Patrick Halter
PHYS 613
HW3

Report on Homework 3: Elliptic Partial Differential Equations

Précis

I created an algorithm is capable of solving to second order all boundary value problems for constant solutions to Poisson's Equation in two Cartesian dimensions, assuming a rectilinear problem space and constant x and y step size. Step sizes, problem domain, and boundaries can be completely specified in the included problem.cfg file. For ease of discussion, the rest of this document assumes the special case solution to Poisson's Equation:

$$\nabla^2 = 0$$

otherwise known as Laplace's equation. (All illustrations are for step size = 0.01, unless noted

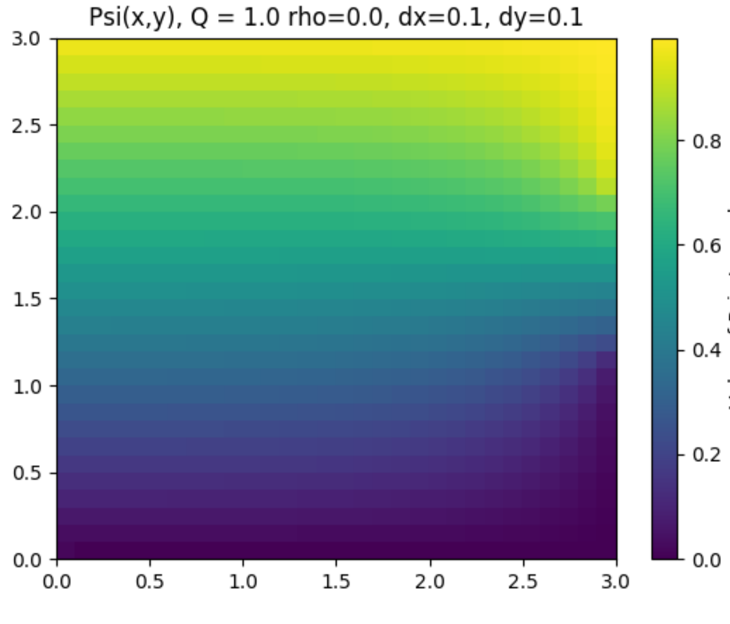

Illustration 1: Colormap of psi for default config

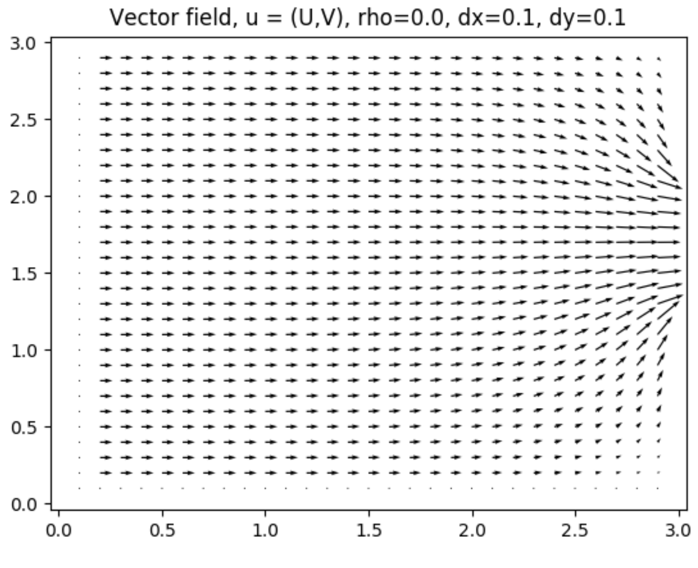

Illustration 2: Vector map of Psi for default config

Patrick Halter
PHYS 613
HW3

Overview of Algorithm

The algorithm proceeds in N parts.

The first portion of the algorithm, specified in buildCartStencil() in libs/setup_2d_PDE.py sets up the matrix equation AΨ = b that corresponds to the 2nd order stencil for the specified conditions. Neumann boundary conditions are evaluated to first order, which is less accurate but makes no presumptions on the size of the matrix. This matrix A is dense, meaning that step sizes and domains should be wisely chosen not to overstress memory or computational speed. (For a machine with 8 gigabytes of DDR4 RAM, and domains on the order of 1e2, minimum recommended step sizes are on the order of 1e-1.) It does so from left to right (increasing X), wrapping around bottom to top (so that when an end of the X's for a certain 'row' is reached, another 'row' corresponding to a step up in Y – where these rows should not be confused with rows in the matrix A, which contains coefficient entries for each point in the domain of the problem). Boundary conditions are applied to make sure that b is not singular. The A matrix is also reduced, meaning that, while boundary conditions are accounted for, this is done without including a special row for them.

