Dan Card
dcard@umich.edu
Aerospace 523: Computational Fluid Dynamics
Midterm
Due: October 16th, 2020

1 Undetermined Coefficients [25%]

States u_i are given at three nodes in a non-uniform, one-dimensional grid, as shown below. Using the method of undetermined coefficients, derive the most accurate formula for du/dx at node 2, and give the order of accuracy, with respect to h, of your formula.

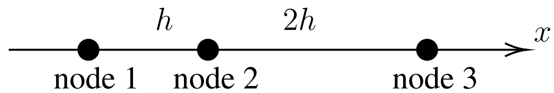

Figure 1: Undetermined coefficients non-uniform, one-dimensional grid.

First I will consider forward, backward, and central differences

Forward Difference:

$$\sum_{i=2}^{3} a_i u_i = a_2 u_2 + a_3 u_3$$

$$u_3 \approx u_2 + 2hu' + \frac{1}{2}(2h)^2 u'' + \frac{1}{6}(2h)^3 u'''$$

$$\frac{du}{dx}|_2 = a_2 u_2 + a_3 \left(u_2 + 2hu' + \frac{1}{2}(2h)^2 u'' + \frac{1}{6}(2h)^3 u''' \right)$$

This gives the following systems of equations,

$$a_2 + a_3 = 0, \quad 2ha_3 = 1 \text{ Solving gives, } a_3 = \frac{1}{2h}, \quad a_2 = -\frac{1}{2h}$$

$$\frac{du}{dx}|_2 \approx \frac{1}{2h}(u_3 - u_2)$$

$$a_3 2h^2 u'' \rightarrow \frac{1}{2h} 2h^2 u'' \rightarrow hu'', \quad \mathcal{O}(h) \rightarrow \text{First-Order}$$

Backward Difference:

$$\sum_{i=1}^{2} a_i u_i = a_1 u_1 + a_2 u_2$$

$$u_1 \approx u_2 + (-h)u' + \frac{1}{2}(-h)^2 u'' + \frac{1}{6}(-h)^3 u'''$$

$$\frac{du}{dx}|_2 = a_1(u_2 + (-h)u' + \frac{1}{2}(-h)^2 u'' + \frac{1}{6}(-h)^3 u''') + a_2 u_2$$

$$a_1 + a_2 = 0, \quad -ha_1 = 1, \text{ Solving gives, } a_1 = -\frac{1}{h}, \quad a_2 = \frac{1}{h}$$

$$\frac{du}{dx}|_2 \approx \frac{1}{h}(u_2 - u_1)$$

$$a_1 \frac{h^2}{2} u'' \rightarrow -\frac{1}{h}\frac{h^2}{2} u'' \rightarrow -\frac{h}{2} u'', \quad \mathcal{O}(h) \rightarrow \text{First-Order}$$

Central Difference:

$$\sum_{i=1}^{3} = a_1u_1 + a_2u_2 + a_3u_3$$

Using the expressions for u_1 and u_3 from forward and backward gives,

$$a_1(u_2 - hu' + \frac{1}{2}h^2u'' - \frac{1}{6}h^3u''') + \dots$$
$$a_2u_2 + \dots$$
$$a_3(u_2 + 2hu' + 2h^2u'' + \frac{4}{3}h^3u''')$$

This gives the following systems of equations,

$$a_1 + a_2 + a_3 = 0$$
$$-ha_1 + 2ha_3 = 1$$
$$\frac{1}{2}h^2a_1 + 2h^2a_3 = 0$$

Solving for $a_1,\ a_2,\ a_3$ gives,

$$a_1 = -\frac{2}{3h}, \quad a_2 = \frac{1}{2h}, \quad a_3 = \frac{1}{6h}$$

This gives that the approximation is,

$$\frac{du}{dx}|_2 \approx \frac{1}{6h}\left(u_3 + 3u_2 - 4u_2\right)$$

Finding the order of accuracy can be done by,

$$\left(a_1\frac{1}{2}h^2 + a_3 2h^2\right)u''$$
$$\left(-\frac{2}{3h}\frac{1}{2}h^2 + \frac{1}{6h}2h^2\right)u''$$
$$\left(-\frac{1}{3}h + \frac{1}{3}h\right)u'' = 0u''$$

