MATH 517 Finite Differences Homework 3

Caleb Logemann

March 2, 2016

1. Consider Poisson's equation in 2D:

$$\begin{aligned} -u_{xx} - u_{yy} = f(x,y) \text{ in } \Omega = [0,1]\times[0,1] \\ u = g(x,y) \text{ on } \partial\Omega \end{aligned}$$

Discretize this equation using the 9-point Laplacian on a uniform mesh $\Delta x = \Delta y = h$. Use the standard row-wise ordering.

First we need to specify the discretization of the space Ω. Let $x_i = ih$ for $i = 0, 1, \ldots, N+1$ and let $y_j = jh$ for $j = 0, 1, \ldots, N+1$, where $h = \frac{1}{N+1}$. Then the solution to this PDE can be described by approximating u on this mesh, that is $U_{i,j} \approx u(x_i, y_j)$ is an approximation to the exact solution.

Now we can apply finite differences to this PDE. The 9-point Laplacian on a uniform mesh is given by

$$\Delta_9 u = \frac{1}{6h^2}(U_{i-1,j-1} + U_{i-1,j+1} + U_{i+1,j-1} + U_{i+1,j+1} + 4U_{i-1,j} + 4U_{i+1,j} + 4U_{i,j-1} + 4U_{i,j+1} - 20U_{i,j}).$$

It can be shown that

$$\Delta_9 u = \Delta u + \frac{h^2}{12}\Delta f + O(h^4).$$

Using this finite difference in the PDE results in the following set of equations that include the Laplacian of the forcing function.

$$\begin{aligned} \frac{1}{h^2}(20U_{i,j} - 4U_{i-1,j} - 4U_{i+1,j} - 4U_{i,j-1} - 4U_{i,j+1} - U_{i-1,j-1} - U_{i-1,j+1} - U_{i+1,j-1} - U_{i+1,j+1}) \\ = f_{ij} + \frac{h^2}{2}\Delta f_{ij} \end{aligned}$$

where $f_{ij} = f(x_i, y_j)$ and $\Delta f_{ij} = \Delta f(x_i, y_j)$.

Now in order to turn this into a linear system, a new numbering scheme needs to be imposed. I will use the natural row-wise ordering where each point is numbered along the rows starting at $U_{1,1} = U_1$. In general $U_k = U_{i,j}$ when $k = i + (j-1)N$.

Using this numbering scheme, the finite difference method turn into the following linear system.

$$A\mathbf{U} = \mathbf{f}$$

$A \in \mathbb{R}^{N^2 \times N^2}$ is the block matrix

$$A = \begin{bmatrix} T & S & & & \\ S & T & S & & \\ & \ddots & \ddots & \ddots & \\ & & S & T & S \\ & & & S & T \end{bmatrix}$$

$$T = \begin{bmatrix} 20 & -4 & & & \\ -4 & 20 & -4 & & \\ & \ddots & \ddots & \ddots & \\ & & -4 & 20 & -4 \\ & & & -4 & 20 \end{bmatrix}$$

$$S = \begin{bmatrix} -4 & -1 & & & \\ -1 & -4 & -1 & & \\ & \ddots & \ddots & \ddots & \\ & & -1 & -4 & -1 \\ & & & -1 & -4 \end{bmatrix}$$

$$\mathbf{f} = \begin{bmatrix} 6h^2 f(x_1, y_1) + h^4/2\Delta f(x_1, y_1) + g(x_1, y_0) + g(x_0, y_1) \\ 6h^2 f(x_2, y_1) + h^4/2\Delta f(x_2, y_1) + g(x_2, y_0) \\ \vdots \\ 6h^2 f(x_{N-1}, y_N) + h^4/2\Delta f(x_{N-1}, y_N) + g(x_{N-1}, y_{N+1}) \\ 6h^2 f(x_N, y_N) + h^4/2\Delta f(x_N, y_N) + g(x_{N+1}, y_N) + g(x_N, y_{N+1}) \end{bmatrix}$$

For the vector **f** the boundary conditions are present for any index k that appears on the boundary of the mesh, otherwise the entry of **f** is simply $6h^2 f(x_i, y_j) + h^4/2\Delta f(x_i, y_j)$.

