MATH 330 - Computational Analysis
Dr. Eduardo Corona
Assignment # 4
Mikhail Smirnov (ID: 1249994)
December 22, 2020

The purpose of this project was to solve a homogenous and non-homogenous case of Poisson's Equation numerically on a 2D square on the domain [0,1] for both x and y. This was done by discretizing the equation by using finite difference formulas for the derivatives. Typically, in one dimension, the domain would be split in n subintervals where each step is incremented by $1/n$. From 0, this would mean there are (n+1) points. However, in a PDE, with a second dimension being varied, there would be $(n+1)^2$ points along an entire grid, in order to include all of the combinations of 2D coordinates, (x,y). In this case, there are also 4 boundary conditions that span along each side of the grid (Left, Right, Up, and Down). Therefore, the amount of unknowns that have to be solved is $(n-1)^2$.

The first part included making a function called indexgrid, which created a matrix of size (n+1)×3, where the first column was all of the possible combinations of the indices of x, the second column was all of the possible combinations of the indices of y, and the third column was a linear index, k, which simply numbered the combinations of the (x,y) indices. This resulting grid would help with incrementing and cycling through the indices.

The next part was related to creating a function, Laplace_square, in order to solve the boundary value problem of Poisson's Equation. By using the finite difference method, the difference formulas were substituted into the derivatives of Poisson's Equation and this would result in $(n-1)^2$ equations, which would all have varying indices of the x and y coordinates of the 2D grid. The coefficients of this set of equations were then placed in a matrix, L, while each of the solutions at certain points were placed in a vector, T. The matrix, L, ended up being pentadiagonal due to the discretized set of equations each having 5 terms, and therefore 5 nonzero coefficients. The right hand side of the set of equations were placed in a vector, B, also with varying points. This would then result in a system of equations, $LT=B$, being solved using Gaussian Elimination.

First, the solution in the code, T, was solved for the $(n-1)^2$ equations without boundary conditions. Something to note is that the boundary conditions still had to be added at certain elements in the matrix by subtracting them from the right hand side. This was due to the indices starting at (2,2) instead of (1,1), leading to missing boundary conditions in the L matrix. The resulting solution vector, T, was then reshaped into a matrix (size (n-1)×(n-1)), so that the columns rows can correspond to x and y, respectively. The last step was to incorporate the boundary conditions by creating a matrix, T_{sq} ,which was of size (n+1)×(n+1) and contained the T matrix and the boundary conditions on the new top row, bottom row, left column, and right column. This resulted in the final solution of $T(x,y)$ at each point in the x and y directions along a 2D grid in the domain [0,1], [0,1].

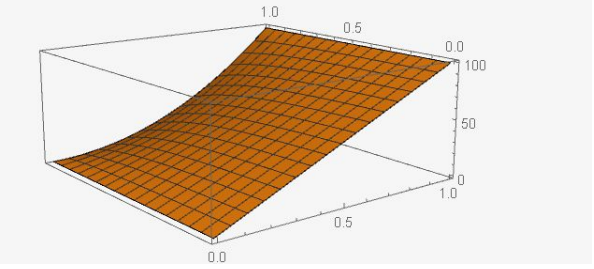
Once this function was created, a homogenous and non-homogenous case of Poisson's Equation was plugged into the function in order to plot and get a result, from the MATLAB script titled, PoissonBVP. Six figures were plotted, where the homogenous case was solved at varying amounts of points, n. This would mean that a higher amount of points should result in a clearer and smoother graph. The values of n were varied as 10, 20, and 40. The solutions of both the homogenous and the non-homogenous case had some similarities and differences. Firstly, it is interesting to note that a homogenous case of Poisson's Equation simplifies to Laplace's Equation. Graphically, both solutions look like an inclined, slightly curved sheet. However, the homogenous solution looks a lot more flat, with no bumps or hills. The non-homogenous solution instead has multiple "hills" and

valleys on the 2D grid, making it much less flat bumpier. Lastly, the MATLAB plots were compared with Mathematica plots from using the NDSolve function. The Mathematica plots have much smoother functions due to possibly having more points and incorporating interpolation, and much more accurate boundary conditions. An issue with my MATLAB graphs had to do with the boundary conditions varying at different points along the matrix and not being completely consistent with varying values of *n*.

Mathematica Plots

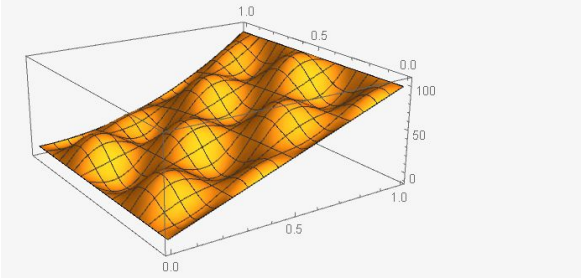
Solving Txx(x,y) + Tyy(xy) = 0 on domain [0,1]
With boundary conditions:
T(x,0) = 100x, T(x,1) = 100x^2,
T(0,y) = 0, T(1,y) = 100

```
PDE = D[T[x,y],x,x] + D[T[x,y],y,y];
BCs={T[x,0]==100*x,
T[x,1]==100*x^2,
T[0,y]==0,
T[1,y]==100};
homogenousSol=NDSolve[{PDE==0, BCs},T,{x,0,1},{y,0,1}];
Plot3D[Evaluate[T[x,y]/.homogenousSol,{x,0,1},{y,0,1}]]
```



Solving Txx(x,y) + Tyy(xy) = 6400sin(4πx)cos(4πy) on domain [0, 1]
With boundary conditions
T(x,0) = 100x, T(x,1) = 100x^2,
T(0,y) = 0, T(1,y) = 100

```
PDE = D[T[x,y],x,x] + D[T[x,y],y,y];
BCs={T[x,0]==100*x,
T[x,1]==100*x^2,
T[0,y]==0,
T[1,y]==100};
nonHomogenousSol=NDSolve[{PDE==6400*Sin[4*Pi*x]*Cos[4*Pi*y],
BCs},T,{x,0,1},{y,0,1}];
Plot3D[Evaluate[T[x,y]/.nonHomogenousSol,{x,0,1},{y,0,1}]]
```



MATLAB Plots