Need to go higher,

$$\left(a_1\left(-\frac{1}{6}\right)h^3 + a_3\left(\frac{4}{3}\right)h^3\right)u'''$$
$$\left(\frac{2}{3h}\frac{1}{6}h^3 + \frac{1}{6h}\frac{4}{3}h^3\right)$$
$$\rightarrow \left(\frac{h^2}{9} + \frac{2}{9}h^2\right)u''' \rightarrow \frac{h^2}{2}u'''$$

Thus, a central order difference is more accurate $\mathcal{O}(\mathbf{h^2})$, meaning that it is second-order accurate and is more accurate than forward or backward differences (with them being first-order). The expression for du/dx at node 2 is given below.

$$\boxed{\frac{du}{dx}|_2 \approx \frac{1}{6h}\left(u_3 + 3u_2 - 4u_2\right)}$$

2 Gram Schmidt [25%]

Using the function inner product $(f,g)=\int_0^1 fg\ dx$, apply the Gram-Schmidt algorithm to orthonormalize the following two one-dimensional functions.

$$f_1 = x^4, \quad f_2 = 2x.$$

Then, project the function $g = x^2$ onto the space spanned by these two functions.

First is to orthogonalize the basis function,

$$\begin{aligned}
u_1 &= f_1 = \boxed{x^4} \\
u_2 &= f_2 - \frac{(u_1,f_2)}{(u_1,u_1)}u_1 \\
&= 2x - \frac{\int_0^1 x^4 \cdot 2x\ dx}{\int_0^1 x^4 \cdot x^4\ dx}x^4 = 2x - \frac{\int_0^1 2x^5\ dx}{\int_0^1 x^8\ dx}x^4 \\
&= 2x - \frac{1/3}{1/9}x^4 = \boxed{2x - 3x^4}
\end{aligned}$$

Then next is to normalize the orthongal functions,

$$\begin{aligned}
u_{1,norm} &= \frac{u_1}{\sqrt{(u_1,u_1)}} = \frac{x^4}{\sqrt{\int_0^1 x^8\ dx}} = \frac{x^4}{1/3} \\
u_{2,norm} &= \frac{u_2}{\sqrt{(u_2,u_2)}} = \frac{(2x-3x^4)}{\int_0^1 (2x-3x^4)^2\ dx} = \frac{2x-3x^4}{1/\sqrt{3}}
\end{aligned}$$

$$\boxed{u_{1,norm} = 3\,x^4, \quad u_{2,norm} = \sqrt{3}\,\left(2\,x - 3\,x^4\right)}$$

The projection of the vector can be expressed as

$$\mathrm{proj}_{u_i}(g) = \frac{(u_i,g)}{(u_i,u_i)}u_i$$

Performing the projection of the orthonormal vector gives,

$$\begin{aligned}
\mathrm{proj}_{u_1}(g) &= \frac{(u_1,g)}{(u_1,u_1)}u_1 = \frac{\int_0^1 x^4 \cdot x^2\ dx}{\int_0^1 x^4 \cdot x^4\ dx}x^4 = \frac{\int_0^1 x^6\ dx}{\int_0^1 x^8\ dx}x^4 = \boxed{\frac{9\,x^4}{7}} \\
\mathrm{proj}_{u_2}(g) &= \frac{(u_2,g)}{(u_2,u_2)}u_2 = \frac{\int_0^1 (2x-3x^4)\cdot x^2\ dx}{\int_0^1 (2x-3x^4)\cdot(2x-3x^4)\ dx}\left(2x-3x^4\right)
\end{aligned}$$

Computing the integral on matlab gives,

$$= \frac{\int_0^1 x^2\,(2\,x-3\,x^4)\ dx}{\int_0^1 (2\,x - 3\,x^4)^2\ dx}\left(2x-3x^4\right) = \boxed{\frac{3\,x}{7} - \frac{9\,x^4}{14}}$$

