Problem 1

1. Show

$$\boldsymbol{\Delta}_+^s z(x) = \sum_{i=0}^{s} (-1)^{s-i} \binom{s}{i} z(x+ih)$$

Proof. If $s=1$, $\boldsymbol{\Delta}_+ z(x) = -z(x) + z(x+h) = \sum_{i=0}^{s} (-1)^{s-i}\binom{s}{i} z(x+ih)$. If $s>1$, assume

$$\boldsymbol{\Delta}_+^{s-1} z(x) = \sum_{i=0}^{s-1} (-1)^{s-1-i} \binom{s-1}{i} z(x+ih)$$

Therefore,

$$\begin{aligned}
\boldsymbol{\Delta}_+^s z(x) &= \boldsymbol{\Delta}_+ \left(\boldsymbol{\Delta}_+^{s-1} z(x)\right) \\
&= \boldsymbol{\Delta}_+ \left(\sum_{i=0}^{s-1} \left((-1)^{s-1-i} \binom{s-1}{i} z(x+ih) \right) \right) \\
&= \sum_{i=0}^{s-1} \left((-1)^{s-1-i} \binom{s-1}{i} z(x+(i+1)h) \right) - \sum_{i=0}^{s-1} \left((-1)^{s-1-i} \binom{s-1}{i} z(x+ih) \right) \\
&= \left[\sum_{j=1}^{s-1} \left((-1)^{s-j} \binom{s-1}{j-1} z(x+jh) \right) + (-1)^{s-s} \binom{s-1}{s-1} z(x+sh) \right] \\
&\quad - (-1) \left[\sum_{j=1}^{s-1} \left((-1)^{s-j} \binom{s-1}{j} z(x+jh) \right) - (-1)^s z(x) \right] \\
&= \sum_{j=1}^{s-1} \left((-1)^{s-j} \left[\binom{s-1}{j-1} + \binom{s-1}{j} \right] z(x+jh) \right) + z(x+sh) + (-1)^s z(x) \\
&= \sum_{i=0}^{s} (-1)^{s-i} \binom{s}{i} z(x+ih)
\end{aligned}$$

where last step is given by the recurrence relation

$$\binom{s}{j} = \binom{s-1}{j-1} + \binom{s-1}{j}$$

□

2. Show

$$\sum_{i=0}^{s} (-1)^{s-i} \binom{s}{i} i^j = 0$$

3. Give a rigorous, shorter proof of $\boldsymbol{\Delta}_+^s z(x) = \mathcal{O}(h^s)$

Proof. Given $z(x)$ has s continuous derivatives, use mean value theorem repeatedly. We claim that

$$\boldsymbol{\Delta}_+^s z(x) = h^s z^{(s)}(\eta^{(s)}) \qquad \text{for some} \qquad \eta^{(s)} \in (x, x+sh)$$

Use induction. When $s=1$,

$$\boldsymbol{\Delta}_+ z(x) = h \frac{z(x+h) - z(x)}{h} = h z^{(1)}(\eta^{(1)}) \qquad \eta^{(1)} \in (x, x+h)$$

when $s>1$ assume $\boldsymbol{\Delta}_+^{s-1} z(x) = h^{s-1} z^{(s-1)}(\eta^{(s-1)})$ for some $\eta^{(s-1)} \in (x, x+(s-1)h)$, then

$$\boldsymbol{\Delta}_+^s z(x) = \boldsymbol{\Delta}_+(\boldsymbol{\Delta}_+^{s-1} z(x)) = h^s \frac{z^{(s-1)}(\eta^{(s-1)}+h) - z^{(s-1)}(\eta^{(s-1)})}{h} = h^s z^{(s)}(\eta^{(s)})$$

for some $\eta^{(s)} \in (x, x+sh)$. □

Problem 2 (textbook 166 8.1)

Prove

$$\boldsymbol{\Delta}_- + \boldsymbol{\Delta}_+ = 2\boldsymbol{\Upsilon}_0\boldsymbol{\Delta}_0 \qquad \text{and} \qquad \boldsymbol{\Delta}_-\boldsymbol{\Delta}_+ = \boldsymbol{\Delta}_0^2$$