Then, in solvePsi() in libs/setup_2d.PDE.py, the matrix equation AΨ = b is solved, using a solver and parameters defined in the config file. Since A is reduced, this Ψ vector corresponds to every point in the problem domain excepting points on the boundaries.

Finally, in wrapPsi() in libs/setup_2d.PDE.py, the Ψ vector is transformed from a vector in the configuration space to a matrix in the problem domain. Then, values of Ψ on the boundaries are calculated and inserted.

Principal Findings

The default setup is as shown below (next page):

Patrick Halter
PHYS 613
HW3

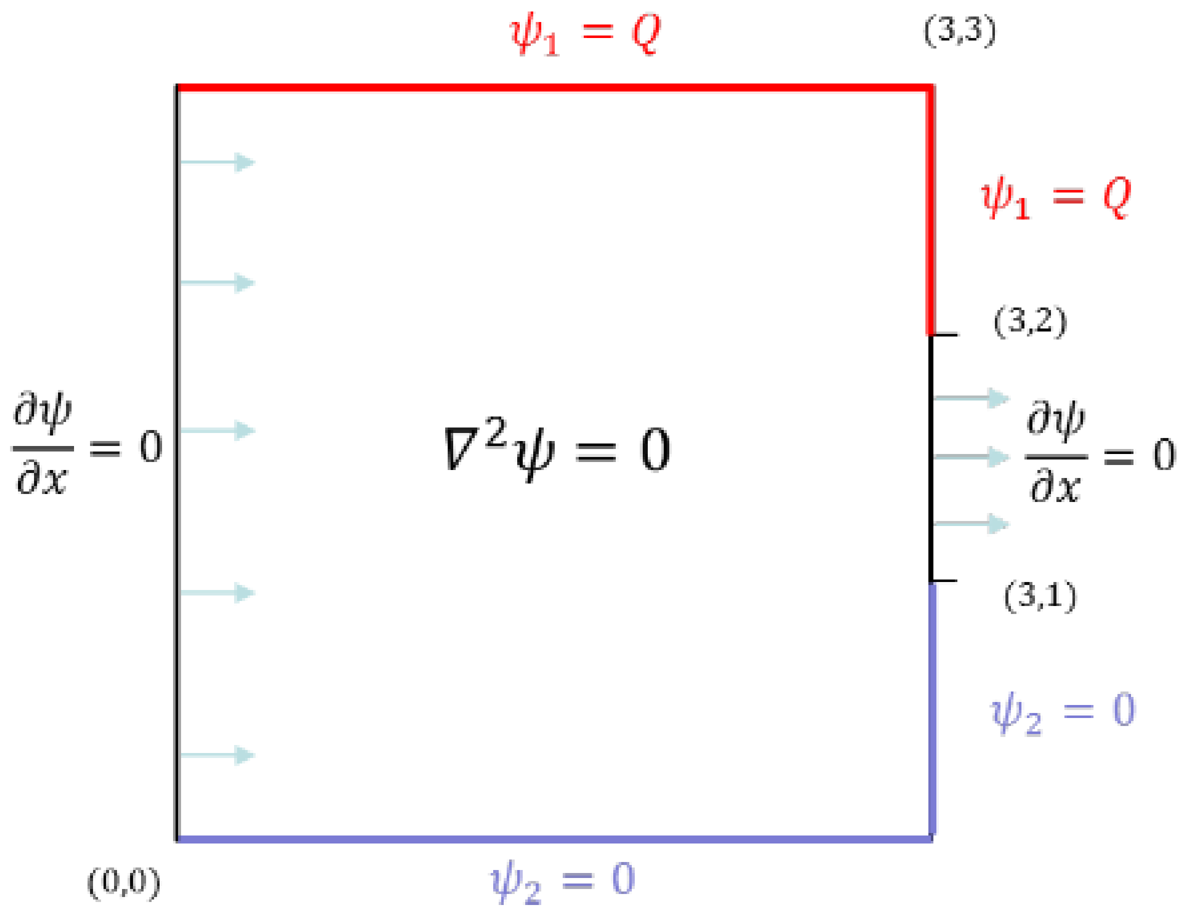

Illustration 3: Default conditions (as specified in homework handout)

Where Q = 1. Results are as seen above in Illustrations 1 and 2.