Again, expressing in vector form the projection of g onto the orthonormal vector function $\vec{u}$ gives,

$$\boxed{\mathrm{proj}_{u_1}(g) = \frac{9\,x^4}{7}, \quad \mathrm{proj}_{u_2}(g) = \frac{3\,x}{7} - \frac{9\,x^4}{14}}$$

Checking that these are orthogonal taking the inner product of $(\mathbf{u_1},\ \mathbf{u_2})$ gives that the inner produce is zero. Re-affirming that these are indeed orthogonal to each other. My matlab code can be found attached at the end of the assignment.

3 Gauss Seidel [25%]

Consider the successively over-relaxed, left-to-right Gauss-Seidel iterative smoother applied to a standard, second-order, central-difference discretization of the 1D Poisson equation ($-u_{xx} = f$) on $x \in [0, 1]$ with homogeneous Dirichlet boundary conditions. The grid is uniform and contains N intervals.

a. Write an expression for the iteration matrix, $\underline{\underline{S}}$, for this smoother, using the decomposition $\underline{\underline{A}} = \underline{\underline{L}} + \underline{\underline{D}} + \underline{\underline{U}}$, where $\underline{\underline{A}}\underline{u} = \underline{f}$ is the discretized system, as in the notes. Note that ω is a <u>successive</u> over-relaxation factor, applied immediately on each node update.

Since I will be implementing a second-order central-difference discretization of 1D Poisson's equation with homogeneous Dirichlet boundary conditions. The expression $\underline{\underline{A}}\underline{u} = \underline{f}$

$$\underbrace{\begin{bmatrix} 1 & 0 & 0 & 0 & \cdots & 0 & 0 \\ -1 & 2 & 1 & 0 & \cdots & 0 & 0 \\ 0 & -1 & 2 & -1 & \cdots & 0 & 0 \\ 0 & 0 & \ddots & \ddots & \ddots & 0 & 0 \\ \vdots & \vdots & & \ddots & \ddots & \ddots & \\ 0 & 0 & \cdots & 0 & -1 & 2 & -1 \\ 0 & 0 & \cdots & 0 & 0 & 0 & 1 \end{bmatrix}}_{\underline{\underline{A}}} \underbrace{\begin{bmatrix} u_1 \\ u_2 \\ u_3 \\ \vdots \\ \vdots \\ u_n \\ u_{n+1} \end{bmatrix}}_{\underline{u}} = \underbrace{\begin{bmatrix} 0 \\ h^2 f_2 \\ h^2 f_3 \\ \vdots \\ \vdots \\ h^2 f_n \\ 0 \end{bmatrix}}_{\underline{f}}$$

Re-writing $\underline{\underline{A}}$ in terms of $\underline{\underline{L}}$, $\underline{\underline{D}}$, $\underline{\underline{U}}$ gives,

$$\underline{\underline{L}} = \begin{bmatrix} 0 & 0 & 0 & \cdots & 0 \\ -1 & 0 & 0 & \cdots & 0 \\ 0 & \ddots & 0 & 0 & 0 \\ 0 & \cdots & -1 & 0 & 0 \\ 0 & \cdots & 0 & 0 & 0 \end{bmatrix}, \quad \underline{\underline{D}} = \begin{bmatrix} 1 & 0 & 0 & \cdots & 0 \\ 0 & 2 & 0 & \cdots & 0 \\ 0 & 0 & \ddots & \cdots & 0 \\ 0 & \cdots & 0 & 2 & 0 \\ 0 & \cdots & 0 & 0 & 1 \end{bmatrix}, \quad \underline{\underline{U}} = \begin{bmatrix} 0 & -1 & 0 & \cdots & 0 \\ 0 & 0 & \ddots & \cdots & 0 \\ 0 & 0 & 0 & -1 & 0 \\ 0 & \cdots & 0 & 0 & -1 \\ 0 & \cdots & 0 & 0 & 0 \end{bmatrix}$$