Proof. Note $\boldsymbol{\Delta}_- = \boldsymbol{\mathcal{I}} - \boldsymbol{\mathcal{E}}^{-1}$, $\boldsymbol{\Delta}_+ = \boldsymbol{\mathcal{E}} - \boldsymbol{\mathcal{I}}$, $\boldsymbol{\Delta}_0 = \boldsymbol{\mathcal{E}}^{1/2} - \boldsymbol{\mathcal{E}}^{-1/2}$, and $\boldsymbol{\Upsilon}_0 = 1/2(\boldsymbol{\mathcal{E}}^{1/2} + \boldsymbol{\mathcal{E}}^{-1/2})$. So

$$\begin{aligned}
\boldsymbol{\Delta}_- + \boldsymbol{\Delta}_+ &= \boldsymbol{\mathcal{I}} - \boldsymbol{\mathcal{E}}^{-1} + \boldsymbol{\mathcal{E}} - \boldsymbol{\mathcal{I}} \\
&= \boldsymbol{\mathcal{E}} - \boldsymbol{\mathcal{E}}^{-1} \\
&= (\boldsymbol{\mathcal{E}}^{1/2} + \boldsymbol{\mathcal{E}}^{-1/2})(\boldsymbol{\mathcal{E}}^{1/2} - \boldsymbol{\mathcal{E}}^{-1/2}) \\
&= 2\boldsymbol{\Upsilon}_0\boldsymbol{\Delta}_0
\end{aligned}$$

and

$$\begin{aligned}
\boldsymbol{\Delta}_-\boldsymbol{\Delta}_+ &= (\boldsymbol{\mathcal{I}} - \boldsymbol{\mathcal{E}}^{-1})(\boldsymbol{\mathcal{E}} - \boldsymbol{\mathcal{I}}) \\
&= \boldsymbol{\mathcal{E}} - \boldsymbol{\mathcal{I}} - \boldsymbol{\mathcal{I}} + \boldsymbol{\mathcal{E}}^{-1} \\
&= (\boldsymbol{\mathcal{E}}^{1/2} - \boldsymbol{\mathcal{E}}^{-1/2})^2 \\
&= \boldsymbol{\Delta}_0^2
\end{aligned}$$

□

Problem 3 (textbook 167 8.5)

Consider finite difference approximations to the derivative that use one point to the left and $s \geq 1$ points to the right of x

1. Determine constants α_j, $j = 1, 2, \cdots$ such that

$$\boldsymbol{\mathcal{D}} = \frac{1}{h}\left(\beta\boldsymbol{\Delta}_- + \sum_{j=1}^{\infty}\alpha_j\boldsymbol{\Delta}_+^j\right)$$

where $\beta \in \mathbb{R}$ is given.

Proof. Write the formula w.r.t. $\boldsymbol{\Delta}_+$ and use the following expansion to determine α_js

$$\boldsymbol{\Delta}_+ = \boldsymbol{\mathcal{E}} - \boldsymbol{\mathcal{I}} = \boldsymbol{\mathcal{I}} - e^{-h\boldsymbol{\mathcal{D}}} = \boldsymbol{\mathcal{I}} - \sum_{i=0}^{\infty}\frac{(-h\boldsymbol{\mathcal{D}})^i}{i!} = \sum_{i=1}^{\infty}\frac{(-h\boldsymbol{\mathcal{D}})^i}{i!}$$

Express $\boldsymbol{\Delta}_-$ as an expansion of $\boldsymbol{\Delta}_+$

$$\boldsymbol{\Delta}_- = \boldsymbol{\mathcal{I}} - \boldsymbol{\mathcal{E}}^{-1} = \boldsymbol{\mathcal{I}} - (\boldsymbol{\mathcal{I}} + \boldsymbol{\Delta}_+)^{-1} = \boldsymbol{\mathcal{I}} - \sum_{i=0}^{\infty}(-1)^i\boldsymbol{\Delta}_+^i = \sum_{i=1}^{\infty}(-1)^{i+1}\boldsymbol{\Delta}_+^i$$

Now we have

$$\begin{aligned}
h\boldsymbol{\mathcal{D}} &= \beta\boldsymbol{\Delta}_- + \sum_{j=1}^{\infty}\alpha_j\boldsymbol{\Delta}_+^j \\
&= \sum_{j=1}^{\infty}\left(\beta(-1)^{j+1} + \alpha_j\right)\boldsymbol{\Delta}_+^j
\end{aligned}$$

But note

$$h\boldsymbol{\mathcal{D}} = \ln(\boldsymbol{\mathcal{E}}) = \ln(\boldsymbol{\mathcal{I}} + \boldsymbol{\Delta}_+) = \sum_{j=1}^{\infty}\frac{(-1)^{j+1}}{j}\boldsymbol{\Delta}_+^j$$

Therefore,

$$\frac{(-1)^{j+1}}{j} = \beta(-1)^{j+1} + \alpha_j \qquad \Rightarrow \qquad \alpha_j = (-1)^{j-1}(\frac{1}{j} - \beta)$$

□

2. Given integer $s \geq 1$, show how to choose parameter β so that

$$\boldsymbol{\mathcal{D}} = \frac{1}{h}\left(\beta\boldsymbol{\Delta}_- + \sum_{j=1}^{s}\alpha_j\boldsymbol{\Delta}_+^j\right) + \mathcal{O}(h^{s+1}) \qquad h \to 0$$

