Mario L. Gutierrez Abed
364009832
mlg3843@rit.edu

Problem Set 2
Numerical Methods for PDEs

03-08-2021

Problem 1. Consider the following linear boundary value problem with Dirichlet boundary conditions:

$$u''(x) + u(x) = 0 \quad \text{for } a < x < b$$
$$u(a) = \alpha, \quad u(b) = \beta.$$

This equation arises from a linearized pendulum, for example.

a) *Consider the problem with*

$$a = 0, \quad b = 1, \quad \alpha = 2, \quad \beta = 3.$$

Find the exact solution of the problem. Run the script `bvp_2.m` *and provide a table of errors in L^∞ norm for mesh sizes $h = 1/10, 1/20, 1/40$ and $1/80$. Show that your method is second order accurate (by either estimating the slope of the log-log plot of h versus the norm of the error or by showing the ratios of errors $||e_k||/||e_{k+1}|| = 4$). You can use the script* `error_table.m` *and/or* `error_loglog.m` *to produce tables with error values and approximate orders of convergence for this problem (and other problems).*

b) *Let $a = 0$ and $b = \pi$. For what values of α and β does this boundary value problem have solutions? What are the family of solutions in a case where there are infinitely many solutions?*

c) *Solve the problem with*

$$a = 0, \quad b = \pi, \quad \alpha = 1, \quad \beta = -1.$$

using `bvp_2.m`*. Which solution to the boundary value problem does this appear to converge to as $h \to 0$? Use the same testing procedure as described in part a).*

Change the boundary value at $b = \pi$ to $\beta = 1$. Now how does the numerical solution behave as $h \to 0$?

d) *You might expect the linear system in part c) to be singular since the boundary value problem is not well-posed. Compute the eigenvalues of the matrix A (i.e. matrix of the resulting linear system) for this problem and show that an eigenvalue approaches 0 as $h \to 0$. Also show that $\|A^{-1}\|_2$ blows up as $h \to 0$.*

Solution to a). The general solution to the equation is of the form

$$u(x) = c_1 \cos x + c_2 \sin x. \tag{1}$$

From $u(0) = 2$ we gather

$$u(0) = 2 = c_1 \cos 0 + c_2 \sin 0 = c_1 \implies c_1 = 2.$$

Furthermore, from $u(1) = 3$ we get

$$u(1) = 3 = 2\cos 1 + c_2 \sin 1 \implies c_2 = \frac{3 - 2\cos 1}{\sin 1} = 3\csc 1 - 2\cot 1.$$

Hence, the closed form solution of the BVP is

$$\boxed{u(x) = 2\cos x + (3\csc 1 - 2\cot 1)\sin x} \tag{2}$$

The table for errors of the algorithm presented in the `bvp_2.m` *script for the different mesh sizes is now provided:*

h	E
$1/10$	3.24687×10^{-4}
$1/20$	8.10848×10^{-5}
$1/40$	2.02705×10^{-5}
$1/80$	5.06999×10^{-6}

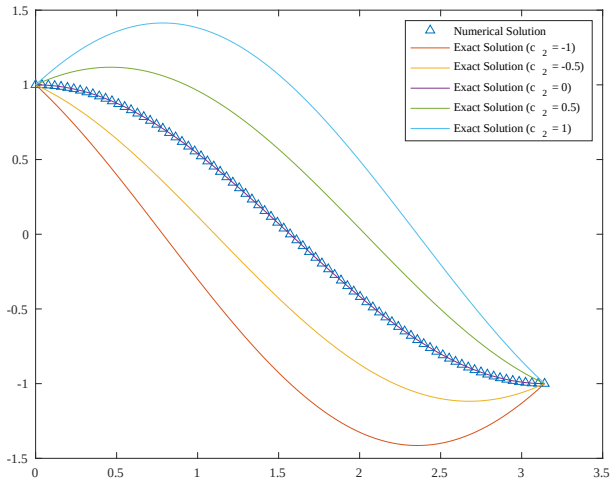
A least-squares fit yields $E(h) = 0.0324764h^{2.00028}$, which shows that the method is $O(h^2)$. □

Solution to b). Plugging $a = 0$ and $b = \pi$ into Eq. (1) yields $\alpha = c_1$, $\beta = -\alpha$, and c_2 is left arbitrary. We conclude that the BVP has an infinite number of solutions for $c_2 \in \mathbb{R}$ arbitrary and for all $\alpha, \beta \in \mathbb{R}$ that satisfy $\alpha + \beta = 0$. □