The smoothing iteration matrix is then given by,

$$\underline{\underline{S}} = -\left(\underline{\underline{D}} + \underline{\underline{L}}\right)^{-1} \underline{\underline{U}}$$

Doing so and creating a general expression for the matrix I get that $\underline{\underline{S}}$ can be expressed to be,

$$\boxed{\underline{\underline{S}} = \begin{bmatrix} 0 & 0 & 0 & 0 & \cdots & \cdots & 0 & 0 \\ 0 & 0 & \frac{1}{2} & 0 & \cdots & \cdots & 0 & 0 \\ 0 & 0 & \frac{1}{4} & \frac{1}{2} & \ddots & \cdots & 0 & 0 \\ 0 & 0 & \frac{1}{8} & \frac{1}{4} & \ddots & \ddots & 0 & 0 \\ 0 & 0 & \vdots & \ddots & \ddots & \ddots & \ddots & 0 \\ 0 & 0 & \frac{1}{2^{n-1}} & \cdots & \ddots & \ddots & \ddots & 0 \\ 0 & 0 & \frac{1}{2^n} & \frac{1}{2^{n-1}} & \cdots & \frac{1}{8} & \frac{1}{4} & \frac{1}{2} \\ 0 & 0 & \cdots & \cdots & 0 & 0 & 0 & 0 \end{bmatrix}}$$

Looking to the matrix above, it takes the form that the first two columns are completely zero and that the first/last rows are fully zero as well. Then along the lower matrix it follows a pattern of decreasing by factors of two from the value of the index above it. Meaning the values decrease in $\frac{1}{2}, \frac{1}{2^2}, \frac{1}{2^3}, \frac{1}{2^4}, \ldots, \frac{1}{2^{n-1}}, \frac{1}{2^n}$ along the rows of the matrix along the diagonal.

b. In the complex number plane, plot the eigenvalues of the iteration matrix, $\underline{\underline{S}}$, for this smoother,using $N = 32$ and an over-relaxation factor of $\omega = 1.5$.

Plotting the eigenvalues of the iteration matrix by the formula,

$$\lambda(\underline{\underline{S}}_{\omega}) = \omega\lambda(\underline{\underline{S}}) + (1 - \omega)$$

Gives the plot shown below in Figure 2,

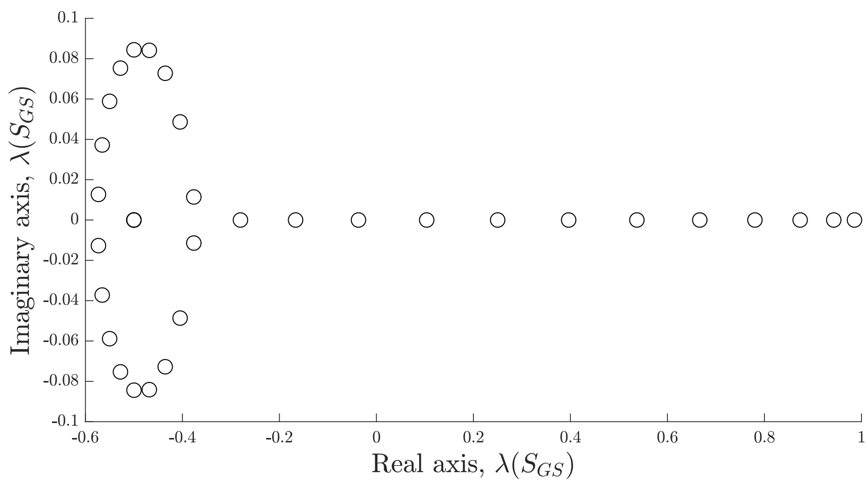

Figure 2: Eigenvalues in the complex number plane.

c. Make a plot of the magnitude of the largest magnitude eigenvalue of $\underline{\underline{S}}$ versus ω, and identify the optimal over-relaxation factor. Use $N = 32$.