Proof. From conclusion of previous, we can write

$$\begin{aligned}
\boldsymbol{\mathcal{D}} &= \frac{1}{h}\left(\beta\boldsymbol{\Delta}_- + \sum_{j=1}^{\infty}\alpha_j\boldsymbol{\Delta}_+^j\right) \\
&= \frac{1}{h}\left(\beta\boldsymbol{\Delta}_- + \sum_{j=1}^{s}\alpha_j\boldsymbol{\Delta}_+^j\right) + \frac{1}{h}\alpha_{s+1}\boldsymbol{\Delta}_+^{s+1} + \frac{1}{h}\sum_{j=s+2}^{\infty}\alpha_j\boldsymbol{\Delta}_+^j \\
&= \frac{1}{h}\left(\beta\boldsymbol{\Delta}_- + \sum_{j=1}^{s}\alpha_j\boldsymbol{\Delta}_+^j\right) + \frac{1}{h}\alpha_{s+1}\boldsymbol{\Delta}_+^{s+1} + \mathcal{O}(h^{s+1}) \qquad (\text{from Q1 } \boldsymbol{\Delta}_+^s = \mathcal{O}(h^s)) \\
&= \frac{1}{h}\left(\beta\boldsymbol{\Delta}_- + \sum_{j=1}^{s}\alpha_j\boldsymbol{\Delta}_+^j\right) + \mathcal{O}(h^{s+1})
\end{aligned}$$

where last equality holds if and only if $\alpha_{s+1} = 0$, from previous, we have

$$\alpha_{s+1} = (-1)^s(\frac{1}{s+1} - \beta) \qquad \Rightarrow \qquad \beta = \frac{1}{s+1}$$

□

Problem 4

Implementation of 5-point centered difference approximation to the poisson equation $\nabla^2 u = f$ on $\Omega = (0,1) \times (0,1)$ with Dirichlet boundary condition is in appendix. The maximum error is given as follows

```
>> q4
n       error
9       0.0005536347
19      0.0001401999
39      0.0000353078
79      0.0000088343
```

The rate of convergence for five point method is Δx^2. The result agrees with the theory, i.e. approximately, as n doubles, $\Delta x = \Delta y$ halfs, and error decreases by a factor of 4.

Problem 5

Implementation of 5-point centered difference approximation to possion equation $\nabla^2 u = f$ on $\Omega = (0,1)^2$ with Dirichlet boundary conditions on three sides and a Neumann boundary condition on the fourth side is in appendix. For both approximation method to the Neumann boundary condition, we simply change how we construct A and b matrix while leaving the rest the same.

1. **(first order)** Approximate $u_x(1,y)$ by $\frac{1}{\Delta x}\Delta_{-,x}$, we have the following boundary condition

$$\frac{u_{n+1,j} - u_{n,j}}{\Delta x} = -\frac{1}{4} \qquad \Rightarrow \qquad u_{n+1,j} = -\frac{1}{4}\Delta x + u_{n,j} \tag{1}$$

The discretization of $\frac{\partial^2}{\partial x^2}$ as $\Delta_{0,x}$ at the Neumann boundary, i.e. $(n\Delta x, j\Delta y)$ for $j = 1, 2, \cdots, n$, is

$$\begin{aligned}
\frac{u_{n+1,j} - 2u_{n,j} + u_{n-1,j}}{\Delta x^2} &= \frac{\left(-\frac{1}{4}\Delta x + u_{n,j}\right) - 2u_{n,j} + u_{n-1,j}}{\Delta x^2} \\
&= -\frac{1}{4\Delta x} + \frac{-u_{n,j} + u_{n-1,j}}{\Delta x^2}
\end{aligned}$$

we change the corresponding entries in A and b to reflect this.

$$A_{nj,nj} = -\frac{1}{\Delta x^2} - \frac{2}{\Delta y^2} \qquad b_{nj} \mathrel{+}= \frac{1}{4\Delta x}$$

2. **(second order)** Approximate $u_x(1, y)$ by $\frac{1}{\Delta x}(\varDelta_{-,x} + \frac{1}{2}\varDelta^2_{-,x})$, we have the following boundary condition at $j = 1, 2, \cdots, n$

$$\frac{\frac{3}{2}u_{n+1,j} - 2u_{n,j} + \frac{1}{2}u_{n-1,j}}{\Delta x} = -\frac{1}{4} \qquad \Rightarrow \qquad u_{n+1,j} = -\frac{1}{6}\Delta x + \frac{4}{3}u_{n,j} - \frac{1}{3}u_{n-1,j} \tag{2}$$