Solution to c). As we concluded in part b), we have an infinite number of solutions of the form

$$u(x) = \alpha \cos x + c_2 \sin x, \tag{3}$$

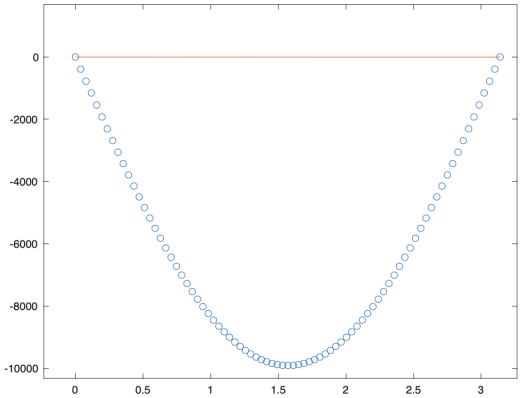
where $c_2 \in \mathbb{R}$ is arbitrary. Plotting this solution for all possible values of c_2 would fill up the entire canvas, of course, since there are infinitely many possibilities. Instead let us plot the exact solution (along with the numerical one) for some values of c_2, say, $c_2 \in \{-1, -0.5, 0, 0.5, 1\}$. The following plot shows that the numerical solution converges to the solution (3) with $c_2 = 0$:



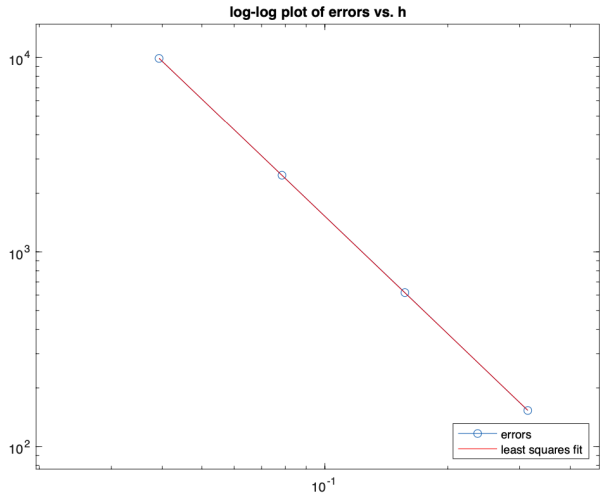
The errors are presented in the following tabel. A least-squares fit $E(h) = 0.0233814h^{2.00026}$ confirms that the method is $O(h^2)$:

h	E
0.31416	2.31489×10^{-3}
0.15708	5.73244×10^{-4}
0.07854	1.44354×10^{-4}
0.03927	3.60616×10^{-5}

Now, if we changed the boundary value at $b = \pi$ to $\beta = 1$, we know that this is nonsensical since $\alpha = 1$ and we have the condition $\beta = -\alpha$. At the numerical level what happens if we do this is that we have a cosine function with a rapidly-growing amplitude, as the figure indicates:



As we let $h \to 0$ the "solution" is blowing up to $-\infty$. In fact, the errors are larger with a smaller h, which is the opposite of what we would expect from a well-posed problem:



□

Solution to d). In Leveque's code A is saved as a sparse matrix; I simply used

```
    disp( ['for h = ', num2str(h), ' the smallest eigenvalue (in magnitude) is ',
          num2str(min(abs(eig(full(A)))))])
```

which has output

```
for h = 0.31416 the smallest eigenvalue (in magnitude) is 0.0081977
for h = 0.15708 the smallest eigenvalue (in magnitude) is 0.0020545
for h = 0.07854 the smallest eigenvalue (in magnitude) is 0.00051394
for h = 0.03927 the smallest eigenvalue (in magnitude) is 0.0001285
```

This shows that as $h \to 0$, the smallest eigenvalue (in magnitude) is also approaching 0. Similarly, adding the snippet

```
    disp( ['for h = ',num2str(h),' the 2-norm of A^{-1} is ',
          num2str(norm(inv(full(A)),2))])
```

yields

```
for h = 0.31416 the 2-norm of A^{-1} is 270.6151
for h = 0.15708 the 2-norm of A^{-1} is 1463.4067
for h = 0.07854 the 2-norm of A^{-1} is 8064.5131
for h = 0.03927 the 2-norm of A^{-1} is 44977.653
```

which shows that as $h \to 0$, we have $\|A^{-1}\|_2 \to \infty$. This was to be expected since $\|A^{-1}\|_2 = \rho(A^{-1}) = 1/\lambda_{min}$, where λ_{min} is the minimum eigenvalue (in magnitude) of A, which as we saw gets close to 0 as h decreases. □

The Kronecker Product and ∇_5^2

As we saw in Leveque's text, the 5-point stencil for Poisson's equation

$$\nabla^2 u = f$$

with $\Delta x = \Delta y = h$ leads to

$$\frac{u_{i-1,j} + u_{i+1,j} + u_{i,j-1} + u_{i,j+1} - 4u_{i,j}}{h^2} = f_{i,j}. \tag{4}$$