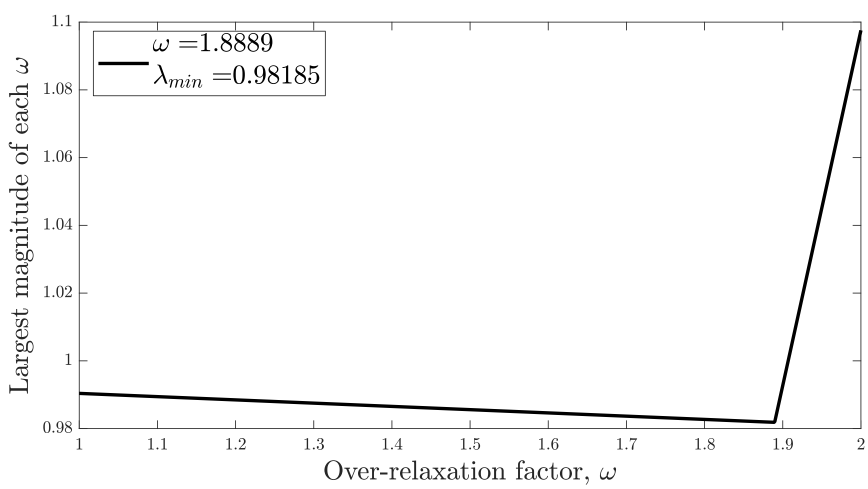

Figure 3: Determining the optimal over-relaxation factor.

Looking above to Figure 3, the optimal over-relaxation value occurs when the magnitude of the eigenvalues reach a minimum. This is because we need $\lambda(\underline{\mathbf{S}}) <$ 1 for convergence and further from 1 for fast convergence. This minimum value can be found on the Figure to be $\lambda_{\mathbf{min}} = 0.98185$, at this minimum amplification factor the over-relaxation factor is found to be an optimal of $\omega = 1.8889$. Any value of ω greater than this value will risk deteriorating the convergence rates. Matlab code can be found attached at the end of my assignment.

4 Mesh Connectivity [25%]

Section 1.7.5 of the notes introduces matrices $\underline{\underline{E}}$ and $\underline{\underline{N}}$ for describing connections between elements and nodes of a mesh. Assume that both matrices are made unique by using a counter-clockwise ordering of nodes/elements and by beginning each row with the smallest index. Element/node numbering starts at 1 and no numbers are skipped. Assume triangular elements.

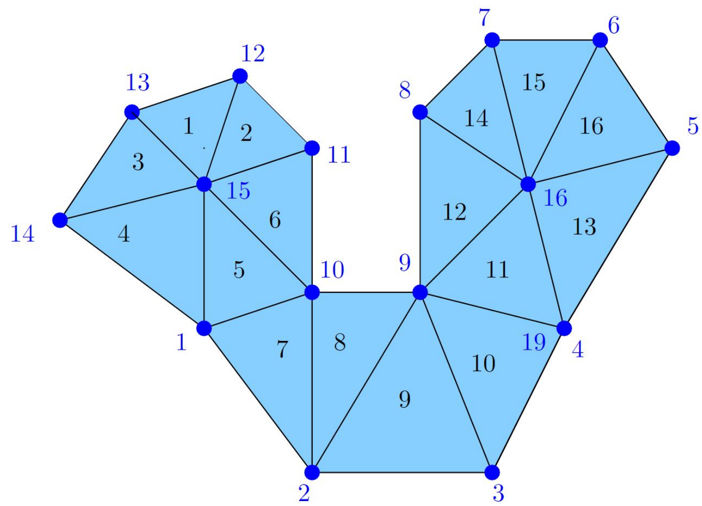

Figure 4: Triangular mesh grid.

a. For the mesh shown above, write down the matrices $\underline{\underline{E}}$ and $\underline{\underline{N}}$.