The discretization of $\frac{\partial^2}{\partial x^2}$ as $\varDelta_{0,x}$ at the Neumann boundary is

$$\begin{aligned}\frac{u_{n+1,j} - 2u_{n,j} + u_{n-1,j}}{\Delta x^2} &= \frac{\left(-\frac{1}{6}\Delta x + \frac{4}{3}u_{n,j} - \frac{1}{3}u_{n-1,j}\right) - 2u_{n,j} + u_{n-1,j}}{\Delta x^2} \\ &= -\frac{1}{6\Delta x} + \frac{-\frac{2}{3}u_{n,j} + \frac{2}{3}u_{n-1,j}}{\Delta x^2}\end{aligned}$$

we change the corresponding entries in A and b to reflect this

$$A_{nj,nj} = -\frac{2}{3\Delta x^2} - \frac{2}{\Delta y^2} \qquad A_{(n-1)j,nj} = \frac{2}{3\Delta x^2} \qquad b_{nj} \mathrel{+}= \frac{1}{6\Delta x}$$

Note for $i = n+1$, we obtain $\tilde{u}_{n+1,j}$ by evaluating $u(1, y) = \frac{1}{2} + \frac{1}{y+1}$ and we obtain $u_{n+1,j}$ by computing **(1)** for 1st order approximation method and **(2)** for 2nd order approximation method. The maximum error is given as follows

```
>> q5
n       order   error
9       1       0.0061413439
9       2       0.0015515841
19      1       0.0026819084
19      2       0.0003745321
39      1       0.0012470421
39      2       0.0000918166
79      1       0.0006004602
79      2       0.0000227175
```

Approximately, as n doubles, $\Delta x = \Delta y$ halves, error for first order method decreases by a factor of 2 while error for second order method decreases by a factor of 4. The error for first order method appears to be first-order accurate and that for second order method appears to be second-order accurate. Note since we are taking into account the $i = n+1$, the dominant factor in determining rate of convergence is at the Neumann boundary by observation. The rate of convergence of first/second order approximation at boundary agrees with observation.

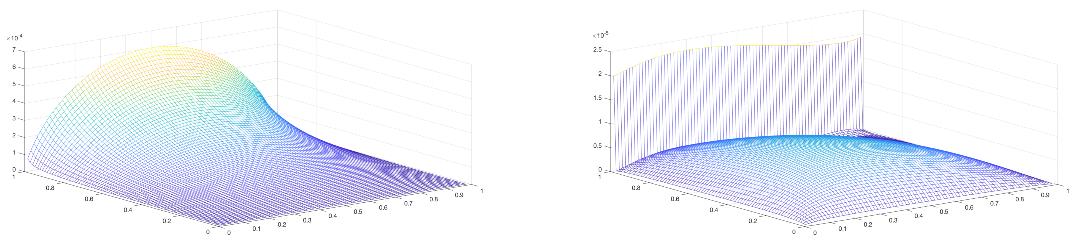

Figure 1: error graph for (left) first-order and (right) second-order method (n=79)

Problem 6

Implementation of 5-point centered difference approximation to possion equation $\nabla^2 u = f$ on $\Omega = \{(x,y) \mid x^2 + y^2 < 1\}$ with Dirichlet boundary conditions with specified f and u is in appendix. The maximum error is given as follows

```
>> q6
n       order    error
9       1        0.0093605463
9       2        0.0099505274
19      1        0.0024054093
19      2        0.0024963919
39      1        0.0006119863
39      2        0.0006247526
79      1        0.0001546807
79      2        0.0001562352
```

Approximately, as n doubles, $\Delta x = \Delta y$ halves, and error for both first order and second order method decreases by a factor of 4. The error for both first and second order method at boundary appears to be second-order accurate. This is counterintuitive. However, after plotting the error graph, it seems that error value is dominated by internal grid point estimates. Even though error is decreasing linearly at boundary for first order method, the error at boundary is masked by large error incurred at internal grid points.

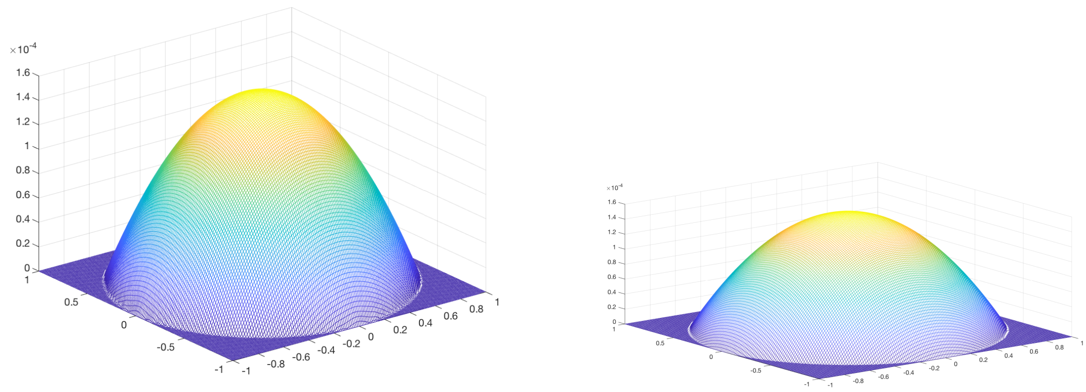

Figure 2: error graph for (left) first-order and (right) second-order method (n=79)