For all problem permutations, the values of the Neumann conditions and the non-Q Dirichlet conditions were kept the same, as was the domain of the problem. Positions of Q and the Neumann conditions were changed, as was the value of Q and the step size for the solution algorithm.

Q

When the value of Q is changed, the values of Ψ change throughout the problem domain. However, the overall colormap and vector map remain the same. In effect, changes in Q change the 'slope' of the problem, in that values (and differences) are more extreme, but relative to one another they remain the same. (Illustrations on next page)

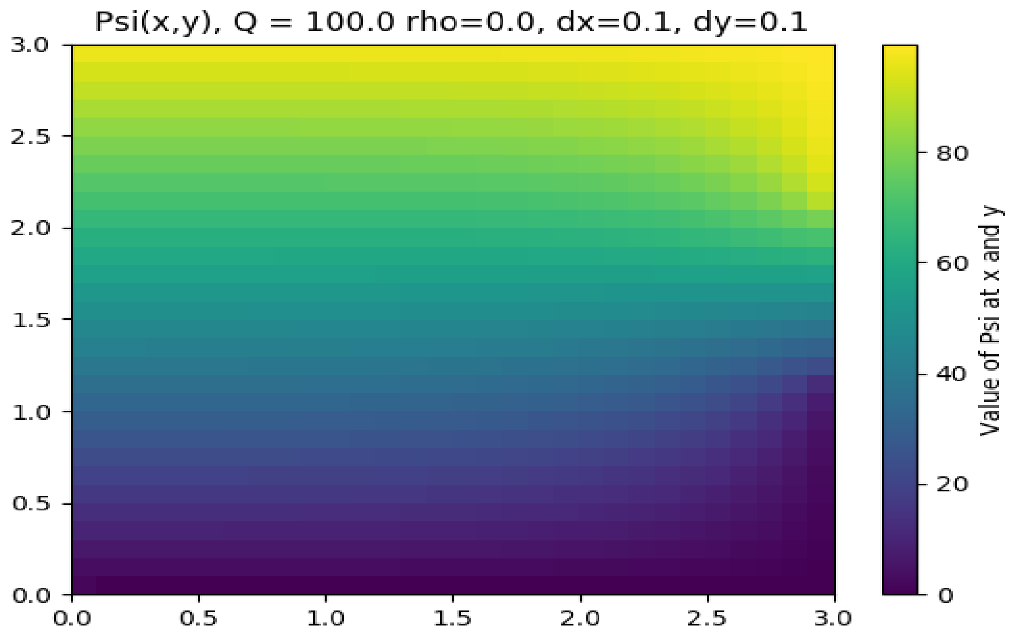

Illustration 4: Q = 100, all other defaults untouched

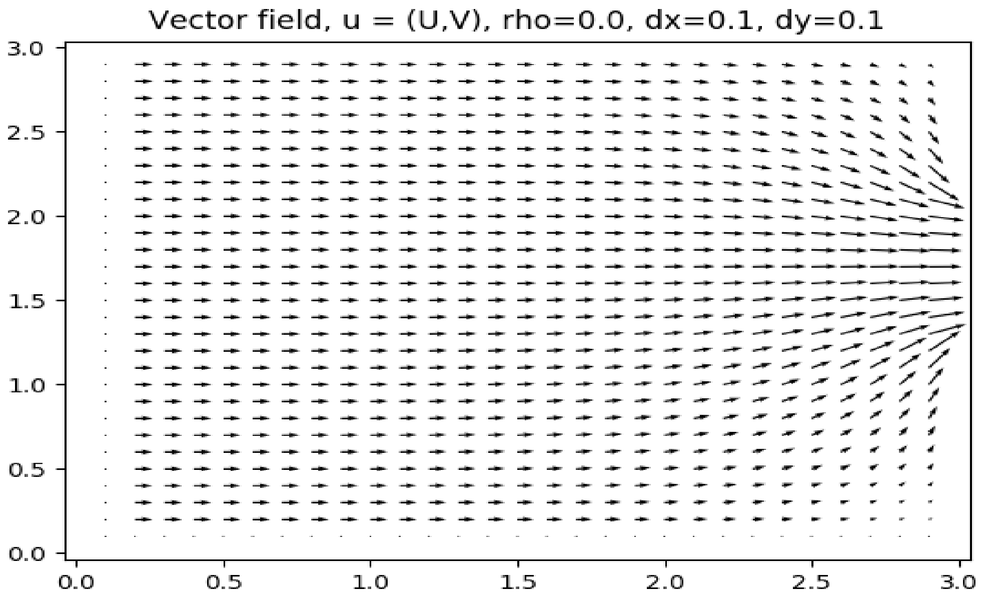

Illustration 5: Vectormap for same