$$\underline{\underline{E}} = \begin{bmatrix} 12 & 13 & 15 \\ 11 & 12 & 15 \\ 13 & 14 & 15 \\ 1 & 15 & 14 \\ 1 & 10 & 15 \\ 10 & 11 & 15 \\ 1 & 2 & 10 \\ 2 & 9 & 10 \\ 2 & 3 & 9 \\ 3 & 4 & 9 \\ 4 & 16 & 9 \\ 8 & 9 & 16 \\ 4 & 5 & 16 \\ 7 & 8 & 16 \\ 6 & 7 & 16 \\ 5 & 6 & 16 \end{bmatrix}, \qquad \underline{\underline{N}} = \begin{bmatrix} 4 & 5 & 7 & & & \\ 7 & 8 & 9 & & & \\ 9 & 10 & & & & \\ 10 & 11 & 13 & & & \\ 13 & 16 & & & & \\ 15 & 16 & & & & \\ 14 & 15 & & & & \\ 12 & 14 & & & & \\ 8 & 9 & 10 & 11 & 12 & \\ 5 & 6 & 7 & 8 & & \\ 2 & 6 & & & & \\ 1 & 2 & & & & \\ 1 & 3 & & & & \\ 3 & 4 & & & & \\ 1 & 2 & 3 & 4 & 5 & 6 \\ 11 & 12 & 13 & 14 & 15 & 16 \end{bmatrix}$$

Shown above are the matrices for $\underline{\underline{\mathbf{E}}}$ and $\underline{\underline{\mathbf{N}}}$ for the irregular mesh as indicated in Section 1.7.5 of the course notes.

b. Write a pseudo-code for <u>efficiently</u> determining $\underline{\underline{E}}$ given $\underline{\underline{N}}$. For this part, the rows of the $\underline{\underline{E}}$ matrix need not satisfy the ordering requirement.

<u>Pseudo-Code</u>

```
Create E = [ ]
for i = 1:length(N)   Loop through all the nodes in N
  for j = 1:max(size(N(i,:)))   Loop through all the elements for a given node
    E(N(i, j), end) = i   Place the node value at the end of the the i^th column in E
  end
end
```

This is the code in theory that should be implemented to be efficiently implemented meaning that it scales $\mathcal{O}(\texttt{size}(\underline{\underline{N}}))$. However, placing these values into the $\underline{\underline{\mathbf{E}}}$ matrix is unique to each given coding programing language (i.e. C++ would use `.pushback()`) so in application, with Matlab at least, I would pre-allocate $\underline{\underline{\mathbf{E}}}$ to be a matrix of `nan` and have several `if, elseif` statements to determine the ending index for a given iteration `if` the last index is `nan` and replace with the current node value from the iteration through $\underline{\underline{\mathbf{N}}}$.

c. How many edges (total, interior and boundary) are in the above mesh? Write a pseudo-code for <u>efficiently</u> determining the number of edges in a mesh given $\underline{\underline{E}}$.

Total:	31
Interior:	17
Boundary:	14

<u>Pseudo-Code</u>

```
Create N to have empty indices of nan
counter = 0   Initialize counter to be zero
for j = 1:max(size(N))   Loop through all the nodes
  for i = 1:max(size(N(j, :)))   Loop through each element adjacent to a node
    if isnan(N(j, i))  = 1   For easy implementation check if value is nan, because
    if so it has been used before and should be skipped
      [row, col] = find(N == N(j,i))   Find the indices of the node
      N(row(end), col(end)) = nan   Replace with nan to indicate it's been used
      counter += 1   Increment the counter
    end if, end for(i), end for(j)
counter -= 1   Since the final pass is ineveitable without heavy modification, increment down after the full pass
```

In the pseudo-code shown above, it scales $\mathcal{O}(\mathrm{N})$ making it as efficient as possible in determining the number of edges. Actual code implementation can be found attached at the end of my assignment.

