2D Poisson Equation:

A Finite Difference Approach

Zach Myrick

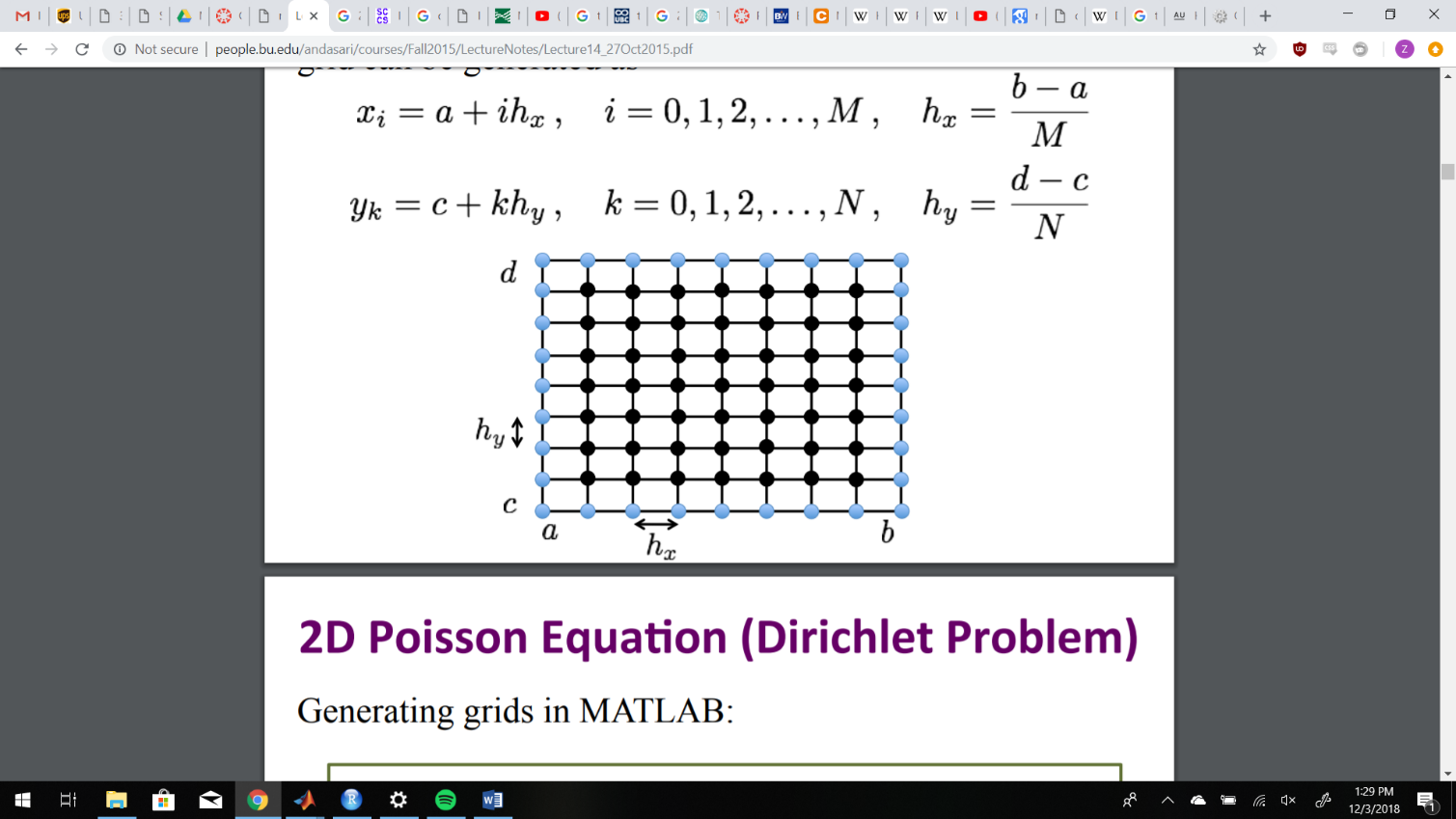
Section 2

In this paper I will develop a method for approximating solutions to the 2-dimentional Poisson equation with the finite difference method in MATLAB at a fixed point in time. A centered difference formula for second order partial derivatives over the discretized domain will form a linear system, then solving this system over a square domain will lead to approximate solutions.

Recall the one-dimensional heat equation which was examined in class, https://latex.codecogs.com/gif.latex?u_t%20%3D%20u_x_x. This is a specific case of multi-dimensional form https://latex.codecogs.com/gif.latex?%5Cfrac%7B%5Cpartial%20u%7D%7B%5Cpartial%20t%7D%20%3D%20k%5Ctriangledown%20%5E2u%20&plus;%5Cfrac%7BQ%7D%7Bcp%7D where Q is heat energy generated per unit time, c is specific heat, p is mass density, and u is temperature at any point and time. For simplicity, it will be assumed that Q is 0, meaning there are no external sources relating to heat inflow or outflow from the system, eliminating the https://latex.codecogs.com/gif.latex?%5Cfrac%7BQ%7D%7Bcp%7D term. Also recall from Calculus 3 that  . The equation  is called Laplace’s equation, and is the equivalent of solving for a time independent solution to the heat equation. But Laplace’s equation is just the homogenous case of the Poisson equation, i.e. f =0 in the partial differential equation https://latex.codecogs.com/gif.latex?%5Cbigtriangledown%20%5E2u%20%3D%20f.

Now, finding analytic solutions to this equation is not a simple matter, and often times requires analytic methods well beyond the scope of what I have learned up to this point. One method involves solving Green’s function https://latex.codecogs.com/gif.latex?u%28%5Cmathbf%7Br%7D%29%20%3D%20-%5Cint%20%5Cint%20%5Cint%20%5Cfrac%7Bf%28%5Cmathbf%7Br%27%7D%29%7D%7B4%5Cpi%20%5Cleft%20%7C%20%5Cboldsymbol%7Br%7D%20-%5Cboldsymbol%7Br%7D%27%20%5Cright%20%7C%7Dd%5E3r%27, another involves separation of variables. There are often times complex, harmonic, or Fourier solutions. Instead of dealing with these, utilize the finite difference method to obtain approximate values for u, but do so at a fixed point in time.