Problem 7

Upper and Lower Bounds for Inverse Elements of Finite and Infinite Tridiagonal Matrices **(https://core.ac.uk/download/pdf/82217635.pdf)** seems to solve this particular problem

Problem 8

BLOCK DIAGONAL DOMINANCE OF MATRICES REVISITED: BOUNDS FOR THE NORMS OF INVERSES AND EIGENVALUE INCLUSION SETS **(https://arxiv.org/pdf/1712.05662.pdf)** seems to solve this particular problem

Problem 9

(https://mathoverflow.net/questions/72832/overlapping-gershgorin-disks) seems to solve the problem

Appendix

problem 4 code

```
global m n;

fprintf('n\terror\n');
for i = [9, 19, 39, 79]
    m = i; n = i;
    solution = poisson_solver();
    exact_solution = maketildeu();
    e = max(arrayfun(@(x) abs(x), ...
        v2m(solution)-exact_solution), [], 'all');
    fprintf('%d\t%.10f\n', n, e);
end

% plot error function at grid points
function plot_err(expected, actual)
    global n;
    Dx = 1/(n+1);
    [X,Y] = meshgrid(0+Dx:Dx:1-Dx);
    mesh(X,Y,arrayfun(@(x) abs(x), actual-expected));
end


% construct and solve poisson equation over (0,1)^2
%       under dirichlet boundary condition
function sol = poisson_solver()
    A   = makeA();
    rhs = makerhs();
    sol = A \ rhs;
end


% gives exact solution to `A\tilde{u} = f+b`
%       tildeu    m x n
function tildeu = maketildeu()
    global m n;
    Dx = 1/(m+1);
    Dy = 1/(n+1);
    tildeu = zeros(m, n);
    for j = 1:n
        for i = 1:m
            tildeu(i, j) = tu(Dx*i, Dy*j);
        end
    end
end


% make right hand side to `Au = f+b`
%       rhs      m*n x 1
function rhs = makerhs()
    global m n;
    Dx = 1/(m+1);
```

```
    Dy = 1/(n+1);

    rhs = zeros(m*n, 1);
    for j = 1:n
        for i = 1:m
            % index to rhs
            p = ij2p(i, j);
            % x,y value
            x = Dx*i;
            y = Dy*j;

            % `f` evaluated at (x,y)
            fxy = f(x, y);

            % determine boundary value
            bnd = ...
                -u(Dx*(i-1), y)/Dx^2 ...
                -u(Dx*(i+1), y)/Dx^2 ...
                -u(x, Dy*(j-1))/Dy^2 ...
                -u(x, Dy*(j+1))/Dy^2;

            % populate rhs of `Au = f + b`
            rhs(p,1) = fxy + bnd;
        end
    end
end

% subroutines for conversion between
%       - grid indexed matrix     m   x n
%       - flattened vectors       m*n x 1
function mat = v2m(v)
    global m n;
    mat = zeros(m, n);
    for p = 1:(m*n)
        [i, j] = p2ij(p);
        mat(i, j) = v(p,1);
    end
end
function v = m2v(mat)
    global m n;
    v = zeros(m*n, 1);
    for j = 1:n
        for i = 1:m
            p = ij2p(i,j);
            v(p,1) = mat(i, j);
        end
    end
end

% subroutines for conversion between
%       - grid indices (i, j)
%       - corresponding index in `u,f,b` `p`
function [i, j] = p2ij(p)
    global m;
```

```
    j = floor((p-1) / m) + 1;
    i = mod(p-1, m) + 1;
end
function p = ij2p(i,j)
    global m;
    p = i + (j-1)*m;
end

% construct matrix 'A' on (0,1)^2
%       A     m*n x m*n
function A = makeA()
    global m n;
    Dx = 1/(m+1);
    Dy = 1/(n+1);
    Tsupdiag = repmat(1/Dx^2, m, 1);
    Tdiag = repmat(-2./Dx^2 -2/Dy^2, m, 1);
    Tsubdiag = repmat(1/Dx^2, m, 1);
    Tfar = repmat(1/Dy^2, m, 1);
    % When m == n or m > n, spdiags takes
    %       elements of the super-diagonal in A from the lower part of the correspo
    %       elements of the sub-diagonal   in A from the upper part of the correspo
    Tsupdiag(1,1) = 0;
    Tsubdiag(m,1) = 0;
    Tdiag = repmat(Tdiag, n, 1);
    Tsupdiag = repmat(Tsupdiag, n, 1);
    Tsubdiag = repmat(Tsubdiag, n, 1);
    Tfar = repmat(Tfar, n, 1);
    A = spdiags([Tfar Tsubdiag Tdiag Tsupdiag Tfar], [-m -1 0 1 m], m*n,m*n);
end


% evaluate 'u' at dirichlet boundary
%       outputs '0' for (x,y) not on boundary
function uxy = u(x, y)
    if x == 0
        uxy = 1 + 1/(1+y);
    elseif x == 1
        uxy = 1/2 + 1/(1+y);
    elseif y == 0
        uxy = 1 + 1/(1+x);
    elseif y == 1
        uxy = 1/2 + 1/(1+x);
    else
        uxy = 0;
    end
end

% evaluate function 'f' at grid location '(x,y)'
function fxy = f(x, y)
    fxy = 2/(1+x)^3 + 2/(1+y)^3;
end

% ground truth value for 'u'
function uxy = tu(x, y)
```

```
    uxy = 1/(1+x) + 1/(1+y);
end
```