From this discretization we get the matrix

$$A = \frac{1}{h^2}\begin{bmatrix} T & I & & & \\ I & T & I & & \\ & I & T & I & \\ & & \ddots & \ddots & \ddots \\ & & & I & T \end{bmatrix}, \tag{5}$$

where I is the $m \times m$ identity matrix and T is the $m \times m$ matrix containing the coefficients of the $u_{i,j}$:

$$T = \begin{bmatrix} -4 & 1 & & & \\ 1 & -4 & 1 & & \\ & 1 & -4 & 1 & \\ & & \ddots & \ddots & \ddots \\ & & & 1 & -4 \end{bmatrix}. \tag{6}$$

We can get this matrix A by using either of the following Kronecker tensor products:

$$A = \frac{1}{h^2}\left(I \otimes_{\mathbb{k}} T + S \otimes_{\mathbb{k}} I\right) \tag{7a}$$

$$A = \frac{1}{h^2}\left(\mathcal{T} \otimes_{\mathbb{k}} I + I \otimes_{\mathbb{k}} \mathcal{T}\right), \tag{7b}$$

where S is a sparse matrix with 1's in both subdiagonals and 0's elsewhere, and $\mathcal{T}$ is identical to T with the exception that we have -2 along the diagonal as opposed to -4 (this will be crucial later when we consider $\Delta x \neq \Delta y$). Let us show that both of these expressions yield the matrix (5). For clarity, let us consider $m = 3$. Then, for (7a),

$$I \otimes_{\mathbb{k}} T = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \otimes_{\mathbb{k}} \begin{bmatrix} -4 & 1 & 0 \\ 1 & -4 & 1 \\ 0 & 1 & -4 \end{bmatrix}$$

$$= \left[\begin{array}{ccc:ccc:ccc} -4 & 1 & 0 & & & & & & \\ 1 & -4 & 1 & & \mathbf{0} & & & \mathbf{0} & \\ 0 & 1 & -4 & & & & & & \\ \hdashline & & & -4 & 1 & 0 & & & \\ & \mathbf{0} & & 1 & -4 & 1 & & \mathbf{0} & \\ & & & 0 & 1 & -4 & & & \\ \hdashline & & & & & & -4 & 1 & 0 \\ & \mathbf{0} & & & \mathbf{0} & & 1 & -4 & 1 \\ & & & & & & 0 & 1 & -4 \end{array}\right];$$

$$S \otimes_{\mathbb{k}} I = \begin{bmatrix} 0 & 1 & 0 \\ 1 & 0 & 1 \\ 0 & 1 & 0 \end{bmatrix} \otimes_{\mathbb{k}} \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} = \left[\begin{array}{c:c:c} \mathbf{0} & I & \mathbf{0} \\ \hdashline I & \mathbf{0} & I \\ \hdashline \mathbf{0} & I & \mathbf{0} \end{array}\right].$$

Thus we have

$$\frac{1}{h^2}\left(I \otimes_{\mathbb{k}} T + S \otimes_{\mathbb{k}} I\right) = \frac{1}{h^2}\left[\begin{array}{ccc:ccc:ccc} -4 & 1 & 0 & & & & & & \\ 1 & -4 & 1 & & I & & & \mathbf{0} & \\ 0 & 1 & -4 & & & & & & \\ \hdashline & & & -4 & 1 & 0 & & & \\ & I & & 1 & -4 & 1 & & I & \\ & & & 0 & 1 & -4 & & & \\ \hdashline & & & & & & -4 & 1 & 0 \\ & \mathbf{0} & & & I & & 1 & -4 & 1 \\ & & & & & & 0 & 1 & -4 \end{array}\right],$$

which is precisely in the form of the matrix (5). This implementation of the Kronecker product to compute the matrix A in MATLAB would be given by the following snippet:

```
% form matrix A:
I = speye(m);
e = ones(m,1);
T = spdiags([e -4*e e],[-1 0 1],m,m);
S = spdiags([e e],[-1 1],m,m);
A = (kron(I,T) + kron(S,I)) * (1/h^2);
```

Similarly, to show (7b), we have

$$\mathcal{T} \otimes_{\mathbb{k}} I = \begin{bmatrix} -2 & 1 & 0 \\ 1 & -2 & 1 \\ 0 & 1 & -2 \end{bmatrix} \otimes_{\mathbb{k}} \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

$$= \left[\begin{array}{ccc|ccc|ccc} -2 & 0 & 0 & & & & & & \\ 0 & -2 & 0 & & I & & & 0 & \\ 0 & 0 & -2 & & & & & & \\ \hline & & & -2 & 0 & 0 & & & \\ & I & & 0 & -2 & 0 & & I & \\ & & & 0 & 0 & -2 & & & \\ \hline & & & & & & -2 & 0 & 0 \\ & 0 & & & I & & 0 & -2 & 0 \\ & & & & & & 0 & 0 & -2 \end{array}\right];$$