As is common in numerical analysis, begin by breaking the problem down into discrete subintervals. Since this problem is in 2 dimensions, and we are interested in approximating solutions to u(x,y) on the vertical axis, discretize the (x,y) plane into a uniform Cartesian grid. Do so with equations https://latex.codecogs.com/gif.latex?x_i%20%3D%20a%20&plus;%20ih_x, https://latex.codecogs.com/gif.latex?y_k%20%3D%20c%20&plus;%20kh_y, for i=0,1,…,M, and j=0,1,…,N and https://latex.codecogs.com/gif.latex?h_x%20%3D%20%28b-a%29/M, https://latex.codecogs.com/gif.latex?h_y%20%3D%20%28d-c%29/N.

Now let’s derive the second order partial derivative centered difference formula, which will be seen to be https://latex.codecogs.com/gif.latex?O%28h%5E4%29 accurate.

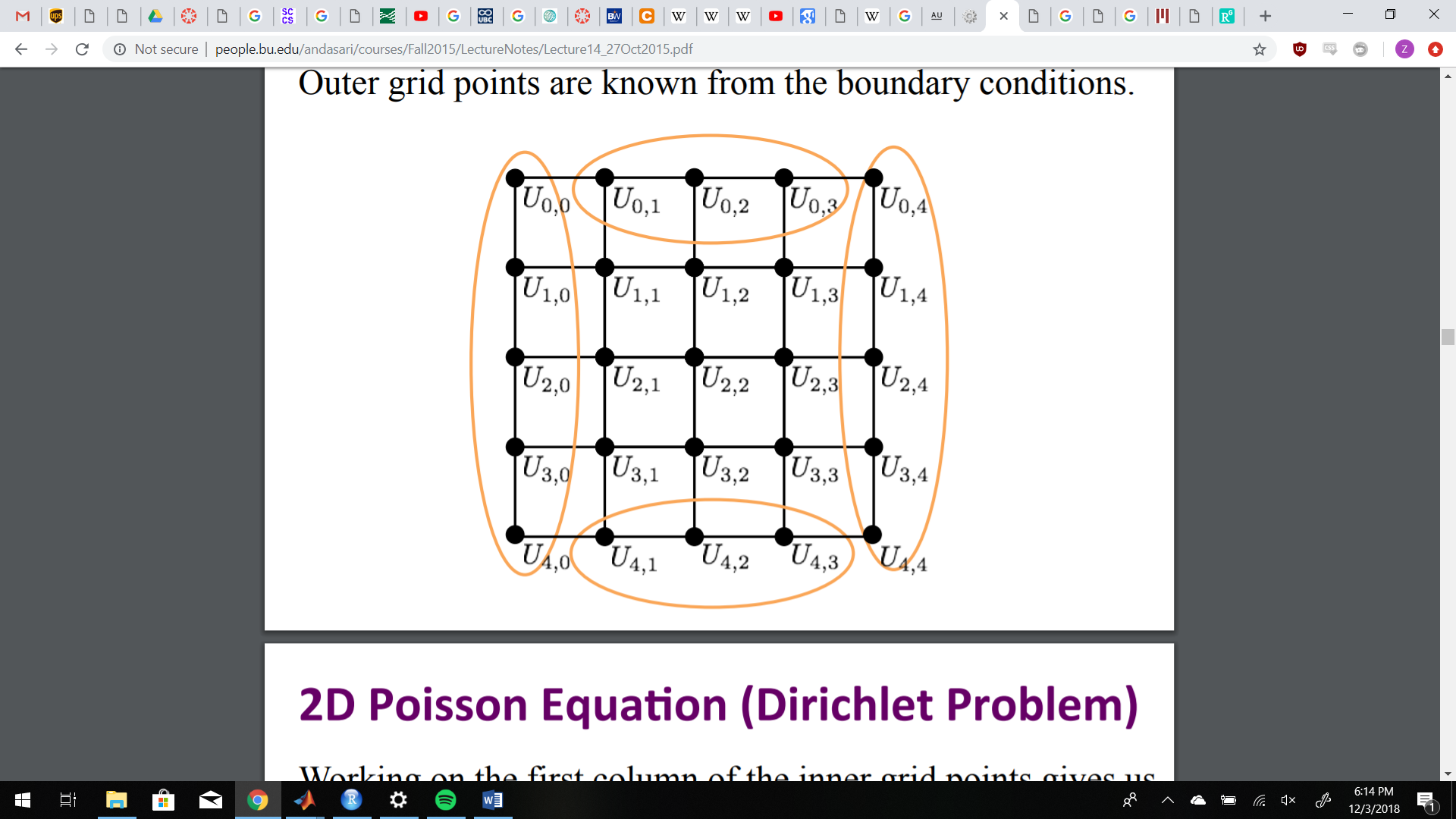
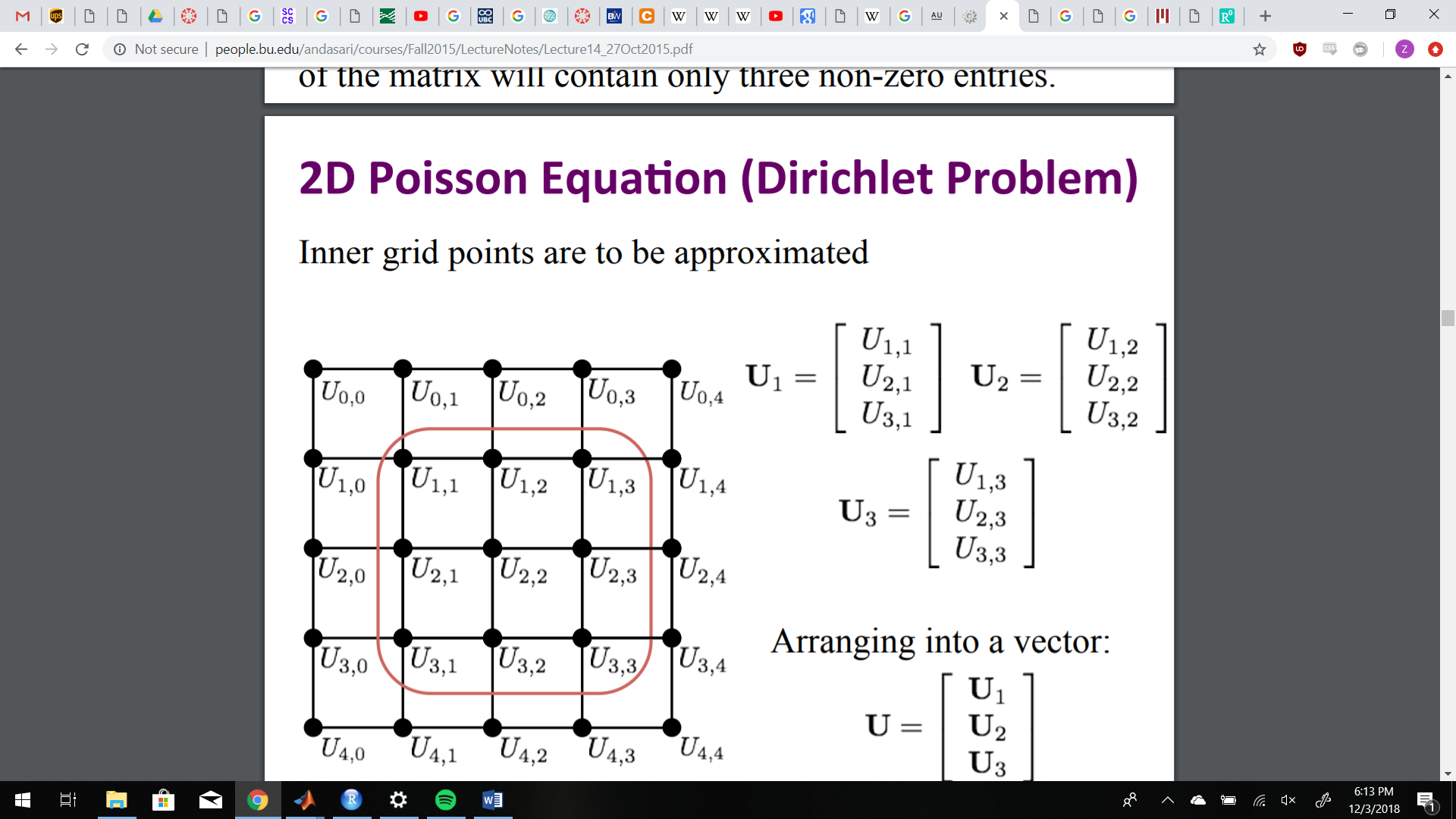
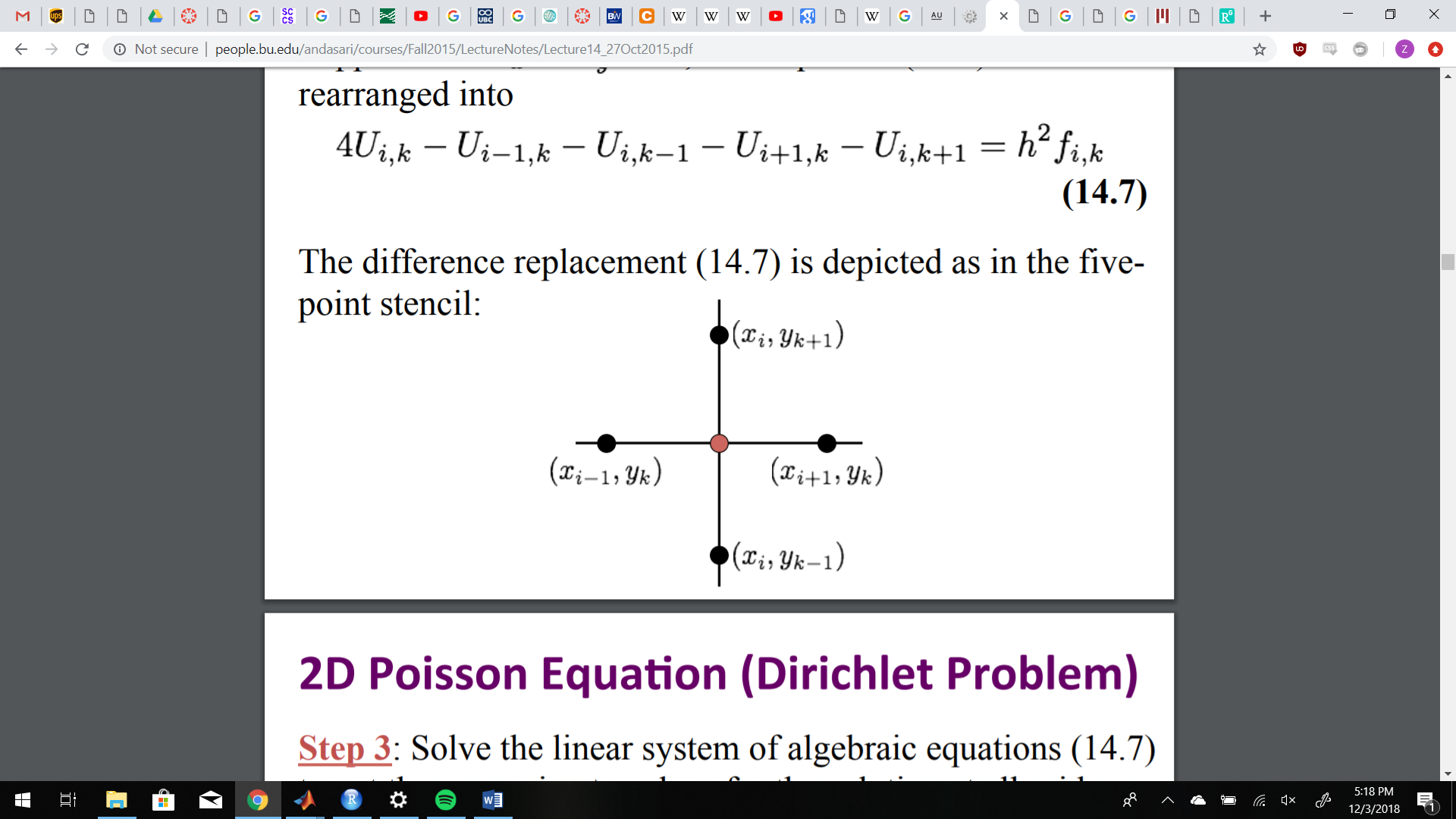
With that out of the way, the Poisson equation can now be approximated as 