problem 5 code

```
global m n;
global od;  % order of approximation to boundary derivative

fprintf('n\torder\terror\n');
for i = [9 19 39 79]
    m = i; n = i;
    Dx = 1/(m+1);
    for approx_order = [1 2]
        od = approx_order;
        solution = poisson_solver();
        solution = v2m(solution);
        exact_solution = maketildeu();
        for j = 1:n
            solution(m+1,j) = -0.25*Dx + solution(m, j);
        end
        e = max(arrayfun(@(x) abs(x), solution-exact_solution), [], 'all');
        fprintf('%d\t%d\t%.10f\n', n, od, e);
        plot_err(exact_solution, solution);
        saveas(gcf,strcat('q5_od_',num2str(od),'.png'));
    end
end


% plot error function at grid points
function plot_err(expected, actual)
    global m n;
    Dx = 1/(m+1);
    Dy = 1/(n+1);
    [X,Y] = meshgrid(0+Dx:Dx:1-Dx, 0+Dy:Dy:1);
    mesh(X,Y,arrayfun(@(x) abs(x), actual-expected));
end

% construct and solve poisson equation over (0,1)^2
%       under dirichlet boundary condition on 3 sides
%       and Neumann boundary condition on 1 side
function sol = poisson_solver()
    A   = makeA();
    rhs = makerhs();
    sol = A \ rhs;
end


% gives exact solution to `A\tilde{u} = f+b`
%       tildeu   m x n
function tildeu = maketildeu()
    global m n;
    Dx = 1/(m+1);
    Dy = 1/(n+1);
    tildeu = zeros(m, n);
    for j = 1:n
        for i = 1:(m+1)
            tildeu(i, j) = tu(Dx*i, Dy*j);
```

```
        end
    end
end


% make right hand side to `Au = f+b`
%          rhs       m*n x 1
function rhs = makerhs()
    global m n;
    Dx = 1/(m+1);
    Dy = 1/(n+1);

    rhs = zeros(m*n, 1);
    for j = 1:n
        for i = 1:m
            % index to rhs
            p = ij2p(i, j);
            % x,y value
            x = Dx*i;
            y = Dy*j;

            % `f` evaluated at (x,y)
            fxy = f(x, y);

            % determine boundary value
            bnd = ...
                 boundary_value(Dx*(i-1), y) ...
                +boundary_value(Dx*(i+1), y) ...
                +boundary_value(x, Dy*(j-1)) ...
                +boundary_value(x, Dy*(j+1));

            % populate rhs of `Au = f + b`
            rhs(p,1) = fxy + bnd;
        end
    end
end


% construct matrix `A` on (0,1)^2
%          A      m*n x m*n
function A = makeA()
    global m n od;
    Dx = 1/(m+1);
    Dy = 1/(n+1);
    Tsupdiag = repmat(1/Dx^2, m, 1);
    Tdiag = repmat(-2./Dx^2 -2/Dy^2, m, 1);
    Tsubdiag = repmat(1/Dx^2, m, 1);
    Tfar = repmat(1/Dy^2, m, 1);
    Tsupdiag(1,1) = 0;
    Tsubdiag(m,1) = 0;
    %%%%%%%%%%%%%%%%%%%%%%%%%%%%%
    % Neumann boundary changes
    if od == 1
        Tdiag(m,1) = -1/Dx^2 -2/Dy^2;
```

```
    elseif od == 2
        Tdiag(m,1) = -2/(3*Dx^2) - 2/Dy^2;
        Tsubdiag(m-1,1) = 2/(3*Dx^2);
    else
        assert(false);
    end
    %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
    Tdiag = repmat(Tdiag, n, 1);
    Tsupdiag = repmat(Tsupdiag, n, 1);
    Tsubdiag = repmat(Tsubdiag, n, 1);
    Tfar = repmat(Tfar, n, 1);
    A = spdiags([Tfar Tsubdiag Tdiag Tsupdiag Tfar], [-m -1 0 1 m], m*n,m*n);
end


% subroutines for conversion between
%       - grid indexed matrix    m   x n
%       - flattened vectors      m*n x 1
function mat = v2m(v)
    global m n;
    mat = zeros(m, n);
    for p = 1:(m*n)
        [i, j] = p2ij(p);
        mat(i, j) = v(p,1);
    end
end
function v = m2v(mat)
    global m n;
    v = zeros(m*n, 1);
    for j = 1:n
        for i = 1:m
            p = ij2p(i,j);
            v(p,1) = mat(i, j);
        end
    end
end

% subroutines for conversion between
%       - grid indices (i, j)
%       - corresponding index in `u,f,b` `p`
function [i, j] = p2ij(p)
    global m;
    j = floor((p-1) / m) + 1;
    i = mod(p-1, m) + 1;
end
function p = ij2p(i,j)
    global m;
    p = i + (j-1)*m;
end

% evaluate `u` at dirichlet boundary on 3 sides and Neumann boundary on 1 side
%       outputs `0` for (x,y) not on boundary
function uxy = boundary_value(x, y)
    global m n od;
```

```
    Dx = 1/(m+1);
    Dy = 1/(n+1);

    if x == 0
        uxy = -(1 + 1/(1+y))/Dx^2;
    elseif x == 1
        %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
        % Neumann boundary changes
        if od == 1
            uxy = 1/(4*Dx);
        elseif od == 2
            uxy = 1/(6*Dx);
        else
            assert(false);
        end
        %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
    elseif y == 0
        uxy = -(1 + 1/(1+x))/Dy^2;
    elseif y == 1
        uxy = -(1/2 + 1/(1+x))/Dy^2;
    else
        uxy = 0;
    end
end

% evaluate function `f` at grid location `(x,y)`
function fxy = f(x, y)
    fxy = 2/(1+x)^3 + 2/(1+y)^3;
end

% ground truth value for `u`
function uxy = tu(x, y)
    uxy = 1/(1+x) + 1/(1+y);
end
```

