Name: ________________________________

1. (15 points) In class, we discussed iterative schemes based on a specific **splitting** of an $n \times n$ matrix A to solve the linear system $A\mathbf{x} = \mathbf{b}$.

   (a) (5 points) If $A = M - N$, then show that the following schemes are equivalent:

$$
\begin{aligned}
M\mathbf{x}_{k+1} &= N\mathbf{x}_k + \mathbf{b}; \\
\mathbf{x}_{k+1} &= (\mathbb{I} - M^{-1}A)\mathbf{x}_k + M^{-1}\mathbf{b}, \quad \text{where } \mathbb{I} \text{ is the } n \times n \text{ identity matrix;} \\
\mathbf{x}_{k+1} &= \mathbf{x}_k + M^{-1}\mathbf{r}_k, \quad \text{where} \quad \mathbf{r}_k = \mathbf{b} - A\mathbf{x}_k.
\end{aligned}
$$

   (b) (10 points) A totally **equivalent splitting** of A that we discussed is of the form: $A = L + U + D$. Note that, L is **strictly** lower triangular, U **strictly** upper triangular and D diagonal. This way, the Jacobi method reads

$$\boxed{\mathbf{x}_{k+1} = R_J\mathbf{x}_k + D^{-1}\mathbf{b},}$$

   with R_J the Jacobi iteration matrix $R_J = -D^{-1}(L + U)$.

   If A is **strictly diagonally dominant**, show that the Jacobi iteration matrix satisfies

$$\|R_J\|_\infty < 1.$$

   Note that if this condition holds, then the Jacobi method converges for **any** initial vector $\mathbf{x}_0$.

   Hints:

   - An $n \times n$ matrix A is strictly diagonally dominant if $|a_{ii}| > \sum_{\substack{j=1 \\ j \neq i}}^{n} |a_{ij}|$ holds.
   - Note that if A is an $n \times n$ matrix, then $\|A\|_\infty = \max_{1 \leq i \leq n} \sum_{j=1}^{n} |a_{ij}|$.

Solution:

(a) Multiplying the first scheme by M^{-1} and using the fact that $N = M - A$, we obtain

$$\mathbf{x}_{k+1} = M^{-1}(M - A)\mathbf{x}_k + M^{-1}\mathbf{b} \Rightarrow \boxed{\mathbf{x}_{k+1} = \left(\mathbb{I} - M^{-1}A\right)\mathbf{x}_k + M^{-1}\mathbf{b}},$$

which is the second scheme. Expanding the terms in the right-hand-side of the second scheme and re-grouping them we obtain

$$\mathbf{x}_{k+1} = \mathbb{I}\mathbf{x}_k - M^{-1}A\mathbf{x}_k + M^{-1}\mathbf{b} \Rightarrow \boxed{\mathbf{x}_{k+1} = \mathbf{x}_k + M^{-1}\underbrace{(\mathbf{b} - A\mathbf{x}_k)}_{\mathbf{r}_k}},$$

which is the third scheme.

(b) Since A is strictly diagonally dominant, this means that $a_{ii} \neq 0$ and D^{-1} exists. This way, we explicitly write out R_J as follows

$$R_J = -\begin{bmatrix} (a_{1,1})^{-1} & & & & \\ & (a_{2,2})^{-1} & & & \\ & & \ddots & & \\ & & & (a_{n-1,n-1})^{-1} & \\ & & & & (a_{n,n})^{-1} \end{bmatrix} \cdot \begin{bmatrix} 0 & a_{1,2} & \cdots & a_{1,n-1} & a_{1,n} \\ a_{2,1} & 0 & a_{2,3} & \cdots & a_{2,n} \\ \vdots & \ddots & \ddots & \ddots & \vdots \\ a_{n-1,1} & a_{n-1,2} & \cdots & 0 & a_{n-1,n} \\ a_{n,1} & a_{n,2} & \cdots & a_{n,n-1} & 0 \end{bmatrix},$$