When the position of the Q boundaries is changed, it behaves like changing the position of 'hills' in the path of a stream; for example, with Q on the bottom instead of the top, the direction of the vector field is simply reversed:

Finally, when the positions (or widths) of the Neumann boundary conditions change, that is, in effect, changing the location of allowed flux through the problem domain, or like changing the sources and sinks of a river. However, one should be careful: in a proper analogy a Neumann boundary condition acts as a constant-pressure valve, so whether it acts as a source or a sink is determined by whether or not the 'pressure' is high nearby. In either case, it allows the function to 'flow through' the boundary, either onto or off the problem domain, depending on the current direction of the vector field. (Illustrations on next page)

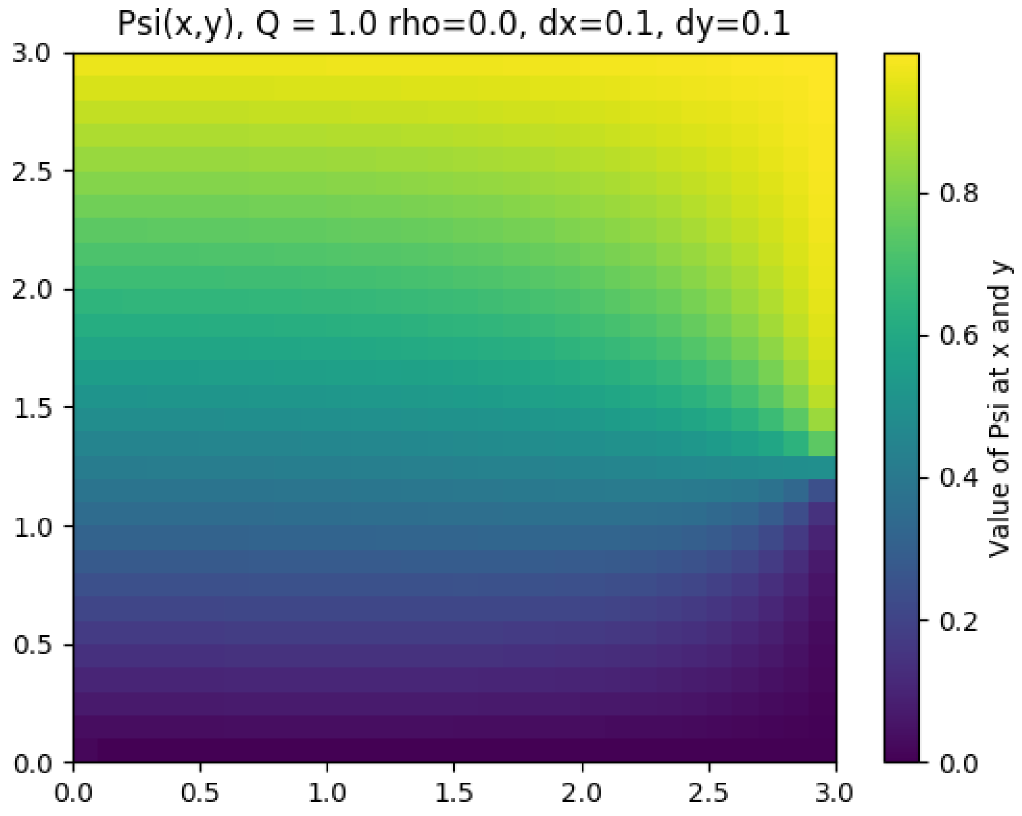

Illustration 6: Neumann condition on right restricted to (3, 1) - (3, 1.2)