2. Write a MATLAB code that constructs the sparse coefficient matrix A and the appropriate right-hand side vector **F**. NOTE: you will need to modify the right-hand side vector to include the appropriate Laplacian of the right-hand side function.

   The following code is a function that solve the 2D Poisson problem using the 9 Point Laplacian.

```
function [U, Ux, Uy] = Poisson2D_9PointLaplacian(f, g, L, N, fLaplacian)
    p = inputParser;
    p.addRequired('f', @Utils.isFunctionHandle);
    p.addRequired('g', @Utils.isFunctionHandle);
    p.addRequired('L', @(x) isnumeric(x) && x > 0);
    p.addRequired('N', @Utils.isInteger);
    p.addRequired('fLaplacian', @Utils.isFunctionHandle);
    p.parse(f, g, L, N, fLaplacian);

    h = L/(N+1);
    x = 0:h:L;
    y = 0:h:L;

    % create functions to swap between row-wise ordering and i,j ordering
    %kFun = @(i, j) i + (j-1)*N;
    iFun = @(k) mod(k-1,N)+1;
    jFun = @(k) floor((k-1)/N) + 1;

    k = 1:N^2;
    % create vectors for x and y positions for each entry in U
    Ux = x(iFun(k) + 1);
    Uy = y(jFun(k) + 1);

    % vector of forcing function values at each index k
    F = 6*h^2*(f(Ux, Uy)+h^2/12*fLaplacian(Ux, Uy));
    % add on boundary conditions to F
    % add bottom boundary
    kBottom = 1:N;
    F(kBottom) = F(kBottom) + 4*g(x(2:N+1),0) + g(x(1:N),0) + g(x(3:N+2),0);
    % add top boundary
    kTop = (N^2-N+1):N^2;
    F(kTop) = F(kTop) + 4*g(x(2:N+1), L) + g(x(1:N), L) + g(x(3:N+2),L);
    % add left boundary
    kLeft = 1:N:(N^2-N)+1;
    F(kLeft) = F(kLeft) + 4*g(0, y(2:N+1)) + g(0, y(1:N)) + g(0, y(3:N+2));
    % add right boundary
    kRight = N:N:N^2;
    F(kRight) = F(kRight) + 4*g(L, y(2:N+1)) + g(L, y(1:N)) + g(L, y(3:N+2));
    % each corner got a double boundary condition
    F(1) = F(1) - g(0, 0);
    F(N) = F(N) - g(L, 0);
    F(N^2 - N + 1) = F(N^2 - N + 1) - g(0, L);
    F(N^2) = F(N^2) - g(L, L);

    % build sparse matrix A
    % A is block tridiagonal with symmetric upper and lower diagonals
    e = ones(N, 1);
    % block on main diagonal
    T = spdiags([-4*e, 20*e, -4*e], [-1, 0, 1], N, N);
    % block for upper and lower diagonals
    S = spdiags([-1*e, -4*e, -1*e], [-1, 0, 1], N, N);
    % shape of main diagonals and off diagonals
    I = eye(N);
    O = spdiags([e, e], [-1, 1], N, N);
    A = kron(I, T) + kron(O, S);

    U = A\F';

    % change U from 1D vector to 2D matrix
    %U = vec2mat(U, N);
end
```

3. Using your code do a numerical convergence study for the following right-hand side forcing and exact solution:

$$f(x, y) = -1.25e^{x+.5y} \quad \text{and} \quad u(x, y) = e^{x+.5y}$$

Just use the built-in backslash operator in MATLAB to solve the linear system.

The following script uses the previous function to test the convergence.

```
f = @(x,y) -1.25*exp(x + .5*y);
u = @(x,y) exp(x + .5*y);
fLaplacian = @(x, y) -1.5625*exp(x + .5*y);
Error = [];
for N = [10*2.^(0:5)-1]
    [U, Ux, Uy] = Poisson2D_9PointLaplacian(f, u, 1, N, fLaplacian);
    uExact = u(Ux, Uy);
%     USquare = vec2mat(U,N);
%     uExactSquare = vec2mat(uExact,N);
%     figure;
%     surf(USquare);
%     hold on
%     surf(uExactSquare);
%     pause
    Error = [Error; Ux(2) - Ux(1), norm(U - uExact', inf)];

end
hRatios = Error(1:end-1,1)./Error(2:end,1);
errorRatios = Error(1:end-1,2)./Error(2:end,2);
order = log(errorRatios)./log(hRatios);
table(hRatios, errorRatios, order)
```

The scripts output is as follows.

```
>> H03_2

ans =

    hRatios    errorRatios     order
    -------    -----------    -------

    2          15.904          3.9914
    2          15.964          3.9968
    2          15.929          3.9936
    2          12.734          3.6706
    2          1.3226         0.40343
```

4. Discretize the above PDE using the 5-point Laplacian. Write a MATLAB code that generates the sparse coefficient matrix A for this discretization.