where https://latex.codecogs.com/gif.latex?f_i_%2C_k%20%3D%20f%28x_i%2Cy_k%29 and i=1,2,…,M-1, k=1,2,…,N-1 since boundary points don’t need to be approximated. Note that terms respectively are not included in the approximation formula. This means they are part of truncation error as well, so https://latex.codecogs.com/gif.latex?T_i_%2C_k%20%7E%7E is given by   
 and https://latex.codecogs.com/gif.latex?h%3Dmax%5Cbegin%7BBmatrix%7D%20h_x%2C%20h_y%5C%5C%20%5Cend%7BBmatrix%7D. It is easily observable that https://latex.codecogs.com/gif.latex?%5Clim_%7Bh%20%5Crightarrow%200%7D%5Cleft%20%5C%7C%20T%20%5Cright%20%5C%7C%3D0, meaning the approximation is consistent, so we may for now ignore the error term and replace the corresponding https://latex.codecogs.com/gif.latex?u%28x_i%2Cy_k%29 values with https://latex.codecogs.com/gif.latex?U_i_%2C_k, yielding equation https://latex.codecogs.com/gif.latex?-%5B%28U_i_-_1_%2C_k%20&plus;%20U_i_%2C_k%20&plus;%20U_i_&plus;_1_%2C_k%29/h_x%5E2%20&plus;%20%28U_i_%2C_k_-_1&plus;U_i_%2C_k&plus;U_i_%2C_k_&plus;_1%29/h_y%5E2%5D%3D%20f_i_%2C_k https://latex.codecogs.com/gif.latex?%5CRightarrow

https://latex.codecogs.com/gif.latex?-%28U_i_-_1_%2C_k%20&plus;%20U_i_&plus;_1_%2C_k%29/h_x%5E2%20-%20%28U_i_%2C_k_-_1&plus;U_i_%2C_k_&plus;_1%29/h_y%5E2%20&plus;%20%5B%281/h_x%5E2%29&plus;%281/h_y%5E2%29%5D2U_i_%2C_k%20%3D%20f_i_%2C_k for the same range of i and k.

However, the grid subdividing the domain was designed to be uniform, so let https://latex.codecogs.com/gif.latex?h%20%3D%20h_x%3Dh_y. Now the approximation equation becomes https://latex.codecogs.com/gif.latex?-%28U_i_-_1_%2C_k%20&plus;U_i_&plus;_1_%2C_k%20%29/h%5E2%20-%28U_i_%2C_k_-_1%20U_i_%2C_k_&plus;_1%20%29/h%5E2%20&plus;%5B%281/h%5E2%29&plus;%281/h%5E2%29%5D2U_i_%2C_k https://latex.codecogs.com/gif.latex?%5CRightarrow

Rft5rhttps://latex.codecogs.com/gif.latex?4U_i_%2C_k%20-U_i_-_1_%2C_k%20-U_i_%2C_k_-_1%20-U_i_&plus;_1_%2C_k-%20U_i_%2C_k_&plus;_1%3Dh%5E2f_i_%2C_k which is depicted by the 5-point stencil image on the domain.



In order to obtain the approximations, the above approximation equation must be solved for at all grid points. In order to do this, the approximation equation will be written in a linear system of the form **AU = F** where **A** is a matrix of coefficients on the approximation equations for all i and k, **U** is a column vector of all https://latex.codecogs.com/gif.latex?U_i_%2C_k, and **F** is a column vector of all https://latex.codecogs.com/gif.latex?h%5E2f_i_%2C_k. Note that some (i,k) pairs will be the values of the boundary conditions and will be subsequently moved to the right hand side of the equation.

For simplicity of example, suppose M = N = 4. We then have boundary values for all pairs of the form (i, 0), (i, 4), (0,k), and (4, k), and must solve for the remaining https://latex.codecogs.com/gif.latex?U_i_%2C_k where 0<i<4 and 0<k<4. The above graphics visualizes this.

For illustrative purposes, it is more convenient to examine individual equations for the time being. The first column of inner grid points, that is k=1, i=1,2,3, are expressed as follows, k=1, i=1 shows the boundary conditions being moved to the right hand side:



https://latex.codecogs.com/gif.latex?4U_2_%2C_1%20-%20U_1_%2C_1%20-%20U_3_%2C_1%20-%20U_2%2C_2%3D%20h%5E2f_2_%2C_1%20&plus;%20U_2_%2C_0

https://latex.codecogs.com/gif.latex?4U_3_%2C_1%20-%20U_2_%2C_1%20-%20U_3_%2C_2%20%3D%20h%5E2f_3_%2C_1%20&plus;%20U_3_%2C_0%20&plus;%20U_4%2C_1

Now the second column of inner grid points, k=2, i=1, 2, 3:

https://latex.codecogs.com/gif.latex?4U_1_%2C_2%20-%20U_2_%2C_2%20-%20U_1_%2C_1%20-%20U_1%2C_3%20%3D%20h%5E2f_1_%2C_2%20&plus;%20U_0_%2C_2https://latex.codecogs.com/gif.latex?4U_2_%2C_2%20-%20U_1_%2C_2%20-%20U_3_%2C_2%20-%20U_2%2C_1%20-%20U_2_%2C_3%20%3D%20h%5E2f_2_%2C_2https://latex.codecogs.com/gif.latex?4U_3_%2C_2%20-%20U_2_%2C_2%20-%20U_3_%2C_1%20-%20U_3%2C_3%20%3D%20h%5E2f_3_%2C_2%20&plus;%20U_4_%2C_2