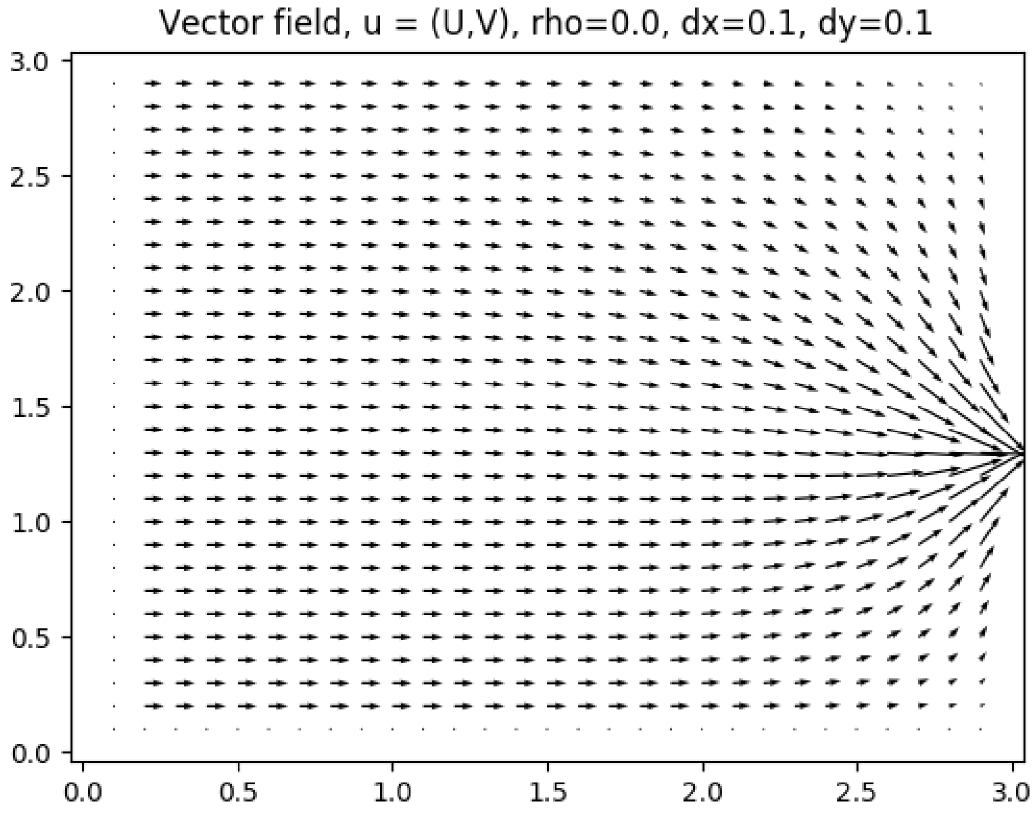

Illustration 7: Neumann condition on right restricted to (3, 1) - (3, 1.2)