problem 6 code

```
global m n od;

od = 1;
[ij2p, p2ij, n_point] = make_conversion_table();

fprintf('n\torder\terror\n');
for i = [9 19 39 79]
    m = i; n = i;
    Dx = 1/(n+1);
    for approx_order = [1 2]
        od = approx_order;
        [A, rhs] = make_Arhs();
        solution = A \ rhs;
        solution = v2m(solution);
        exact_solution = maketildeu();
        e = max(arrayfun(@(x) abs(x), ...
            solution-exact_solution), [], 'all');
        fprintf('%d\t%d\t%.10f\n', n, od, e);
%           plot_err(exact_solution-solution);
%           diff = exact_solution-solution;
%           [X,Y] = meshgrid(-1:Dx:-1+n*Dx);
%           mesh(X,Y,arrayfun(@(x) abs(x), diff(1:(n+1),1:(n+1))));
%           saveas(gcf,strcat('q6_corder_od_',num2str(od),'.png'));
    end
end


% plot error function at grid points
function plot_err(V)
    global n;
    Dx = 1/(n+1);
    [X,Y] = meshgrid(-1:Dx:1);
    mesh(X,Y,arrayfun(@(x) abs(x), V));
end




function [A rhs] = make_Arhs()
    global n od;

    Dx = 1/(n+1); Dy = Dx;
    [ij2p, p2ij, n_point] = make_conversion_table();

    A = sparse(n_point, n_point);
    rhs = zeros(n_point, 1);

    for p = 1:n_point
        i = p2ij(p,1);
        j = p2ij(p,2);
        x1 = -1 + Dx*(i-1);
        y1 = -1 + Dy*(j-1);
```

```
find_tau_y = @(x,y) (sqrt(1-x^2) - abs(y))/Dy;
find_tau_x = @(x,y) (sqrt(1-y^2) - abs(x))/Dx;

% x-direction
if ij2p(i+1,j)==0 || ij2p(i-1,j)==0
    % near-boundary point
    P = ij2p(i,j);
    Q = 0; V = 0;
    if ij2p(i+1,j)==0
        Q = ij2p(i-1,j);
        V = ij2p(i-2,j);
    else
        Q = ij2p(i+1,j);
        V = ij2p(i+2,j);
    end
    tau = find_tau_x(x1,y1);
    assert(tau < 1.001 && tau > -0.001);
    if od == 1
        A(p,Q) = 2/(tau+1) * 1/Dx^2;
        A(p,P) = A(p,P) -2/tau * 1/Dy^2;
        rhs(p) = rhs(p) - 2/(tau*(tau+1)) * 1/Dx^2;
    else
        A(p,V) = (tau-1)/(tau+2) * 1/Dx^2;
        A(p,Q) = 2*(2-tau)/(tau+1) * 1/Dx^2;
        A(p,P) = A(p,P) -(3-tau)/tau * 1/Dy^2;
        rhs(p) = rhs(p) - 6/(tau*(tau+1)*(tau+2)) * 1/Dx^2;
    end
else
    % interior point
    A(p,p          ) = A(p,p) -2/Dy^2;
    A(p,ij2p(i-1,j)) = 1/Dx^2;
    A(p,ij2p(i+1,j)) = 1/Dx^2;
end

% y-direction
if ij2p(i,j+1)==0 || ij2p(i,j-1)==0
    % near boundary point
    P = ij2p(i,j);
    Q = 0; V = 0;
    if ij2p(i,j+1)==0
        Q = ij2p(i,j-1);
        V = ij2p(i,j-2);
    else
        Q = ij2p(i,j+1);
        V = ij2p(i,j+2);
    end