$$I \otimes_{\mathbb{k}} \mathcal{T} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \otimes_{\mathbb{k}} \begin{bmatrix} -2 & 1 & 0 \\ 1 & -2 & 1 \\ 0 & 1 & -2 \end{bmatrix}$$

$$= \left[\begin{array}{ccc|ccc|ccc} -2 & 1 & 0 & & & & & & \\ 1 & -2 & 1 & & 0 & & & 0 & \\ 0 & 1 & -2 & & & & & & \\ \hline & & & -2 & 1 & 0 & & & \\ & 0 & & 1 & -2 & 1 & & 0 & \\ & & & 0 & 1 & -2 & & & \\ \hline & & & & & & -2 & 1 & 0 \\ & 0 & & & 0 & & 1 & -2 & 1 \\ & & & & & & 0 & 1 & -2 \end{array}\right].$$

Thus we have

$$\frac{1}{h^2}\left(\mathcal{T} \otimes_{\mathbb{k}} I + I \otimes_{\mathbb{k}} \mathcal{T}\right) = \frac{1}{h^2}\left[\begin{array}{ccc|ccc|ccc} -4 & 1 & 0 & & & & & & \\ 1 & -4 & 1 & & I & & & 0 & \\ 0 & 1 & -4 & & & & & & \\ \hline & & & -4 & 1 & 0 & & & \\ & I & & 1 & -4 & 1 & & I & \\ & & & 0 & 1 & -4 & & & \\ \hline & & & & & & -4 & 1 & 0 \\ & 0 & & & I & & 1 & -4 & 1 \\ & & & & & & 0 & 1 & -4 \end{array}\right],$$

which is also in the form of the matrix (5). The MATLAB implementation for this approach would be

```
% form matrix A:
I = speye(m);
e = ones(m,1);
T = spdiags([e -2*e e],[-1 0 1],m,m);
A = (kron(T,I) + kron(I,T)) * (1/h^2);
```

In the case where $\Delta x \neq \Delta y$, Eq. (4) is no longer valid; instead we have the discretization

$$\frac{u_{i-1,j} - 2u_{i,j} + u_{i+1,j}}{\Delta x^2} + \frac{u_{i,j-1} - 2u_{i,j} + u_{i,j+1}}{\Delta y^2} = f_{i,j}, \tag{10}$$

which now has matrix

$$A = \frac{1}{\Delta x^2}\begin{bmatrix} \mathcal{T} & I & & & \\ I & \mathcal{T} & I & & \\ & I & \mathcal{T} & I & \\ & & \ddots & \ddots & \ddots \\ & & & I & \mathcal{T} \end{bmatrix} + \frac{1}{\Delta y^2}\begin{bmatrix} \mathcal{T} & I & & & \\ I & \mathcal{T} & I & & \\ & I & \mathcal{T} & I & \\ & & \ddots & \ddots & \ddots \\ & & & I & \mathcal{T} \end{bmatrix}, \tag{11}$$

where $\mathcal{T}$ is, as defined earlier,

$$\mathcal{T} = \begin{bmatrix} -2 & 1 & & & \\ 1 & -2 & 1 & & \\ & 1 & -2 & 1 & \\ & & \ddots & \ddots & \ddots \\ & & & 1 & -2 \end{bmatrix}. \tag{12}$$

The MATLAB implementation for this case, which shall be used on Problem 2c), is given by

```
% form matrix A:
Ix = speye(my);          %my= number of y grid-pts
ex = ones(mx,1);         %mx= number of x grid-pts
Tx = spdiags([ex -2*ex ex],[-1 0 1],mx,mx) * (1/hx^2);

Iy = speye(mx);
ey = ones(my,1);
Ty = spdiags([ey -2*ey ey],[-1 0 1],my,my) * (1/hy^2);

A =  kron(Ix,Tx) + kron(Ty,Iy);
```

Problem 2. The MATLAB script `poisson.m` *solves the Poisson problem on a square $m \times m$ grid with $\Delta x = \Delta y = h$, using the 5-point Laplacian. It is set up to solve a test problem for which the exact solution is $u(x,y) = \exp(x + y/2)$ using Dirichlet boundary conditions and the right hand side $f(x,y) = 1.25\exp(x + y/2)$.*

a) Test this script by performing a grid refinement study to verify that it is second order accurate. Make a table of errors using the L^∞ and L^2 norms for $\Delta x = \Delta y = h = 1/25, 1/50$, and $1/100$.