   This PDE can be discretized very similary to the square geometry with the 5-point Laplacian. In fact I will begin with the same discretization as in that problem which uses the natural row-wise ordering. To recall this discretization resulted in the following linear algebra problem.

$$A\mathbf{U} = \mathbf{f}$$

   $A \in \mathbb{R}^{N^2 \times N^2}$ is the block matrix

$$A = \begin{bmatrix} T & -I & & & \\ -I & T & -I & & \\ & \ddots & \ddots & \ddots & \\ & & -I & T & -I \\ & & & -I & T \end{bmatrix}$$

$$T = \begin{bmatrix} 4 & -1 & & & \\ -1 & 4 & -1 & & \\ & \ddots & \ddots & \ddots & \\ & & -1 & 4 & -1 \\ & & & -1 & 4 \end{bmatrix}$$

and I is the $N \times N$ identity matrix

$$\mathbf{f} = \begin{bmatrix} h^2 f(x_1, y_1) + g(x_1, y_0) + g(x_0, y_1) \\ h^2 f(x_2, y_1) + g(x_2, y_0) \\ \vdots \\ h^2 f(x_{N-1}, y_N) + g(x_{N-1}, y_{N+1}) \\ h^2 f(x_N, y_N) + g(x_{N+1}, y_N) + g(x_N, y_{N+1}) \end{bmatrix}$$

Now when considering the differences between the square geometry and the L-shaped geometry, it is clear that the only difference is that some discretization points now lie outside of the geometry or on the boundary. In order to take this into account, if U_i lies outside the L-shaped geometry, we can change the old equation governing this point to $U_i = 0$. This changes the matrix A so that the ith row is all zeros except for a 1 on the diagonal. In other words $A_{ii} = 1$ and $A_{ij} = 0$ for $j \neq i$. The other change that needs to take place is that now U_i cannot have a weight for any other nodes equation. This changes A so that the ith column is all zeroes except for the diagonal. Thus $A_{ii} = 1$ as before and $A_{ji} = 0$ for $j \neq i$. In essence the rows and columns corresponding to the points outside the L-shaped geometry are zeroed out. They are then replaced with equations settings these points to zero.

The final difference between this problem and the previous square geometry, is that the boundary conditions are set to zero. This means that $g(x, y) = 0$ for all points on the boundary. This difference is reflected in the right-hand side vector.

Lastly we need to identify the points outside the L-shaped geometry. First note that N must be odd so that discretization points lie on the boundaries at $x = .5$ and $y = .5$. Since N is odd, there exists some positive integer m such that $N = 2m + 1$. This integer m can be thought of as the number of discretization points between $y = 0$ and $y = .5$ or $x = .5$ and $x = 1$. The points on the boundary at $y = .5$ from $x = 0$ to $x = .5$ start with $mN + 1$ and end with $mN + 1 + m$. For rows $m + 1$ to N the first $m + 1$ points are not part of the geometry. Therefore indices of the form $mi + j$ for $i = m + 1, m + 2, \ldots, N$ and $j = 1, 2, \ldots m + 1$ are not included in the geometry.