Patrick Halter
PHYS 613
HW3

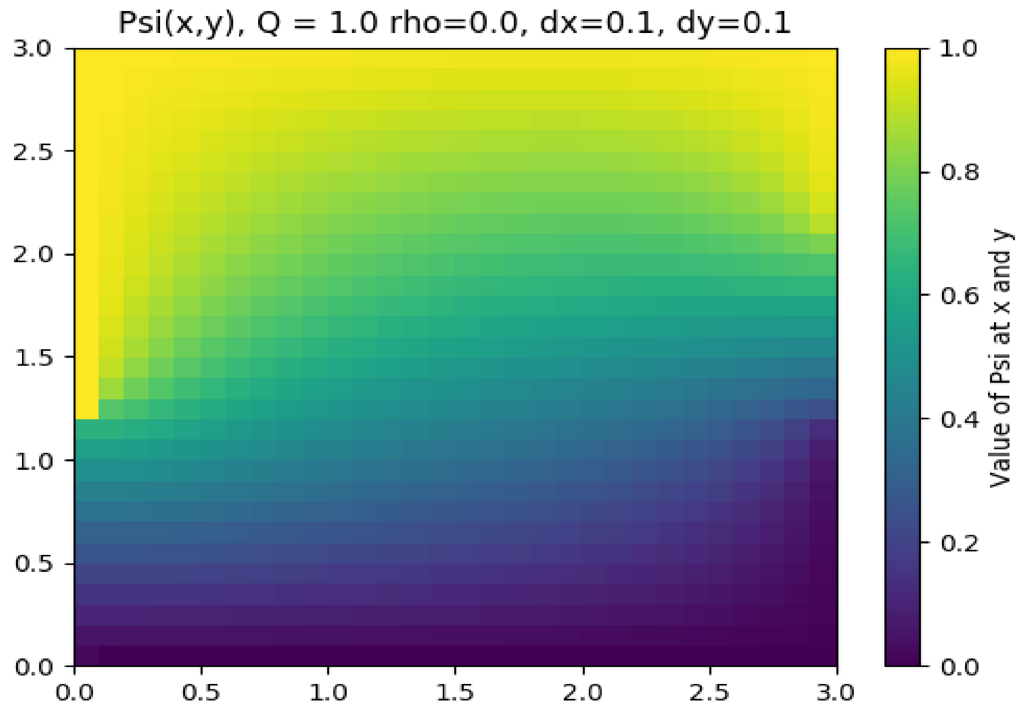

Illustration 8: Neumann on left restricted to (0,0) - (0,1), with the rest replaced with a Dirichlet

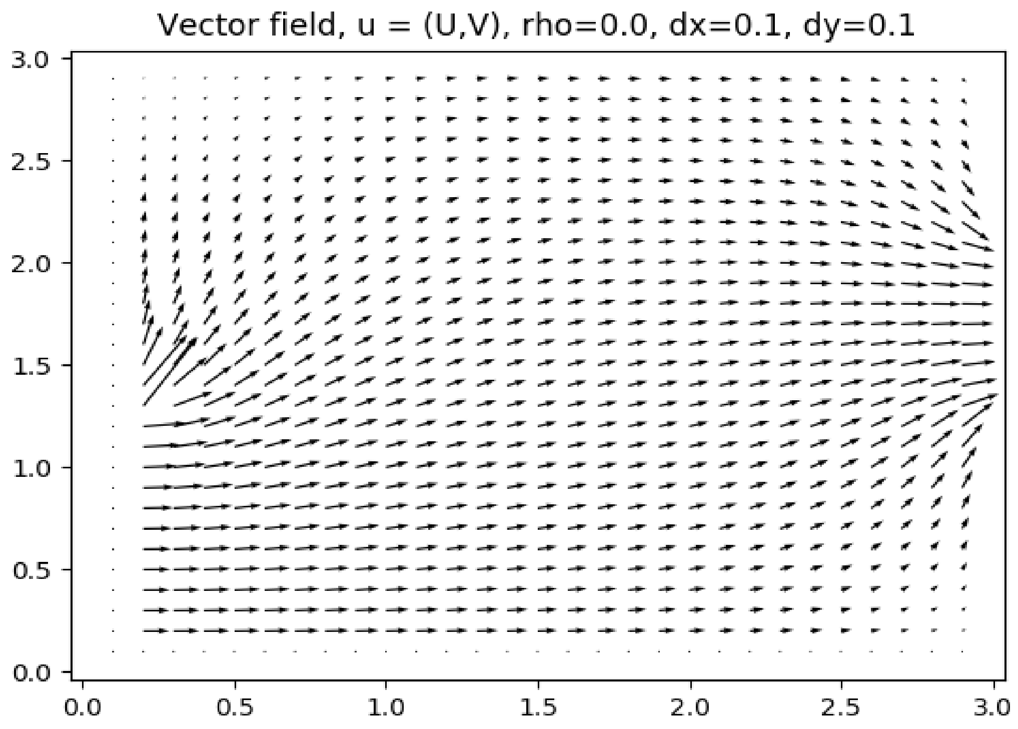

Illustration 9: Neumann on left restricted to (0,0) - (0,1), with the rest replaced with a Dirichlet boundary = Q

Patrick Halter
PHYS 613
HW3

Step Size

The step size parameters affect how large of a step is taken by the algorithm between evaluations of the function, and therefore can be throught of as a sort of ‘resolution’ parameter. With higher step sizes, you obtain much coarser, and therefore less accurate solutions (regardless of the order of approximation used), but the problem is much easier and much faster to evaluate. As noted in the beginning, step sizes of orders lower than 1e-1 are practicably unobtainable without above-average hardware.