and multiplying the matrices we arrive at

$$R_J = -\begin{bmatrix} 0 & a_{1,2}/a_{1,1} & \cdots & a_{1,n-1}/a_{1,1} & a_{1,n}/a_{1,1} \\ a_{2,1}/a_{2,2} & 0 & a_{2,3}/a_{2,2} & \cdots & a_{2,n}/a_{2,2} \\ \vdots & \ddots & \ddots & \ddots & \vdots \\ a_{n-1,1}/a_{n-1,n-1} & a_{n-1,2}/a_{n-1,n-1} & \cdots & 0 & a_{n-1,n}/a_{n-1,n-1} \\ a_{n,1}/a_{n,n} & a_{n,2}/a_{n,n} & \cdots & a_{n,n-1}/a_{n,n} & 0 \end{bmatrix}$$

By definition, the ∞-norm is

$$\|R_J\|_\infty = \max_{1\le i\le n} \sum_{j=1}^{n} |\,(R_J)_{ij}\,| \Rightarrow \boxed{\|R_J\|_\infty = \max_{1\le i\le n} \sum_{\substack{j=1\\ j\neq i}}^{n} \left|\frac{a_{i,j}}{a_{i,i}}\right| < 1},$$

which holds, since A is **strictly diagonally dominant**. As a result, the Jacobi method will converge starting from **any** initial vector $\mathbf{x}_0$.

2. (a) (10 points) Find **by hand** the **eigenvalues**, **eigenvectors** and **spectral radius** of the following matrices:

$$\begin{bmatrix} 2 & -1 \\ -1 & 2 \end{bmatrix}, \quad \begin{bmatrix} 0 & -1 \\ 1 & 0 \end{bmatrix}.$$

You may use MATLAB's `eig` command to **verify** your answers. Furthermore, you can find the spectral radius of a matrix easily using MATLAB by typing `max(abs(eig))`.

(b) (15 points) Find **all** the values of a and b for which the matrix

$$A = \begin{bmatrix} a & 1 & 1+b \\ 1 & a & 1 \\ 1-b^2 & 1 & a \end{bmatrix}$$

is **symmetric positive definite**.

Solution:

(a) In the first part, we determine the eigenvalues via the characteristic equation: $\det(A - \lambda \mathbb{I}) = 0$. Then, upon finding the eigenvalues, we will determine the components of the eigenvectors, i.e., x_1 and x_2 of $\mathbf{x} = [x_1\, x_2]^T$. This way we have:

- The eigenvalues of the first matrix are $\lambda_1 = 1$ and $\lambda_2 = 3$. For λ_1 the eigenvector can be found as follows:

$$\begin{bmatrix} 2 & -1 \\ -1 & 2 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} = \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} \Rightarrow x_1 = x_2.$$

Thus, for $x_1 = 1$, the associated eigenvector is $\mathbf{x}^{(1)} = [1\, 1]^T$. On equally footing and for λ_2 at hand, we have

$$\begin{bmatrix} 2 & -1 \\ -1 & 2 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} = 3 \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} \Rightarrow -x_1 = x_2.$$

For $x_1 = 1$, we obtain $\mathbf{x}^{(2)} = [1\ -1]^T$.

- The eigenvalues of the second matrix are $\lambda_1 = i$ and $\lambda_2 = -i$. The associated eigenvector $\mathbf{x}^{(1)}$ for λ_1 can be obtained via

$$\begin{bmatrix} 0 & -1 \\ 1 & 0 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} = i \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} \Rightarrow -x_2 = ix_1.$$

For $x_1 = 1$ we obtain $\mathbf{x}^{(1)} = [1\ -i]^T$. Finally, and as per the second eigenvalue λ_2, we have similarly

$$\begin{bmatrix} 0 & -1 \\ 1 & 0 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} = -i \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} \Rightarrow x_1 = -ix_2,$$

and for $x_1 = 1$, the eigenvector is $\mathbf{x}^{(2)} = [1\, i]^T$.