All of these chages results in the following linear algebra problem.

$$A\mathbf{U} = \mathbf{f}$$

$A \in \mathbb{R}^{N^2 \times N^2}$ is the block matrix

$$A = \begin{bmatrix} T & -I & & & \\ -I & T & -I & & \\ & \ddots & \ddots & \ddots & \\ & & -I_n & T_n & -I_n \\ & & & -I_n & T_n \end{bmatrix}$$

$$T = \begin{bmatrix} 4 & -1 & & & \\ -1 & 4 & -1 & & \\ & \ddots & \ddots & \ddots & \\ & & -1 & 4 & -1 \\ & & & -1 & 4 \end{bmatrix}$$

$$T_n = \begin{bmatrix} 1 & 0 & & & \\ 0 & 1 & 0 & & \\ & \ddots & \ddots & \ddots & \\ & & -1 & 4 & -1 \\ & & & -1 & 4 \end{bmatrix}$$

$$I_n = \begin{bmatrix} 0 & & & & \\ & 0 & & & \\ & & \ddots & & \\ & & & 1 & \\ & & & & 1 \end{bmatrix}$$

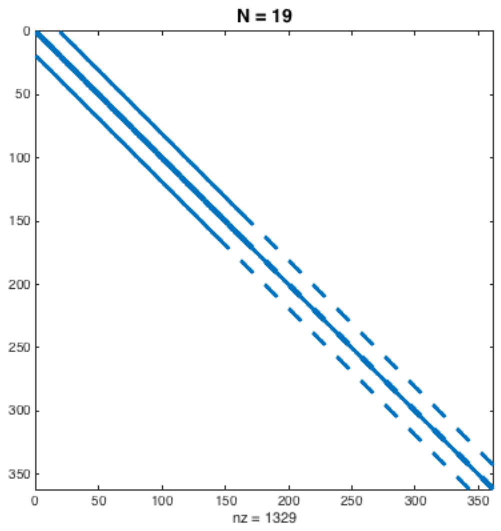
and I is the $N \times N$ identity matrix

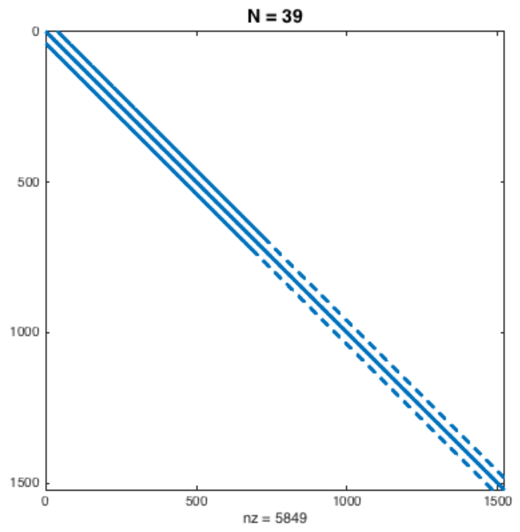
$$\mathbf{f} = \begin{bmatrix} h^2 f(x_1, y_1) \\ h^2 f(x_2, y_1) \\ \vdots \\ h^2 f(x_{N-1}, y_N) \\ h^2 f(x_N, y_N) \end{bmatrix}$$

In this case T_n and I_n make up the lower half of the diagonal, for the $m+1$ block and onward. In T_n the upper half is only 1s on the diagonal, the lower half is identical to T. I_n is zero on the digaonal for $i \leq m+1$, and the identity otherwise. The discretization is implemented in the following function.

```
function [U, Ux, Uy, A] = Poisson2D_5PointLaplacian_IrregularGeometry(f, L, N)
    p = inputParser;
    p.addRequired('f', @Utils.isFunctionHandle);
    p.addRequired('L', @(x) isnumeric(x) && x > 0);
    p.addRequired('N', @Utils.isOddInteger);
    p.parse(f, L, N);

    h = L/(N+1);
    x = 0:h:L;
    y = 0:h:L;
    m = ceil(N/2); % N = 2m + 1

    % indices not in geometry
    indices = N*(floor((0:m^2-1)/m) + N-m) + mod(0:m^2-1,m) + 1;
    % create functions to swap between row-wise ordering and i,j ordering
    %kFun = @(i, j) i + (j-1)*N;
    iFun = @(k) mod(k-1,N)+1;
    jFun = @(k) floor((k-1)/N) + 1;

    k = 1:N^2;
    % create vectors for x and y positions for each entry in U
    Ux = x(iFun(k) + 1);
    Uy = y(jFun(k) + 1);

    % vector of forcing function values at each index k
    F = h^2*f(Ux, Uy);
    F(indices) = 0;
    % boundary conditions are zero

    % build sparse matrix A
    % A is block tridiagonal with symmetric upper and lower diagonals
    e = ones(N, 1);
    % block on main diagonal
    T = spdiags([-e, 4*e, -e], [-1, 0, 1], N, N);
    % shape of main diagonal and off diagonals
    I = eye(N);
    O = spdiags([e, e], [-1, 1], N, N);
    A = kron(I, T) + kron(O, -I);

    % because of irregular geometry need to zero out indices in top left corner
    A(:, indices) = 0;
    A(indices, :) = 0;
    for i=indices
        A(i,i) = 1;
    end

    U = A\F';

    % change U from 1D vector to 2D matrix
    %U = vec2mat(U, N);
end
```