Also of note, step sizes should be chosen to completely span the problem space. If there is a remainder, the boundaries may be misapplied, leading to inaccurate and unpredictable results.

As noted before, all illustrations up to this point have been rendered with step sizes dx = dy = 0.1. However, that is not required:

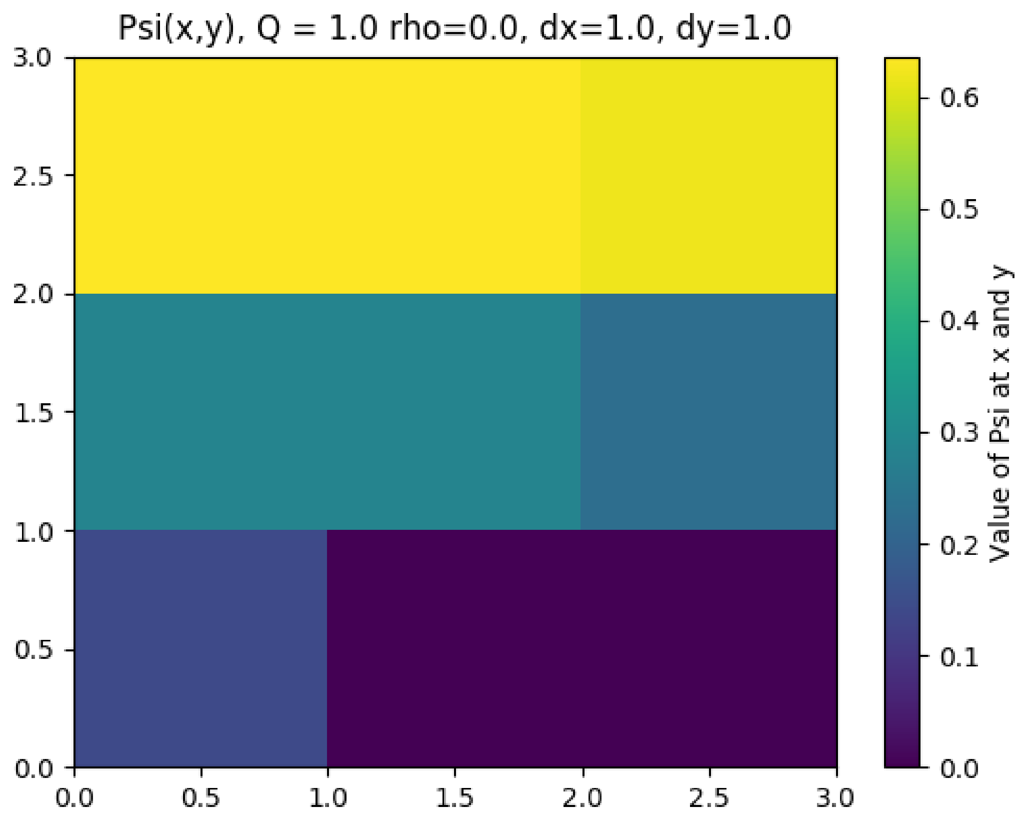

Illustration 10: Extremely coarse step size. The vector field plot is not intelligible for a step size this large.

