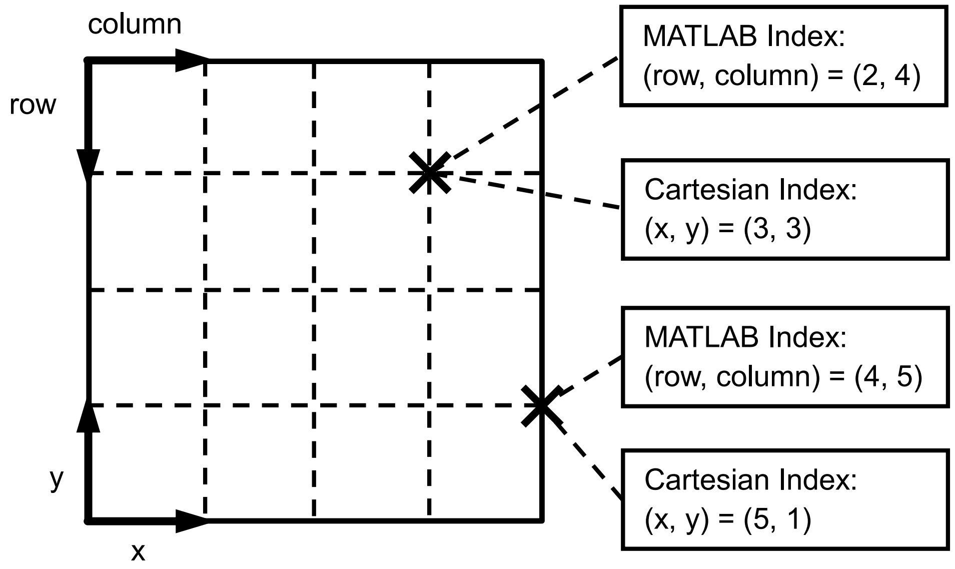
Your task will be to take a given geometry (box size, plate positions and lengths) and plate voltages, and produce a map of V for a grid of points within the box (including the edge). The spacing of the points to find V for will be 1mm and the plates and box edges can be assumed to be truly two-dimensional and to be placed at locations that correspond to points on the grid (e.g. they will have integer positions and lengths.

The testing files you will work with will use the following geometry, though your code must work with general geometries (this will be tested for in the marking!).



Notes:

- For the testing function geometry, your resulting map of V will be a 41 x 41 grid of points, since the box is 40mm x 40mm, each side would have 41 points along it.
- Remember:
  - Indexing of matrices in MATLAB starts at 1
  - MATLAB matrix indexing is (row, column), so x and y coordinates will be swapped
  - MATLAB matrix indexing starts at the top-left corner, while the box's coordinates start at the bottom-left, so y coordinates are opposite (see 5 x 5 example below)



Task A1

When you write your code to solve Laplace's Equation, you will need to set (and reset) your boundary conditions. Write a function `setBoxAndPlateVoltages` with the following inputs/output in the following order:

INPUTS	
`V`	Map of voltages at grid of points
`topPlateX`	x-coord, top plate's leftmost point
`topPlateY`	y-coord, top plate's leftmost point
`topPlateL`	Length of the top plate
`topPlateV`	The top plate's voltage
`botPlateX`	x-coord, bottom plate's leftmost point
`botPlateY`	y-coord, bottom plate's leftmost point
`botPlateL`	Length of the bottom plate
`botPlateV`	The bottom plate's voltage

OUTPUT	
V	Map of voltages at grid of points with the box edges points set to *V=0* and the plates' points set to their respective voltages. All other points are left untouched!

A testing function, **test_setBoxAndPlateVoltages.m,** is available on onQ. Make sure **checkEqual.m** is also downloaded into your current directory.