5. For $N = 19$ and $N = 39$ produce a spy plot of the matrix.



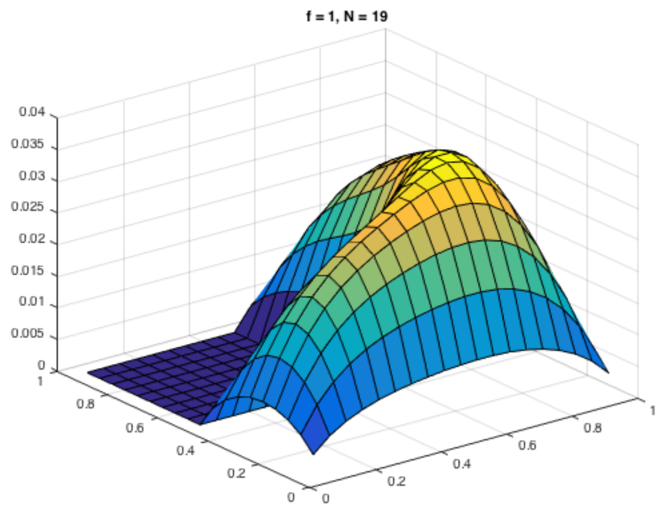


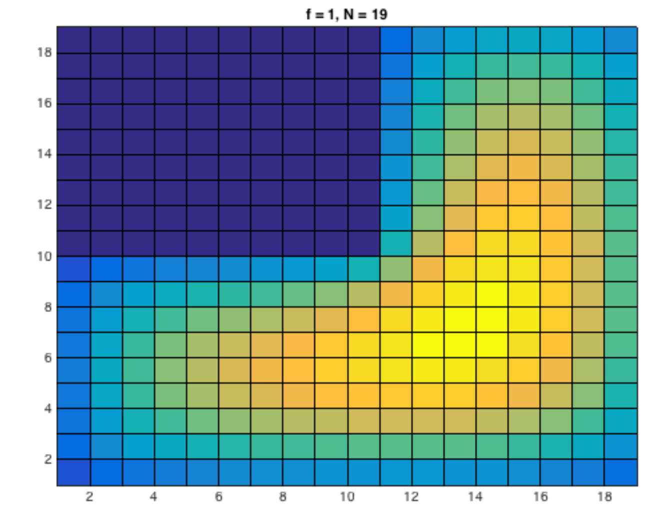
6. Solve the PDE using your code with the right hand side