    tau = find_tau_y(x1,y1);
    assert(tau < 1.001 && tau > -0.001);
    if od == 1
        A(p,Q) = 2/(tau+1) * 1/Dy^2;
        A(p,P) = A(p,P) -2/tau * 1/Dy^2;
        rhs(p) = rhs(p) - 2/(tau*(tau+1)) * 1/Dy^2;
    else
        A(p,V) = (tau-1)/(tau+2) * 1/Dy^2;
```

```
                A(p,Q) = 2*(2-tau)/(tau+1) * 1/Dy^2;
                A(p,P) = A(p,P) -(3-tau)/tau * 1/Dy^2;
                rhs(p) = rhs(p) - 6/(tau*(tau+1)*(tau+2)) * 1/Dy^2;
            end
        else
            % interior point
            A(p,p          ) = A(p,p) -2/Dy^2;
            A(p,ij2p(i,j-1)) = 1/Dy^2;
            A(p,ij2p(i,j+1)) = 1/Dy^2;
        end
        rhs(p) = rhs(p) + f(x1,y1);
    end
end


function tildeu = maketildeu()
    global n;
    Dx = 1/(n+1); Dy = Dx;
    tildeu = zeros(2*n+3, 2*n+3);
    [ij2p, p2ij, n_point] = make_conversion_table();
    for p = 1:n_point
        i = p2ij(p,1);
        j = p2ij(p,2);
        x = -1 + Dx*(i-1);
        y = -1 + Dy*(j-1);
        tildeu(i,j) = tu(x,y);
    end
end


function mat = v2m(v)
    global n;
    mat = zeros(2*n+3,2*n+3);
    [~,p2ij,n_point] = make_conversion_table();
    for p = 1:n_point
        i = p2ij(p,1);
        j = p2ij(p,2);
        mat(i,j) = v(p);
    end
end


% construct matrix `ij2p` and `p2ij` s.t.
%   ij2p(i,j)  is  p
%   p2ij(p,:)  is  [i j]
%       where `i,j = 1,2,...,n+2` are grid indices
%       and `p = 1,2,...,n_gridpoint_inside_Omega` are indices to `u,b,f`

function [ij2p, p2ij, n_point] = make_conversion_table()
    global n;
    Dx = 1/(n+1); Dy = Dx;
    ij2p = zeros(2*n+3, 2*n+3);
    nnz = 1;
    for j = 1:(2*n+3)
```

```
        y = -1 + Dy*(j-1);
        for i = 1:(2*n+3)
            x = -1 + Dx*(i-1);
            if x^2+y^2 < 1
                ij2p(i, j) = nnz;
                nnz = nnz+1;
            end
        end
    end
    n_point = nnz-1;
    p2ij = zeros(n_point, 2);
    nnz = 1;
    for j = 1:(2*n+3)
        y = -1 + Dy*(j-1);
        for i = 1:(2*n+3)
            x = -1 + Dx*(i-1);
            if x^2+y^2 < 1
                p2ij(nnz,:) = [i j];
                nnz = nnz + 1;
            end
        end
    end
    % sanity check
    for k = 1:(nnz-1)
        assert(ij2p(p2ij(k,1), p2ij(k,2)) == k);
    end
end


% evaluate function `f` at grid location `(x,y)`
function fxy = f(x, y)
    fxy = 16*(x^2+y^2);
end

% ground truth value for `u`
function uxy = tu(x, y)
    uxy = (x^2+y^2)^2;
end

function y = sqrd(x, y)
    y = x^2+y^2;
end
```