(b) The matrix given **is** symmetric if $A = A^T$ holds. From the latter condition, we obtain

$$1 + b = 1 - b^2 \Rightarrow b^2 + b = 0 \Rightarrow \boxed{b = 0} \quad \text{or} \quad \boxed{b = -1}.$$

This way, and for the values of b obtained, we have to check whether A is positive definite, that is, whether $\mathbf{x}^T A\mathbf{x} > 0$ holds for any non-zero vector $\mathbf{x} = [x_1\, x_2\, x_3]^T$. Equivalently, we seek for the corresponding values of a such that A is positive definite.

- **Case with** $b = 0$
  The associated matrix is

$$A = \begin{bmatrix} a & 1 & 1 \\ 1 & a & 1 \\ 1 & 1 & a \end{bmatrix}$$

and direct calculation yields to

$$\begin{aligned}
\mathbf{x}^T A\mathbf{x} &= [x_1, x_2, x_3] \begin{bmatrix} a & 1 & 1 \\ 1 & a & 1 \\ 1 & 1 & a \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} \Rightarrow \\
\mathbf{x}^T A\mathbf{x} &= a(x_1^2 + x_2^2 + x_3^2) + 2x_1x_2 + 2x_1x_3 + 2x_2x_3 \overset{a=1}{\Longrightarrow} \\
\mathbf{x}^T A\mathbf{x} &= (x_1 + x_2 + x_3)^2 > 0.
\end{aligned}$$

However, for $a = 1$, the matrix becomes singular and the leading principal minors are all 0 [Recall that a a minor of an $n \times n$ matrix A of order k is principal if it is obtained by deleting $n - k$ rows and the $n - k$ columns with the same numbers. Then, the leading principal minor of A of order k denoted by D_k is the minor of order k obtained by deleting the last $n - k$ rows and columns!]

Therefore, for $b = 0$ and $\forall a > 1$ the matrix A **is** symmetric **and** positive definite assuming that at least one $x_i \neq 0$. We can further corroborate the result by computing the leading principal minors:

$$\begin{aligned}
D_1 &= a, \\
D_2 &= a^2 - 1, \\
D_3 &= a^3 - 3a + 2 = (a - 1)^2(a + 2),
\end{aligned}$$

so, it is clear that if $a > 1$, then $D_k > 0$ holds (with $k = 1, 2, 3$), which suggests that A is positive definite too.

Another way to show that the matrix is positive definite, is by imposing its eigenvalues to be strictly positive. A direct calculation of the eigenvalues of the above matrix reveals that

$$\lambda_{1,2} = a - 1, \quad \lambda_3 = 2 + a.$$

This way, $\lambda_i > 0$ if $a > 1$ and $a > -2$, or simply $\boxed{a > 1}$.

- **Case with** $b = -1$
  The corresponding matrix here is

$$A = \begin{bmatrix} a & 1 & 0 \\ 1 & a & 1 \\ 0 & 1 & a \end{bmatrix},$$

with its associated leading principal minors:

$$\begin{aligned} D_1 &= a, \\ D_2 &= a^2 - 1, \\ D_3 &= a\left(a^2 - 2\right), \end{aligned}$$

where $D_k > 0$ for $k = 1, 2, 3$ holds iff $a > \sqrt{2}$. Thus, the matrix A is symmetric and positive definite for $b = -1$ and $a > \sqrt{2}$.
Similarly, a direct computation of the eigenvalues gives

$$\lambda_1 = a, \quad \lambda_{2,3} = \pm\sqrt{2} + a.$$

The latter is true if $\boxed{a > \sqrt{2}}$.

3. (30 points) The linear system $A\mathbf{x} = \mathbf{b}$ with

$$A = \begin{bmatrix} 4 & 1 \\ 1 & 4 \end{bmatrix}, \quad \mathbf{b} = \begin{bmatrix} 5 \\ 5 \end{bmatrix}$$

has the unique solution $\mathbf{x} = [1\,1]^T$.

(a) (10 points) Determine **by hand** the $R_J = -D^{-1}(L + U)$ and $R_{GS} = -(L + D)^{-1}U$, that is, the Jacobi and Gauss-Seidel iteration matrices, respectively (of course you may use MATLAB to **verify** your answers).