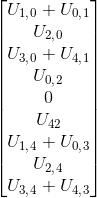
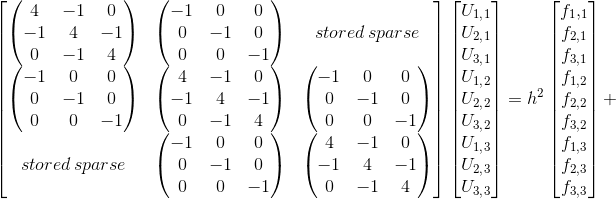
And the third and final column k =3, i=1, 2, 3:

https://latex.codecogs.com/gif.latex?4U_1_%2C_3%20-%20U_2_%2C_3%20-%20U_1_%2C_2%20%3D%20h%5E2f_1_%2C_3%20&plus;%20U_0_%2C_3%20&plus;U_1%2C_4

https://latex.codecogs.com/gif.latex?4U_2_%2C_3%20-%20U_1_%2C_3%20-%20U_3_%2C_3%20-U_2%2C_2%20%3D%20h%5E2f_2_%2C_3%20&plus;%20U_2_%2C_4

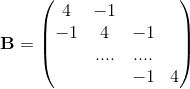
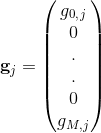
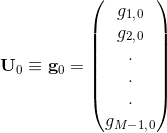
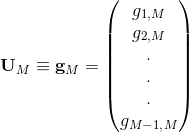
https://latex.codecogs.com/gif.latex?4U_3_%2C_3%20-%20U_2_%2C_3%20-%20U_3_%2C_2%20%3D%20h%5E2f_3_%2C_3%20&plus;%20U_4_%2C_3-U_3%2C_4