Matlab Code

Algorithm 1: Matlab code for Gram Schmidt orthonormalization.

```
%~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
clear all; clc; close all
%~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
syms x
f1 = x^4; f2 = 2*x;

u1 = f1;
u2 = f2 - int(u1*f2, x, 0, 1)/int(u1*u1, x, 0, 1) * u1;

int(u1*u2, x, 0, 1)
tolatex('u1norm', u1/sqrt(int(u1*u1, x, 0, 1)))
tolatex('u2norm', u2/sqrt(int(u2*u2, x, 0, 1)))

g = x^2;
proju1g = int(u1*g, x, 0, 1) / int(u1*u1, x, 0, 1)* u1;
proju2g = int(u2*g, x, 0, 1) / int(u2*u2, x, 0, 1)* u2;
```

Algorithm 2: Matlab implementation to determine over-relaxation factor for Gauss Seidel.

```
%~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
clear all; clc; close all
set(groot,'defaulttextinterpreter','latex');
set(groot, 'defaultAxesTickLabelInterpreter','latex');
set(groot, 'defaultLegendInterpreter','latex');
%~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
% Part b
N = 32; omega = 1.5;
D = eye(N+1); L = zeros(N+1); U = L;

for i = 1:N-1
   L(i+1, i) = -1; % Fill the left-matrix
end
for i = 2:N
    U(i, i+1) = -1;% Fill the right-matrix
    D(i,i) = 2; % Fill the rest of the diagonals
end

Sgs = -inv(D + L)*U; % Solve for the iterative matrix
lambdas = omega.*eig(Sgs) + (1-omega);

figure()
scatter(real(lambdas), imag(lambdas), 75, 'ko')
xlabel('Real axis, $\lambda(S_{GS})$', 'fontsize', 16)
ylabel('Imaginary axis, $\lambda(S_{GS})$', 'fontsize', 16)
set(gcf, 'Color', 'w', 'Position', [200 200 800 400]);
export_fig('q3_eigens.eps')
%~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
% Part c
num = 1000;
omegas = linspace(1, 2, num);
eigval = eig(Sgs);

data = zeros(num, 1);
for i = 1:num
    vals = omegas(i).*eigval + (1-omegas(i));
    norms = sqrt(real(vals).^2 + imag(vals).^2);
    data(i) = max(norms);
end

idx = find(data == min(data));

figure()
plot(omegas, data, 'k', 'linewidth', 2)
xlabel('Over-relaxation factor, $\omega$', 'fontsize', 16)
ylabel('Largest magnitude of each $\omega$', 'fontsize', 16)
legend(['$\omega = $', num2str(omegas(idx)), newline, '$\lambda_{min} = $', num2str(min(data))],
    'location', 'best', 'fontsize', 16)
set(gcf, 'Color', 'w', 'Position', [200 200 800 400]);
export_fig('q3_omegas.eps')
```

Algorithm 3: Matlab implementation for determining number of edges on a mesh.

```
%~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
clear all; clc; close all
set(groot,'defaulttextinterpreter','latex');
set(groot, 'defaultAxesTickLabelInterpreter','latex');
set(groot, 'defaultLegendInterpreter','latex');
%~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
N = [4, 5, 7, nan, nan, nan; 7, 8, 9, nan, nan, nan;
     9, 10, nan, nan, nan, nan; 10,11,13, nan, nan, nan;
     13,16, nan, nan, nan, nan; 15,16, nan, nan, nan, nan;
     14,15, nan, nan, nan, nan; 12,14, nan, nan, nan, nan;
     8:12, nan; 5:8, nan, nan;
     2,6, nan, nan, nan, nan; 1,2, nan, nan, nan, nan;
     1,3, nan, nan, nan, nan; 3,4, nan, nan, nan, nan;
     1:6; 11:16];
E = [12, 13, 15; 11, 12, 15; 13, 14, 15; 1, 15, 14; 1, 10, 15;
    10, 11, 15; 1, 2 , 10; 2, 9 , 10; 2, 3 , 9; 3, 4 , 9;
    4, 16, 9; 8, 9 , 16; 4, 5 , 16; 7, 8 , 16; 6, 7 , 16; 5, 6 , 16];

counter = 0;
for j = 1:max(size(N))
    for i = 1:max(size(N(j, :)))
        if isnan(N(j, i)) ~= 1
            [row, col] = find(N == N(j, i));
            N(row(end), col(end)) = nan;
            counter = counter + 1;
        end
    end
end
counter = counter - 1;
```