b) Modify the script so that it works on a rectangular domain $[a_x, b_x] \times [a_y, b_y]$, but still with $\Delta x = \Delta y = h$. Test your modified script on a non-square domain by solving the equation

$$\frac{\partial^2 u}{\partial x^2} + \frac{\partial^2 u}{\partial y^2} = 0 \quad \text{on} \quad R = \{(x,y) = 1 < x < 2,\ 0 < y < 1\}$$

with the exact solution $u(x,y) = \ln(x^2 + y^2)$. Make a table of errors in the L^∞ norm for $\Delta x = \Delta y = h = 1/25, 1/50$, and $1/100$. Note: You don't need to specify the Dirichlet boundary conditions, the code is set up to extract these values automatically from the solution $u(x,y)$ that you provide.

c) Further modify the code to allow $\Delta x \neq \Delta y$ and test the modified script. You can solve the problem from part a) with solution $u(x,y) = \exp(x + y/2)$. Make a table of errors using the L^∞ and L^2 norms with $\Delta x = 1/25, 1/50$, and $1/100$ and $\Delta y = 1/20, 1/40$, and $1/80$.

Solution to a). Looping over the three given h values and considering the solutions (exact and numerical) as $(m+2)^2$ vectors, from

```
usoln = reshape(usoln, (m+2)*(m+2), 1);
utrue = reshape(utrue, (m+2)*(m+2), 1);

disp(['For h = ', num2str(h),' the L^2 norm is ',num2str(h*norm(usoln-utrue))])
disp(['For h = ', num2str(h),' the L^inf norm is ',num2str(max(abs(usoln-utrue)))])
```

we get

```
For h = 0.04 the L^2 norm is 1.2815e-05
For h = 0.04 the L^inf norm is 2.3087e-05
For h = 0.02 the L^2 norm is 3.2074e-06
For h = 0.02 the L^inf norm is 5.7864e-06
For h = 0.01 the L^2 norm is 8.0207e-07
For h = 0.01 the L^inf norm is 1.4472e-06
```

Here's a table:

h	$\|E\|_2$	$\|E\|_\infty$
$1/25$	1.2815×10^{-5}	2.3087×10^{-5}
$1/50$	3.2074×10^{-6}	5.7864×10^{-6}
$1/100$	8.0207×10^{-7}	1.4472×10^{-6}

Choosing any two error values, say $^{h=1/50}\|E\|_2 = 3.2074 \times 10^{-6}$ *and* $^{h=1/25}\|E\|_2 = 1.2815 \times 10^{-5}$*, we see that cutting h in half yields*

$$\frac{3.2074 \times 10^{-6}}{1.2815 \times 10^{-5}} \approx \frac{1}{4},$$

which shows that the error is $\sim O(h^2)$*.* □

Solution to b). Here is the modified script:

```
%Rectangular (i.e., non-square) domain ( x \in [1,2]; y \in [0,1] )
ax = 1;
bx = 2;
ay = 0;
by = 1;

M = [24, 49, 99];   %corresponds to h = 1/25, 1/50, and 1/100, respectively

for m = M
    h = (bx-ax)/(m+1);
    x = linspace(ax,bx,m+2);   % grid points x including boundaries
    y = linspace(ay,by,m+2);   % grid points y including boundaries


    [X,Y] = meshgrid(x,y);      % 2d arrays of x,y values
    X = X';                     % transpose so that X(i,j),Y(i,j) are
    Y = Y';                     % coordinates of (i,j) point

    Iint = 2:m+1;              % indices of interior points in x
    Jint = 2:m+1;              % indices of interior points in y
    Xint = X(Iint,Jint);       % interior points
    Yint = Y(Iint,Jint);

    utrue = log(X.^2 + Y.^2);

    % set boundary conditions around edges of usoln array:
    usoln = utrue;              % use true solution for this test problem
                                % This sets full array, but only boundary values
                                % are used below.  For a problem where utrue
                                % is not known, would have to set each edge of
                                % usoln to the desired Dirichlet boundary values.

    rhs  = zeros(m,m);
    % adjust the rhs to include boundary terms:
```

```
    rhs(:,1) = rhs(:,1) - usoln(Iint,1)/h^2;
    rhs(:,m) = rhs(:,m) - usoln(Iint,m+2)/h^2;
    rhs(1,:) = rhs(1,:) - usoln(1,Jint)/h^2;
    rhs(m,:) = rhs(m,:) - usoln(m+2,Jint)/h^2;

    % convert the 2d grid function rhs into a column vector for rhs of system:
    F = reshape(rhs,m*m,1);

    % form matrix A:
    I = speye(m);
    e = ones(m,1);
    T = spdiags([e -4*e e],[-1 0 1],m,m);
    S = spdiags([e e],[-1 1],m,m);
    A = (kron(I,T) + kron(S,I)) / h^2;

    % Solve the linear system:
    uvec = A\F;

    usoln(Iint,Jint) = reshape(uvec,m,m);

    usoln = reshape(usoln, (m+2)*(m+2), 1);
    utrue = reshape(utrue, (m+2)*(m+2), 1);

    disp(['For h = ',num2str(h),' the L^inf norm is ',num2str(max(abs(usoln-utrue)))])
end
```