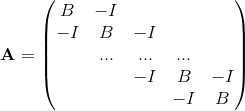
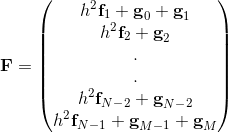
The respective columns may all be individually written as matrices, and then assembled in sequential order to form the following matrix:



Notice the near tridiagonal structure of the resulting system. This will come in handy since it is always a goal is to minimize computation power necessary for calculations, however the zeros disrupting the tridiagonal structure will prove to be an annoyance. This structure is called block tridiagonal.

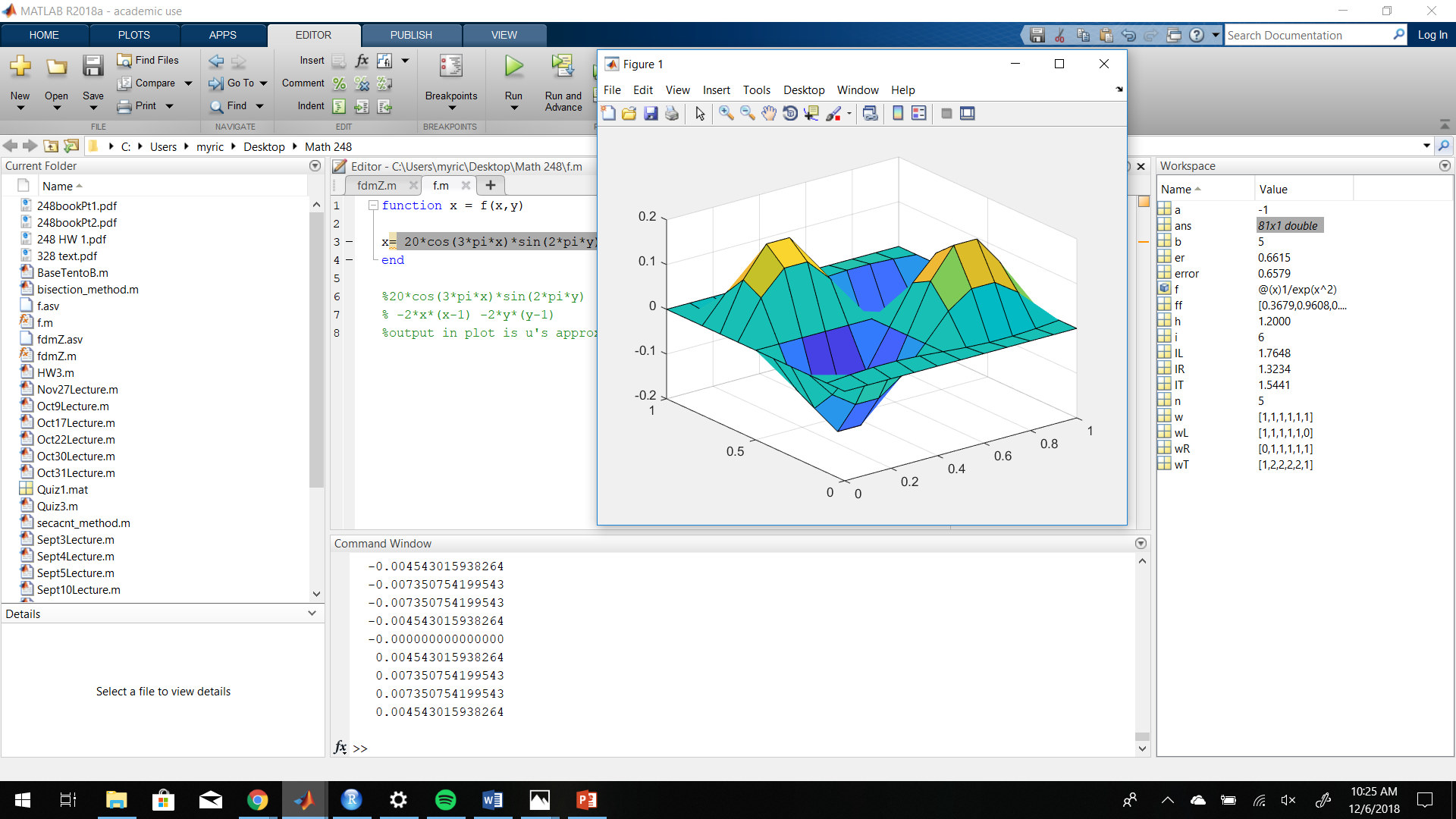
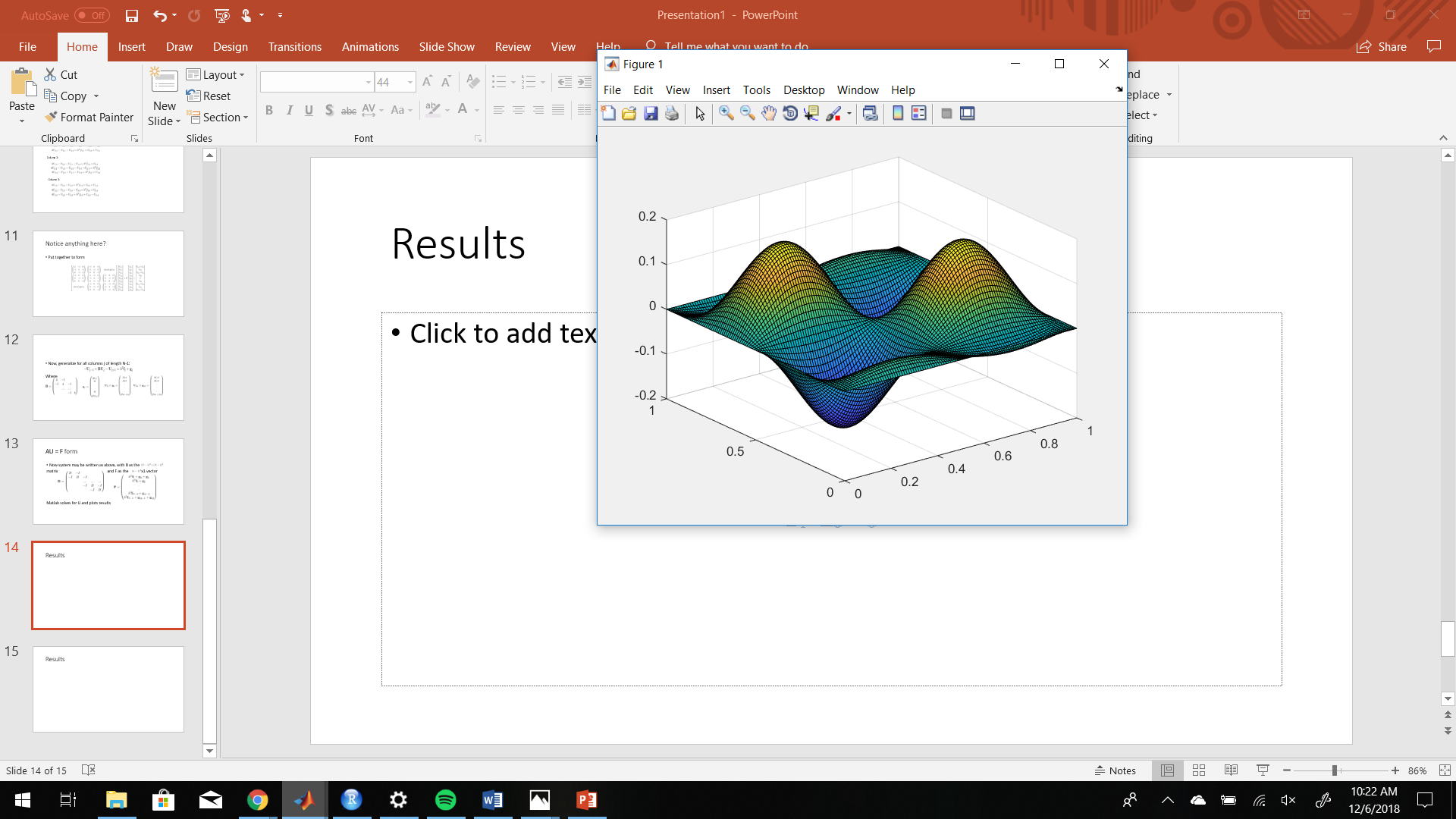
Now, we may generalize the square grid to size MxM with a few similar steps.