Task A2

Now write a function `solveLaplace` that solves Laplace's Equation for the geometry/voltages given as input, and produces a map of voltages at points with a 1mm spacing. The inputs/output should be as follows:

INPUTS	
`boxWidth`	The width of the box
`boxHeight`	The height of the box
`topPlateX`	x-coord, top plate's leftmost point
`topPlateY`	y-coord, top plate's leftmost point
`topPlateL`	Length of the top plate
`topPlateV`	The top plate's voltage
`botPlateX`	x-coord, bottom plate's leftmost point
`botPlateY`	y-coord, bottom plate's leftmost point
`botPlateL`	Length of the bottom plate
`botPlateV`	The bottom plate's voltage

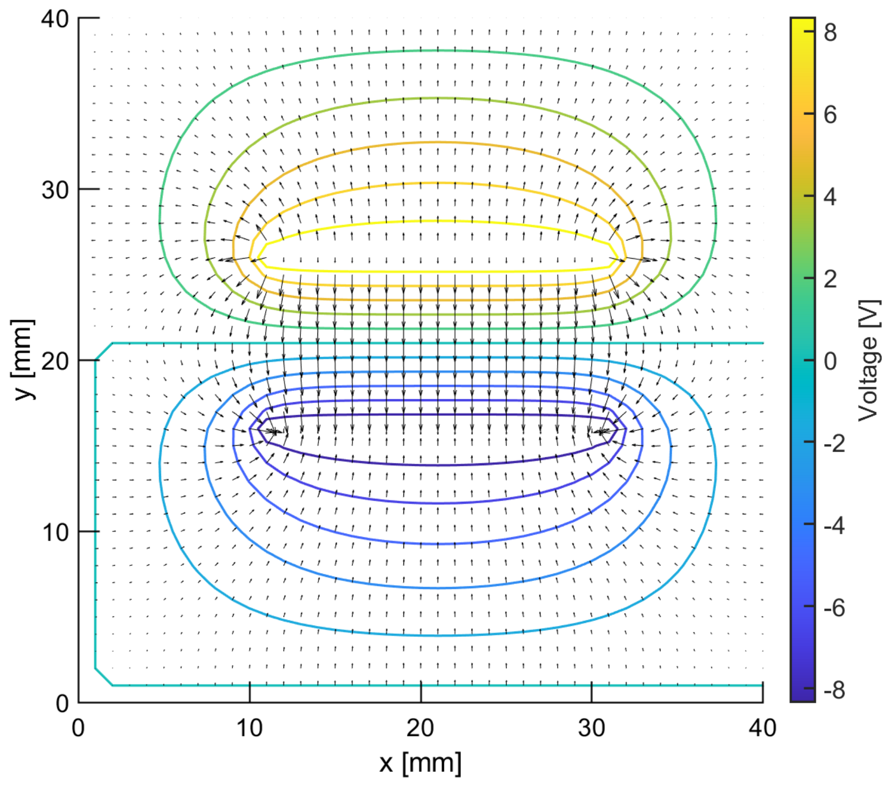
OUTPUT	
V	Map of voltages at grid of points after solving Laplace's Equation for the inputted geometry/voltages.

To solve Laplace's Equation, your code should use an iterative method, where you set the number of iterations yourself after experimenting to find a value that works. Make sure you submit your code with the correct number of iterations saved. Be careful that at the beginning of each iteration the boundary conditions in your map of *V* are properly set, and use your function from Task A1 to reset them if needed. Also be careful that the new values of *V* found by a single iteration is only using values of *V* from the previous iteration and the boundary conditions.
The testing file **test_solveLaplace.m** is given on onQ, though using the function from Task A3 below may help in graphically displaying the results of your function as you develop it.

Task A3

Write a function `plotElectricField` that takes the map of *V* solution from `solveLaplace` and outputs a plot of the solution. The plot should include contours of the *V* values (see the MATLAB `contour` function), as well as a vector field of the electric field, $E = -\nabla V$ (see the MATLAB `gradient` and `quiver` functions).

If you use the provided **test_plotElectricField.m** file, it will first use your `solveLaplace` function from Task A2, and then plot the result of the same geometry. Another file **test_plotElectricField_setData.m** is given that can also be used for testing, but it uses a stored map of *V*, instead of calling your `solveLaplace` function. If everything is working well and you choose to plot ~10 contours, the result should look similar to this:



Part A: Submission List

When complete, please submit the following files to the **Lab 5: Part A** dropbox onQ for marking:

- setBoxAndPlateVoltages.m
- solveLaplace.m
- plotElectricField.m
- Any other functions you wrote that the above files use

No Part B this week!