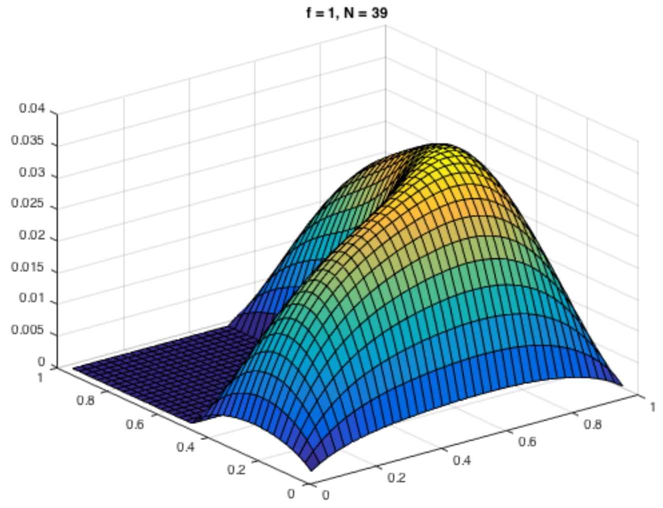
$$f(x, y) = 1.$$

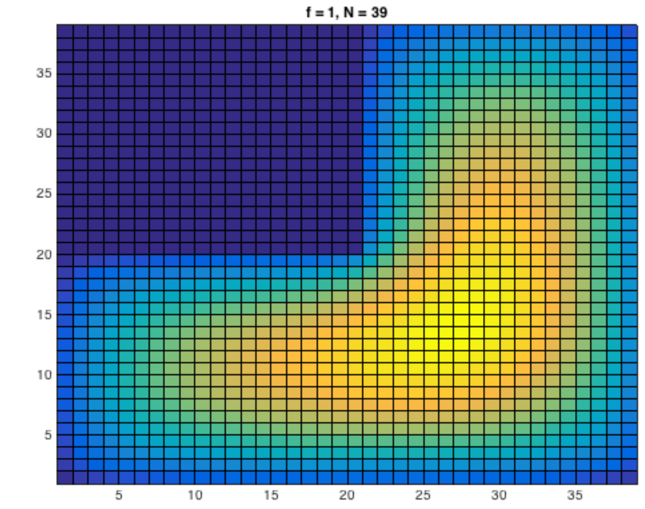
The following code evalates the function from problem 4 for the given right hand side. The images produced are shown below.

```
% Problem 6
f = @(x,y) ones(size(x));
for N = [19, 39]
    [U, Ux, Uy, A] = Poisson2D_5PointLaplacian_IrregularGeometry(f, 1, N);
    USquare = vec2mat(U,N);
    figure;
    surf(vec2mat(Ux, N), vec2mat(Uy, N), USquare);
    title(['f = 1, N = ',num2str(N)])
    figure
    pcolor(USquare);
    title(['f = 1, N = ',num2str(N)])
end
```







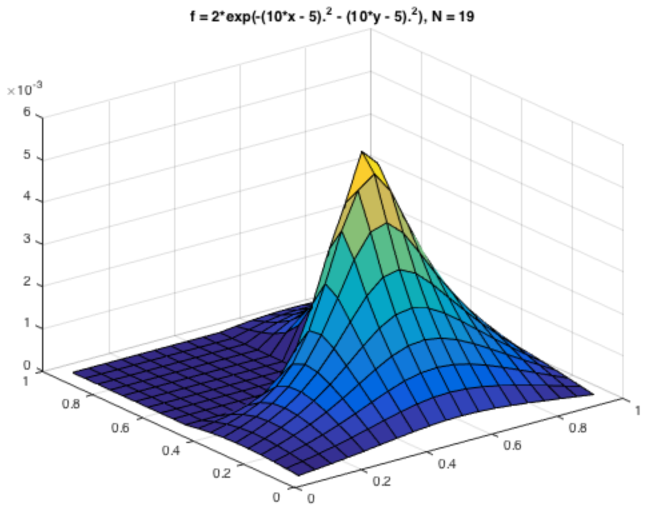


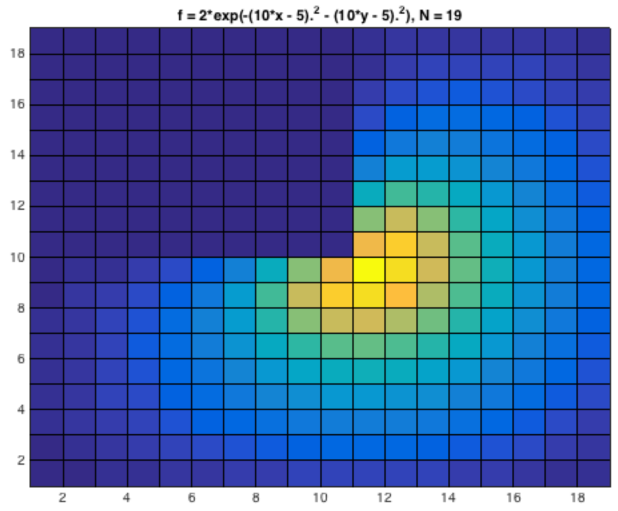
7. Solve the PDE using your code with the right hand side