Begin by noting that each individual column point is dependent only upon the value above it and below it, re-examine the 5-point stencil image on the previous page if necessary. Now, we may write the above matrix in terms of column vectors https://latex.codecogs.com/gif.latex?-%5Cmathbf%7BU%7D_j_-_1%20&plus;%20%5Cmathbf%7BB%7D%5Cmathbf%7BU%7D_j%20-%5Cmathbf%7BU%7D_j_&plus;_1%20%3D%20h%5E2%5Cmathbf%7Bf%7D_j%20&plus;%5Cmathbf%7Bg%7D_j where **B** is the (M-1)x(M-1) matrix  . Note the dimensions, which are as such due to not needing to approximate boundary points. The boundary points are given by column vector  forming the top and bottom boundaries and , filling in the sides.

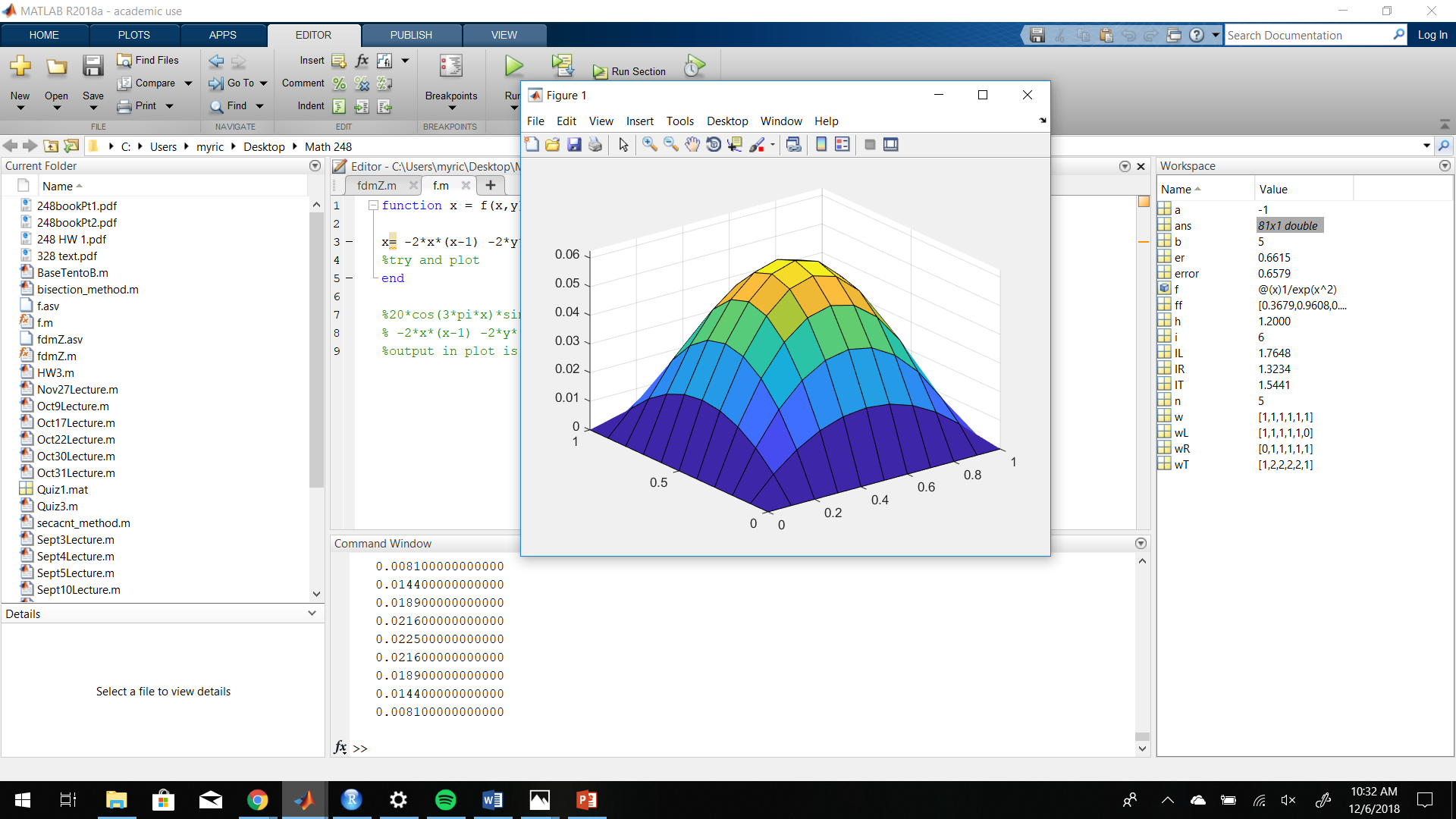
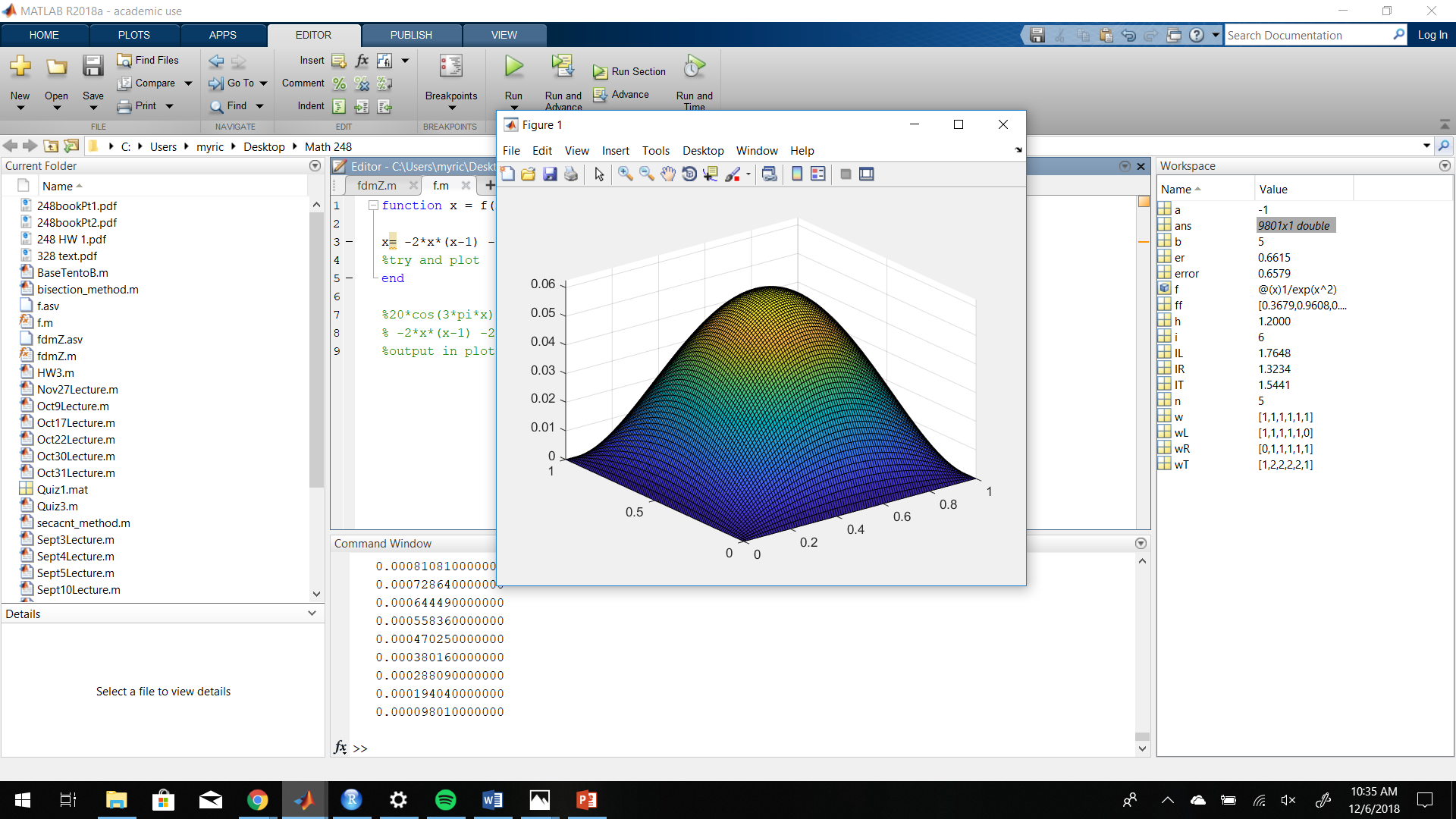
Then, the matrix ban be put into **AU = F** form, with  and. The code in the appendix solves this equation, and outputs a graph of a function approximating u at a fixed point in time. This output plot models the manner in which heat travels through the 2D surface, representing the paths taken and relative rates at which the heat would become “diffused” through a 2D surface when applied. In the output, areas of the surface are color coded to represent relative temperatures, since time is fixed and the external heat source term Q was assumed 0 in the initial heat equation. The given **f** manually input into the function cautiously models properties of the 2D surface, like it’s density and heat capacity.