(b) (5 points) Find the ∞-norm and spectral radius of R_J and R_{GS}.

(c) (15 points) Perform **5 iterations** of both Jacobi and Gauss-Seidel methods using $\mathbf{x}_0 = [0\,0]^T$. For each present your results in a table with the following format:

- column 1: k (iteration step)

- column 2: $x_1^{(k)}$ (1st component of the computed solution vector at step k)
- column 3: $x_2^{(k)}$ (2nd component of the computed solution vector at step k)
- column 4: $\|e^{(k)}\|_\infty$ (error norm at step k)
- column 5: $\|e^{(k)}\|_\infty / \|e^{(k-1)}\|_\infty$ (ratio of successive error norms at step k).

Which method is converging faster? Attach any of your codes and **justify your answer**.

Solution:

(a) The matrices appearing at the iteration matrices therein are

$$D = \begin{bmatrix} 4 & 0 \\ 0 & 4 \end{bmatrix}, \quad L = \begin{bmatrix} 0 & 0 \\ 1 & 0 \end{bmatrix}, \quad U = \begin{bmatrix} 0 & 1 \\ 0 & 0 \end{bmatrix},$$

and in this way, we obtain

$$L + U = \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix}, L + D = \begin{bmatrix} 4 & 0 \\ 1 & 4 \end{bmatrix}, (L + D)^{-1} = \frac{1}{4} \begin{bmatrix} 1 & 0 \\ -1/4 & 1 \end{bmatrix}, D^{-1} = \frac{1}{4}\mathbb{I}.$$

A direct calculation reveals that

$$R_J = -D^{-1}(L + U) = -\frac{1}{4} \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix},$$
$$R_{GS} = -(L + D)^{-1} U = -\frac{1}{4} \begin{bmatrix} 0 & 1 \\ 0 & -1/4 \end{bmatrix}.$$

(b) The ∞-norms of the respective iteration matrices as well as their spectral radii are

$$\|R_J\|_\infty = 1/4, \quad \rho(R_J) = 1/4,$$
$$\|R_{GS}\|_\infty = 1/4, \quad \rho(R_{GS}) = 1/16.$$

Note that $\rho(R_{GS}) < \rho(R_J) < 1$, so both methods **do** converge!