The output is shown in the table:

h	$\|E\|_\infty$
$1/25$	2.1634×10^{-5}
$1/50$	5.4282×10^{-6}
$1/100$	1.3585×10^{-6}

Choosing any two error values, say $^{h=1/100}\|E\|_\infty = 1.3585 \times 10^{-6}$ *and* $^{h=1/50}\|E\|_\infty = 5.4282 \times 10^{-6}$*, we see that cutting* h *in half yields*

$$\frac{1.3585 \times 10^{-6}}{5.4282 \times 10^{-6}} \approx \frac{1}{4},$$

which confirms that the error is $\sim O(h^2)$*.* □

Solution to c). Here is the modified script:

```
%Rectangular (i.e., non-square) domain ( x \in [1,2]; y \in [0,1] )
ax = 1;
bx = 2;
ay = 0;
by = 1;

Mx = [24, 49, 99];   %corresponds to hx = 1/25, 1/50, and 1/100, respectively
My = [19, 39, 79];   %corresponds to hy = 1/20, 1/5=40, and 1/80, respectively

for mx = Mx
    for my = My
        hx = (bx-ax)/(mx+1);
        hy = (by-ay)/(my+1);
        x = linspace(ax,bx,mx+2);   % grid points x including boundaries
        y = linspace(ay,by,my+2);   % grid points y including boundaries


        [X,Y] = meshgrid(x,y);      % 2d arrays of x,y values
        X = X';                     % transpose so that X(i,j),Y(i,j) are
        Y = Y';                     % coordinates of (i,j) point

        Iint = 2:mx+1;              % indices of interior points in x
        Jint = 2:my+1;              % indices of interior points in y
        Xint = X(Iint,Jint);        % interior points
        Yint = Y(Iint,Jint);

```

```
        f = @(x,y) 1.25*exp(x+y/2);
        rhs = f(Xint,Yint);
        utrue = exp(X+Y/2);        % true solution for test problem

        % set boundary conditions around edges of usoln array:

        usoln = utrue;

        % adjust the rhs to include boundary terms:
        rhs(:,1) = rhs(:,1) - usoln(Iint,1)/hy^2;
        rhs(:,my) = rhs(:,my) - usoln(Iint,my+2)/hy^2;
        rhs(1,:) = rhs(1,:) - usoln(1,Jint)/hx^2;
        rhs(mx,:) = rhs(mx,:) - usoln(mx+2,Jint)/hx^2;

        % convert the 2d grid function rhs into a column vector for rhs of system:
        F = reshape(rhs,mx*my,1);

        % form matrix A:
        Ix = speye(my);
        ex = ones(mx,1);
        Tx = spdiags([ex -2*ex ex],[-1 0 1],mx,mx) * (1/hx^2);

        Iy = speye(mx);
        ey = ones(my,1);
        Ty = spdiags([ey -2*ey ey],[-1 0 1],my,my) * (1/hy^2);

        A =  kron(Ix,Tx) + kron(Ty,Iy) ;

        % Solve the linear system:
        uvec = A\F;

        usoln(Iint,Jint) = reshape(uvec,mx,my);
        usoln = reshape(usoln, (my+2)*(mx+2), 1);
        utrue = reshape(utrue, (my+2)*(mx+2), 1);

        disp(['For hx = ',num2str(hx),' and hy =  ',num2str(hy),' the L^inf norm is ',
              num2str( max(abs(usoln-utrue)) )])
        disp(['For hx = ',num2str(hx),' and hy =  ',num2str(hy),' the L^2 norm is ',
              num2str(norm(usoln-utrue,2) * sqrt(hx*hy))])
    end
end
```

The following table shows the output errors in the L^∞ norm:

	$\Delta x = 1/25$	$\Delta x = 1/50$	$\Delta x = 1/100$
$\Delta y = 1/20$	6.4847×10^{-5}	2.0559×10^{-5}	9.4738×10^{-6}
$\Delta y = 1/40$	6.0568×10^{-5}	1.6241×10^{-5}	5.1453×10^{-6}
$\Delta y = 1/80$	5.9527×10^{-5}	1.5167×10^{-5}	4.064×10^{-6}

Similarly, in the L^2 norm:

	$\Delta x = 1/25$	$\Delta x = 1/50$	$\Delta x = 1/100$
$\Delta y = 1/20$	3.5974×10^{-5}	1.1401×10^{-5}	5.2527×10^{-6}
$\Delta y = 1/40$	3.3602×10^{-5}	9.0062×10^{-6}	2.8529×10^{-6}
$\Delta y = 1/80$	3.3008×10^{-5}	8.407×10^{-6}	2.2523×10^{-6}