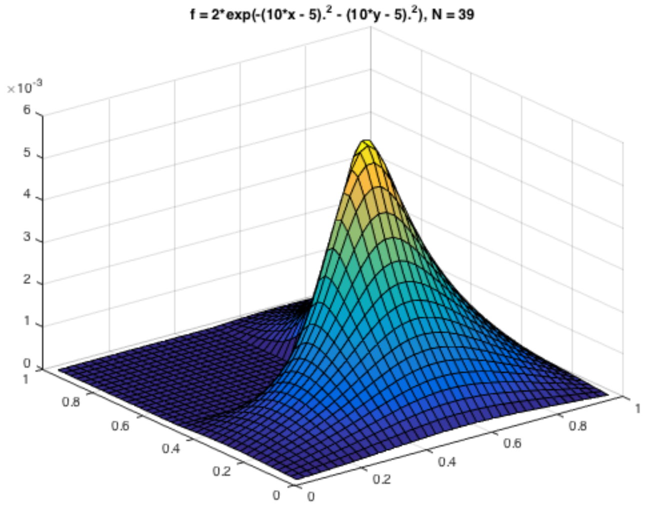
$$f(x, y) = 2exp\Big[-(10x - 5)^2 - (10y - 5)^2\Big].$$

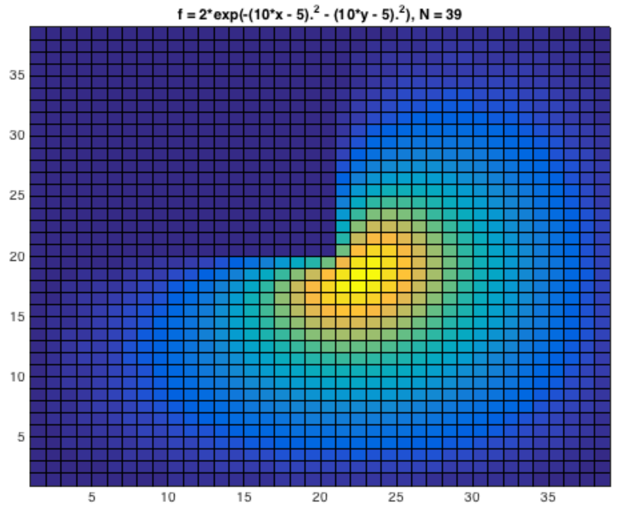
The following code evalates the function from problem 4 for the given right hand side. The images produced are shown below.

```
% Problem 7
f = @(x,y) 2*exp(-(10*x - 5).^2 - (10*y - 5).^2);
for N = [19, 39]
    [U, Ux, Uy, A] = Poisson2D_5PointLaplacian_IrregularGeometry(f, 1, N);
    USquare = vec2mat(U,N);
    figure;
    surf(vec2mat(Ux, N), vec2mat(Uy, N), USquare);
    title(['f = 2*exp(-(10*x - 5).^2 - (10*y - 5).^2), N = ',num2str(N)])
    figure
    pcolor(USquare);
    title(['f = 2*exp(-(10*x - 5).^2 - (10*y - 5).^2), N = ',num2str(N)])
end
```