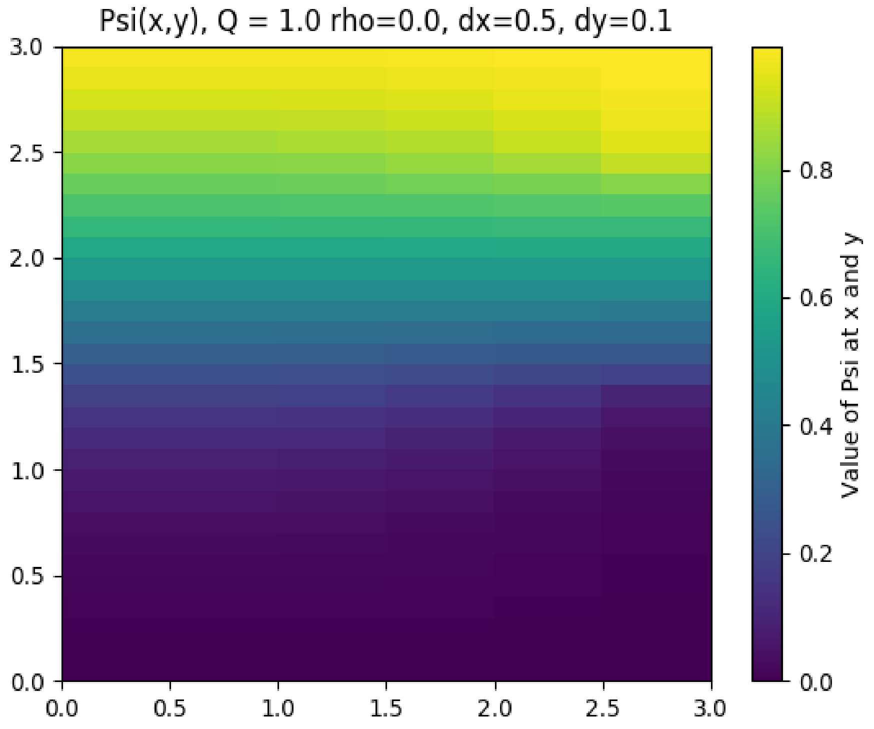

Illustration 11: Different step size for X and Y. Note that for these step sizes, you can see some artifacts in the colormap, but it still looks fairly continuous

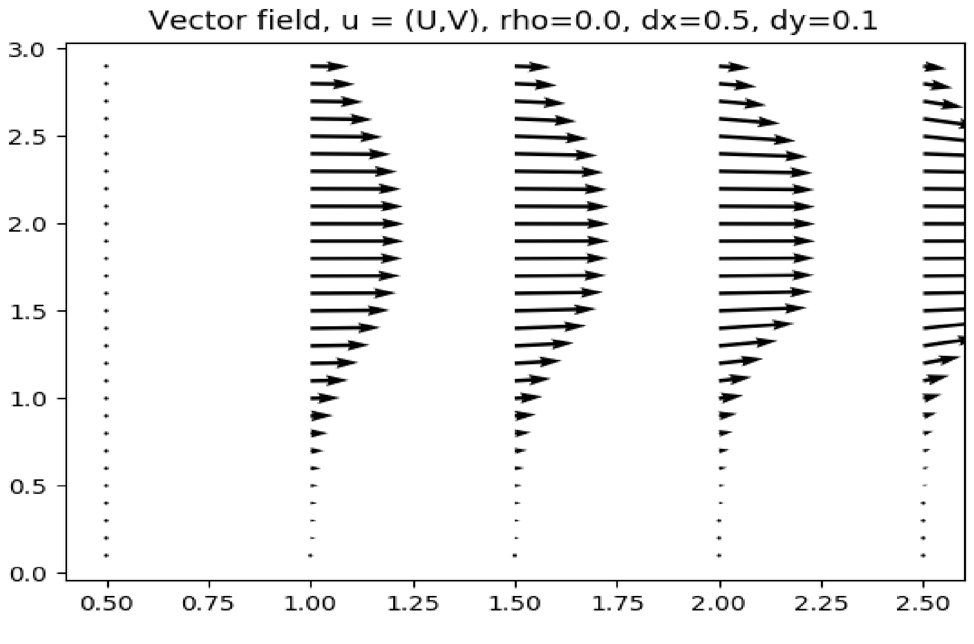

Illustration 12: However, the different step size is clearly visible on the vector map.

Patrick Halter
PHYS 613
HW3

Miscellaneous Addenda

As mentioned above, the majority of the calculated values for Ψ are accurate to 2nd order. However, in the case of Neumann boundary conditions, a 1st order approximation was used, because the 1st order approximation makes no presumptions about the size of the matrix.

Corners were handled as a special case. By construction corners should satisfy two boundary conditions (top left, bottom right, etc.), but cannot be guaranteed to do so, since the boundary conditions may be exclusive (for example, $\Psi = 1$ and $\Psi = 0$). Instead of privileging one boundary condition over the other, instead, the value at the corner was calculated as an average of the two nearest boundary points, both for ease of construction and because this seems likely to be more helpful than picking one boundary in avoiding artifacts.