We see more of an impact on the error norms as we refine the x-grid compared to the refinement on the y-grid. I believe this may be due to the fact that the true solution $u = exp(x + y/2)$ has the y-term halved? □

The Kronecker Product and ∇_9^2

The 9-point stencil for Poisson's equation with $\Delta x = \Delta y = h$ ia given by

$$\frac{4u_{i-1,j} + 4u_{i+1,j} + 4u_{i,j-1} + 4u_{i,j+1} + u_{i-1,j-1} + u_{i+1,j+1} + u_{i-,j+1} + u_{i+1,j-1} - 20u_{i,j}}{6h^2} = f_{i,j}. \qquad (13)$$

From this discretization we get the matrix

$$A = \frac{1}{6h^2}\begin{bmatrix} T & H & & & \\ H & T & H & & \\ & H & T & H & \\ & & \ddots & \ddots & \ddots \\ & & & H & T \end{bmatrix}, \tag{14}$$

where T *is the* $m \times m$ *matrix*

$$T = \begin{bmatrix} -20 & 4 & & & \\ 4 & -20 & 4 & & \\ & 4 & -20 & 4 & \\ & & \ddots & \ddots & \ddots \\ & & & 4 & -20 \end{bmatrix} \tag{15}$$

and H *is the* $m \times m$ *matrix*

$$H = \begin{bmatrix} 4 & 1 & & & \\ 1 & 4 & 1 & & \\ & 1 & 4 & 1 & \\ & & \ddots & \ddots & \ddots \\ & & & 1 & 4 \end{bmatrix} \tag{16}$$

Doing similar Kronecker products as the ones we did before for the 5-point Laplacian, it can be easily shown that the matrix A *can be recovered from the expression*

$$A = \frac{1}{6h^2}\left(H \otimes_{\mathbb{K}} H - 36I \otimes_{\mathbb{K}} I\right). \tag{17}$$

Problem 3. Modify the MATLAB script `poisson.m` *to use the 9-point Laplacian instead of the 5-point Laplacian, and to solve the linear system*

$$\nabla_9^2 u_{ij} = f_{ij}, \tag{18a}$$

where

$$f_{ij} = f(x_i, y_j) + \frac{h^2}{12}\nabla_5^2 f(x_i, y_j). \tag{18b}$$

Test your code for the problem with exact solution $u(x, y) = \sin(\pi x)\sin(\pi y)$ *and perform a grid refinement study to verify that fourth order accuracy is achieved. Use the* L^2 *norm or* L^∞ *norm for the errors. Attach one surface plot of the solution.*

Solution. Before we get to code, we need to first find f*. We know the exact solution* u *so we can just plug into the Poisson equation and find* f*, either analytically or numerically using the 5-point stencil previously studied. Since it is a fairly simple equation, let us tackle it analytically:*

$$f = \nabla^2 u = \partial_x^2 u + \partial_y^2 u = \partial_x^2\left[\sin(\pi x)\sin(\pi y)\right] + \partial_y^2\left[\sin(\pi x)\sin(\pi y)\right] = -2\pi^2 \sin(\pi x)\sin(\pi y).$$

Moreoever, by introducing the second term on the RHS of Eq. (18b) we will end up with a numerical solution that is fourth-order convergent. Let us expand this expression:

$$\frac{h^2}{12}\nabla_5 f(x_i, y_j) = \frac{h^2}{12} \cdot \frac{f(x_i - h, y_j) + f(x_i + h, y_j) + f(x_i, y_j - h) + f(x_i, y_j + h) - 4f(x_i, y_j)}{h^2}$$
$$= \frac{f(x_i - h, y_j) + f(x_i + h, y_j) + f(x_i, y_j - h) + f(x_i, y_j + h) - 4f(x_i, y_j)}{12}.$$

Then, plugging into Eq. (18b), we get

$$f_{ij} = \frac{f(x_i - h, y_j) + f(x_i + h, y_j) + f(x_i, y_j - h) + f(x_i, y_j + h) + 8f(x_i, y_j)}{12}. \tag{18c}$$