(c) The MATLAB output and the script producing it follow:

```
 Jacobi:

k     x_{1}           x_{2}          error_inf     error_ratio
------------------------------------------------------------
0  0.000000e+00  0.000000e+00  1.000000e+00  0.000000e+00
1  1.250000e+00  1.250000e+00  2.500000e-01  2.500000e-01
2  9.375000e-01  9.375000e-01  6.250000e-02  2.500000e-01
```

```
3  1.015625e+00  1.015625e+00  1.562500e-02  2.500000e-01
4  9.960938e-01  9.960938e-01  3.906250e-03  2.500000e-01
5  1.000977e+00  1.000977e+00  9.765625e-04  2.500000e-01


 Gauss Seidel:

k     x_{1}         x_{2}        error_inf    error_ratio
-----------------------------------------------------------
0  0.000000e+00  0.000000e+00  1.000000e+00  0.000000e+00
1  1.250000e+00  9.375000e-01  2.500000e-01  2.500000e-01
2  1.015625e+00  9.960938e-01  1.562500e-02  6.250000e-02
3  1.000977e+00  9.997559e-01  9.765625e-04  6.250000e-02
4  1.000061e+00  9.999847e-01  6.103516e-05  6.250000e-02
5  1.000004e+00  9.999990e-01  3.814697e-06  6.250000e-02
```

```matlab
clearvars; close all; clc; format short;

% System given:
A = [4,1;1,4]; b = [5;5];

% Construct the D, U and L matrices:
D = diag(diag(A));  % Diagonal matrix from A.
U = triu(A,1);      % Strictly upper triangular matrix.
L = tril(A,-1);     % Strictly lower triangular matrix.

% Construct the iteration matrices:
   RJ = - inv(D) * ( L + U ); % For Jacobi method.
  RGS = -inv(L+D) * U;        % For Gauss-Seidel method.
 invD = inv(D);               % Compute it once!
invDL = inv(D+L);             % Compute it once!

% Setup:
    x0 = [0;0];            % Initial vector.
    xe = [1;1];            % Exact solution.
  nmax = 5;                % Max iterations.
err0_1 = norm(xe-x0,inf);  % error_inf.
err0_2 = 0;                % error ratio.
fprintf('%s \n  \n  ',' Jacobi:                ');
fprintf('%s %s %s %s %s\n  ','k','x_{1}','x_{2}',...
                       'error_inf','error_ratio');
fprintf('%s \n  ' ,'-----------------------------...
                      --------------------------');
fprintf ('%d %e %e %e %e\n  ',0,x0(1), x0(2), err0_1, ...
    err0_2 );
```

```
for k = 1:nmax
       xk =  RJ * x0 + invD * b;     % Jacobi's method.
       x0 = xk;                      % Update solution.
     errt = norm(xe-x0,inf);
   err0_2 = errt / err0_1; err0_1 = errt;
   fprintf('%d %e %e %e %e\n  ',k,x0(1), x0(2), err0_1, ...
      err0_2 );
end

% Re-initialize:
    x0 = [0;0];              % Initial vector.
    xe = [1;1];              % Exact solution.
  nmax = 5;                  % Max iterations.
err0_1 = norm(xe-x0,inf);    % error_inf.
err0_2 = 0;                  % error ratio.
fprintf('\n  \n  %s \n  \n  ',' Gauss Seidel:     ');
fprintf('%s %s %s %s %s\n  ','k','x_{1}','x_{2}',...
                        'error_inf','error_ratio');
fprintf('%s \n  ' ,'----------------------------...
                            -------------------------');
fprintf ('%d %e %e %e %e\n  ',0,x0(1), x0(2), err0_1, ...
   err0_2 );
for k = 1:nmax
       xk =  RGS * x0 + invDL * b;   % Gauss-Seidel's method.
       x0 = xk;                      % Update solution.
     errt = norm(xe-x0,inf);
   err0_2 = errt / err0_1; err0_1 = errt;
   fprintf('%d %e %e %e %e\n  ',k,x0(1), x0(2), err0_1, ...
      err0_2 );
end
```

Based on part (b) as well as the numerical results presented here, Gauss-Seidel converges faster than Jacobi due to the fact that the former method has a smaller spectral radius than the latter. Note that the respective spectral radii of both methods appear in column 5.

Remarks: Recall that we discussed two different formulations of the iterative methods presented herein. For your convenience, they are listed here:

- Jacobi:

$$\begin{aligned} \mathbf{x}_{k+1} &= R_J\,\mathbf{x}_k + D^{-1}\mathbf{b}, \quad \text{and} \\ \mathbf{x}_{k+1} &= \mathbf{x}_k + D^{-1}\mathbf{r}_k. \end{aligned}$$

  Note that R_J is the iteration matrix mentioned above while D is the diagonal matrix extracted from A. Also, $\mathbf{r}_k = \mathbf{b} - A\mathbf{x}_k$.

- Gauss-Seidel:

$$\begin{aligned} \mathbf{x}_{k+1} &= R_{GS}\,\mathbf{x}_k + (L+D)^{-1}\mathbf{b}, \quad \text{and} \\ \mathbf{x}_{k+1} &= \mathbf{x}_k + E^{-1}\mathbf{r}_k. \end{aligned}$$

  In this case, L is a **strictly** lower triangular matrix whereas E is **a lower triangular** one, although both are extracted from the same matrix A. Finally, the matrix D and residual vector $\mathbf{r}_k$ are defined in the same way as above.

4. (20 points) Employ the Successive Over-Relaxation (SOR) method to solve the linear system

$$\begin{aligned} 2x_1 - x_2 &= 5, \\ -x_1 + 2x_2 - x_3 &= -2, \\ -x_2 + 2x_3 &= 2, \end{aligned}$$

with $\omega = 1.3$ and initial vector $\mathbf{x}_0 = [0\,0\,0]^T$. Stop the iterations when $\|\mathbf{r}_k\|_2 \leq \text{tol}\,\|\mathbf{b}\|_2$ holds with tol $= 10^{-10}$. Provide your MATLAB code and output which includes the solution.