The plots on the next page display approximations at different values of h on the domain of a unit square.

https://latex.codecogs.com/gif.latex?f%20%3D%2020sin%283%5Cpi%20x%29cos%282piy%29%2C%20h%3D.1%20%5C%2C%20vs%20%5C%3A%20h%3D.01

https://latex.codecogs.com/gif.latex?f%20%3D%20-2x%28x-1%29-2y%28y-1%29%2C%20h%3D.1%20%5C%2C%20vs%20%5C%3A%20h%3D.01

The five point stencil method has an advantage in being able to approximate over any domain that is in combinations of straight lines, including piecewise functions which otherwise wouldn’t have analyitic solutions. Addtiontionally, edits to the matlab code could allow for solving Dirichlet and Neumann type boundry conditoins, which would allow for practically any shape on the domain to be solved for.

In order to display accuracy, I could have potentially plotted a solution to a problem solved analytically for comparison against the approxiation, but unfortunatly due to time constraints I did not.

This code in particular, unfortunately, is rather comutationally expensive. It relies on the Matlab “\” algorithm, a hight level proprietary version of Gaussian elimination with back substitution, in order to solve the system. The algorithm performs the following checks on matrix structure:

1. If A is sparse and banded, employ a banded solver.
2. If A is an upper or lower tridiagonal, employ a backward substitution algorithm
3. If A is symmetric and has real positive diagonal elements, attempt Cholesky factorization. If A is sparse, employ reordering first to minimize fill in.
4. If none of the above are met, do a general triangular fatorization using Gaussian elimination with partial pivoting
5. If A is sparse, employ UMFPACK library.
6. If A is not squarel employ alogrithms based on QR factorization for undetermined systems.

This means the program could be improved by employing the general Thomas Algorithm, cyclic reduction, successive overrelaxation, fast forier fransforms, or a multigrid method could be used to compute a solution of optimal computational complexity order.

C Appendix

% this function approximates the solution u to the differential equation

% with the function in f.m given and f=-u''(x,y) using the finite difference

% method.

function [u\_h] = fdm(A,B,h)

%Input square domain [a,b]\*[c,d]

a=A(1);

b=A(2);

c=B(1);

d=B(2);

%Input h: The meshsize (distance between two grid points)

m=(b-a)/h -1; %Total number of grid points along the x-axis.

n=(d-c)/h -1; %Total number of grid points along the y-axis.

%Creating x-components of the grid points

x=[a+h:h:a+m\*h];

y=[c+h:h:c+m\*h];

%Making A as a sparse matrix.

i(1)=1;

j(1)=1;

s(1)=4;

i(2)=1;

j(2)=2;

s(2)=-1;

i(3)=m\*n;

j(3)=m\*n-1;

s(3)=-1;

i(4)=m\*n;

j(4)=m\*n;

s(4)=4;

count1=5;

for k=2:m\*n-1

i(count1)=k;

i(count1+1)=k;

i(count1+2)=k;

j(count1)=k;

j(count1+1)=k-1;

j(count1+2)=k+1;

s(count1)=4;

s(count1+1)=-1;

s(count1+2)=-1;

count1=count1+3;

end

for k=1:m\*n

if k+n<=m\*n

i(count1)=k;

j(count1)=k+n;

s(count1)=-1;

i(count1+1)=k+n;

j(count1+1)=k;

s(count1+1)=-1;

count1=count1+2;

end

end

%Need one more step before creating the sparse matrix. There are some "holes (zeros)" in the tridiagonal of A. Can you find where they are?

A=sparse(i,j,s,m\*n,m\*n);

for i=1:m-1

A(i\*n+1,i\*n) = 0;

A(i\*n,i\*n+1) = 0;

end

count = 1;

for i=1:m

for j=1:n

rhs(m\*(i-1)+j) = f(a+(i)\*h,c+(j)\*h)\*h^2;

end

count = count + 1;

end

u\_h = A\transpose(rhs); %solution

count = 1;

%plot

for i = 0 : m + 1

for j = 0 : n + 1

X(i+1,j+1) = a + j\*h; %matricies for plotting grid (2d plot, define axis as vector. 3d, define axis as matrix)

Y(i+1,j+1) = c + i\*h;

if (i == 0 || i == m + 1 || j == 0 || j == m + 1) %if on boundry, is zero

Z(i+1,j+1) = 0;

else

Z(i+1,j+1) = u\_h(count); % otherwise plot solution value

count = count + 1;

end

end

end

surf(X,Y,Z);

end

%f.m

function x = f(x,y)

x= -2\*x\*(x-1) -2\*y\*(y-1)

end

Works Cited

Noboru Kobashigowa, former math 449 student. Biggest help on the code.

Various Youtube videos

Wikipedia

Farside.ph.utexas.edu. (2018). *2-d problem with Dirichlet boundary conditions*. [online] Available at: http://farside.ph.utexas.edu/teaching/329/lectures/node67.html [Accessed 7 Dec. 2018].

People.bu.edu. (2018). [online] Available at: http://people.bu.edu/andasari/courses/Fall2015/LectureNotes/Lecture14\_27Oct2015.pdf [Accessed 7 Dec. 2018].

Mathfaculty.fullerton.edu. (2018). [online] Available at: http://mathfaculty.fullerton.edu/mathews/n2003/differentiation/numericaldiffproof.pdf [Accessed 7 Dec. 2018].

Mat.iitm.ac.in. (2018). *Finite Difference approximations to partial derivatives*. [online] Available at: https://mat.iitm.ac.in/home/sryedida/public\_html/caimna/pde/Third/third.html [Accessed 7 Dec. 2018].