We are now ready to implement our code:

```
%9-point Laplacian code

%Square grid (defined in this way in case we want to modify to rectangular later)
ax = 0;
bx = 1;
ay = 0;
by = 1;

M = [24, 49, 99];   %corresponds to h = 1/25, 1/50, and 1/100, respectively

for m = M
    h = (bx-ax)/(m+1);
    x = linspace(ax,bx,m+2);   % grid points x including boundaries
    y = linspace(ay,by,m+2);   % grid points y including boundaries


    [X,Y] = meshgrid(x,y);      % 2d arrays of x,y values
    X = X';                     % transpose so that X(i,j),Y(i,j) are
    Y = Y';                     % coordinates of (i,j) point

    Iint = 2:m+1;              % indices of interior points in x
    Jint = 2:m+1;              % indices of interior points in y
    Xint = X(Iint,Jint);       % interior points
    Yint = Y(Iint,Jint);

    f = @(x,y) -2*pi^2 .* sin(pi .*x) .* sin(pi .*y);         % f(x,y) function

    %resulting RHS from induced error (see Eq 18c from assignment sheet)
    rhs = 1/12 * (f(Xint-h,Yint) + f(Xint+h,Yint) + f(Xint,Yint-h) + f(Xint,Yint+h)
          +8*f(Xint,Yint));

    utrue = sin(pi .* X) .* sin(pi .* Y);   % true solution for test problem

    % set boundary conditions around edges of usoln array:
    usoln = utrue;

    % adjust the rhs to include boundary terms:
    rhs(:,1) = rhs(:,1) - usoln(Iint,1)/h^2;
    rhs(:,m) = rhs(:,m) - usoln(Iint,m+2)/h^2;
    rhs(1,:) = rhs(1,:) - usoln(1,Jint)/h^2;
    rhs(m,:) = rhs(m,:) - usoln(m+2,Jint)/h^2;

    % convert the 2d grid function rhs into a column vector for rhs of system:
    F = reshape(rhs,m*m,1);

    %form matrix A:
    I = speye(m);
    e = ones(m,1);
    H = spdiags([e 4*e e],[-1 0 1],m,m);
    A = (kron(H,H) - 36 * kron(I,I))/(6*h^2);

    % Solve the linear system:
    uvec = A\F;

    usoln(Iint,Jint) = reshape(uvec,m,m);
    usoln = reshape(usoln, (m+2)*(m+2), 1);
    utrue = reshape(utrue, (m+2)*(m+2), 1);

```

```
    disp(['For h = ',num2str(h),' the L^2 norm is ',num2str(h*norm(usoln-utrue))])
    disp(['For h = ',num2str(h),' the L^inf norm is ',num2str( max(abs(usoln-utrue)))])
end
```

The output errors are shown in the table:

h	$\|E\|_\infty$	$\|E\|_2$
$1/25$	6.8931×10^{-7}	3.4602×10^{-7}
$1/50$	4.3283×10^{-8}	2.1641×10^{-8}
$1/100$	2.7054×10^{-9}	1.3527×10^{-9}

Choosing any two error values, say $^{h=1/50}\|E\|_\infty = 4.3283 \times 10^{-8}$ *and* $^{h=1/25}\|E\|_\infty = 6.8931 \times 10^{-7}$*, we see that cutting* h *in half yields*

$$\frac{4.3283 \times 10^{-8}}{6.8931 \times 10^{-7}} \approx \frac{1}{16},$$

which confirms that the error is $\sim O(h^4)$*. On the other hand, had we not introduced the error given by the second term on the RHS of Eq. (18b), we would have gotten only second-order convergence. I tested this in the code by replacing the expression given by* `rhs` *with* `rhs = f(Xint,Yint)`*. This would give the following table of errors*

h	$\|E\|_\infty$	$\|E\|_2$
$1/25$	0.0026243	0.0013173
$1/50$	0.00065815	0.00032907
$1/100$	0.0001645	8.2252×10^{-5}

Choosing any two error values, say $^{h=1/50}\|E\|_\infty = 0.00065815$ *and* $^{h=1/25}\|E\|_\infty = 0.0026243$*, we see that cutting* h *in half yields*

$$\frac{0.00065815}{0.0026243} \approx \frac{1}{4},$$

which confirms that the error is $\sim O(h^2)$*.*

We now conclude the exercise by showing a surface plot of the solution:

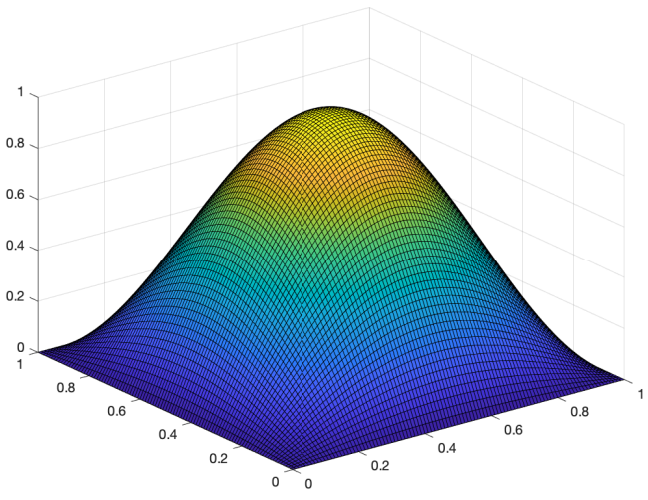


Figure 1: Numerical solution of the Poisson equation $\nabla^2 u = -2\pi^2 \sin(\pi x) \sin(\pi y)$*.*

□