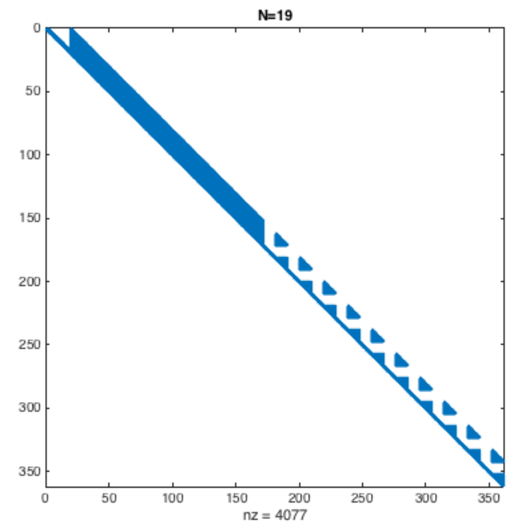


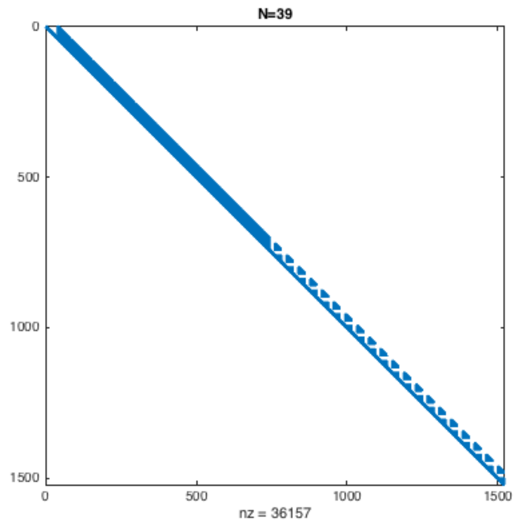
8. Compute the Cholesky factorization of A is MATLAB: R = chol(A); where R is an upper triangular matrix such that $A = R^T R$. For $N = 19$ and $N = 39$ produce a spy plot of R. Create a table showing the non-zeros in R for $N = 9, 19, 39, 79, 159, 319$.

   The following code is for problems 8 and 9.

```
% Problem 8 and 9
countsA = [];
countsB = [];
N = 10*2.^(0:5) - 1;
for iN = N
    [U, Ux, Uy, A] = Poisson2D_5PointLaplacian_IrregularGeometry(f, 1, iN);
    R = chol(A);
    % spy plots
    if(iN == 19 || iN == 39)
        figure
        spy(R);
        title(strcat('N= ',num2str(iN)));
    end
    countsA = [countsA; nnz(R)];
    P = symrcm(A);
    B = A(P, P);
    R = chol(B);
    if(iN == 19 || iN == 39)
        figure
        spy(R);
        title(strcat('Reverse Cuthill-Mckee, N= ',num2str(iN)));
    end
    countsB = [countsB; nnz(R)];
end
table(N', countsA)
table(N', countsB)
```

The following is the spy plots and table for problem 8.

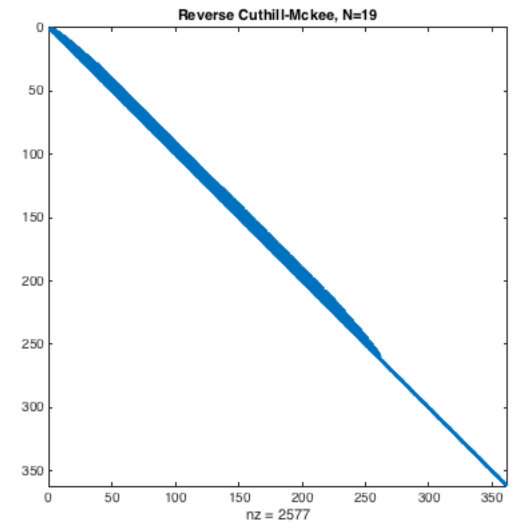


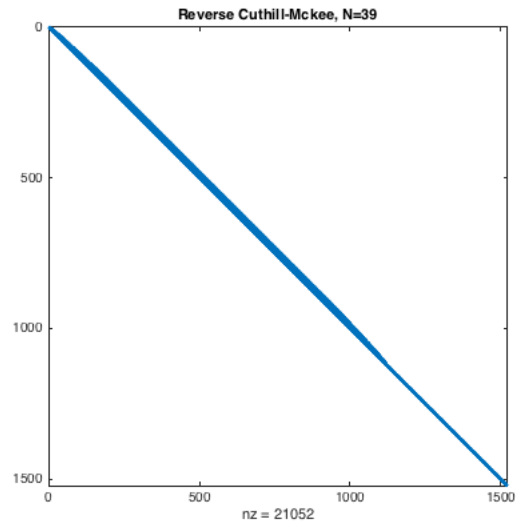


```
ans =

    Var1     countsA
    ----    ----------

    9            412
    19           4077
    39          36157
    79     3.0432e+05
    159     2.4966e+06
    319     2.0225e+07
```

9. Permute the matrix A using the reverse Cuthill-Mckee algorithm in MATLAB: P = symrcm(A); B = A(P, P); Compute the Cholesky factorization of B in MATLAB: R = chol(B); For $N = 19$ and $N = 39$ produce a spy plot of R. Create a table showing the non-zeros in R for $N = 9, 19, 39, 79, 159, 319$.

   The following is the spy plots and table for problem 9 generated by the code shown under problem 8.





```
ans =

    Var1     countsB
    ----    ----------
```

```
9            302
19           2577
39          21052
79       1.697e+05
159      1.3618e+06
319      1.0909e+07
```