Solution: From the linear system given, the matrix A and vector $\mathbf{b}$ are

$$A = \begin{bmatrix} 2 & -1 & 0 \\ -1 & 2 & -1 \\ 0 & -1 & 2 \end{bmatrix}, \quad \mathbf{b} = \begin{bmatrix} 5 \\ -2 \\ 2 \end{bmatrix}.$$

Note that the above matrix is a 3×3 **tridiagonal** one.

The MATLAB output and the script utilizing the SOR method are presented subsequently:

```
k      ||r_{k}||_{2}
-------------------
0      5.744563e+00
1      1.815528e+00
2      9.613257e-01
3      1.842586e-01
4      4.485713e-02
5      2.469100e-02
6      5.833320e-03
```

```
7      1.112535e-03
8      6.186183e-04
9      1.792999e-04
10     2.787215e-05
11     1.521032e-05
12     5.322942e-06
13     7.246924e-07
14     3.700819e-07
15     1.525349e-07
16     2.034011e-08
17     8.994857e-09
18     4.223741e-09
19     6.210435e-10
20     2.199694e-10

>> x0

x0 =

   3.249999999994529
   1.500000000082110
   1.749999999993994
```

```
clearvars; close all; clc; format long;

% The problem:
 A = [2,-1,0;-1,2,-1;0,-1,2];             % 3x3 tridiagonal ...
     matrix.
 b = [5;-2;2];                            % Right hand side ...
     vector.

% The setup:
   D = diag(diag(A));                     % Diagonal matrix.
   E = tril(A);                           % Lower triangular ...
       matrix.
 omg = 1.3;                               % \omega  .
cmat = omg * inv( (1-omg) * D + omg * E ); % Compute it once.
  x0 = [0;0;0];                           % Initial iterate.
 tol = 1e-10;                             % Tolerance.
nmax = 100;                               % Max # of iterations.
fprintf('%s \n  ','k      ||r_{k}||_{2}');
fprintf('%s \n  ','--------------------');
for k = 1:nmax
     res = b - A * x0;                    % Current residual.
```

```
        fprintf ('%d      %e      \n  ',k-1,norm(res,2) );
        xk = x0 + cmat *res;                       % SOR method.
         p = norm(b - A * xk,2);                   % Calculate norm.
        x0 = xk;                                   % Update stuff.
        if( p <= tol * norm(b,2) )                % Checkpoint
            fprintf ('%d      %e      \n  ',k, p );
            break
        end
end
```

Thus, the solution to the system above is

$$\mathbf{x} = [3.25\,1.5\,1.75]^T.$$

Remarks: Note also that there exist two different formulations for the SOR method, although both are identical to each other. For your convenience, they are listed here:

$$\mathbf{x}_{k+1} = R_{SOR}\,\mathbf{x}_k + \omega\,(\omega L + D)^{-1}\,\mathbf{b}, \quad \text{and}$$
$$\mathbf{x}_{k+1} = \mathbf{x}_k + \omega\,[(1-\omega)\,D + \omega E]^{-1}\,\mathbf{r}_k,$$

where L, D and E as well as $\mathbf{r}_k$ are defined similarly as in Question 3.

5. (20 points) Assume that $\omega \in [0.5, 1.8]$ and notice that the case with $\omega < 1$ corresponds to the Successive **Under**-Relaxation and $\omega > 1$ to SOR. Also, when $\omega = 1$, this is the original Gauss-Seidel method.

   Make a graph of the spectral radius of the iteration matrix:

$$R_{SOR} = (\omega L + D)^{-1}\,[(1-\omega)\,D - \omega U]\,,$$

   for the matrix A given in Question 4 as a function of ω. What is the **optimal value** of ω here, i.e., ω_{opt}? **Verify your answer** by running your code developed in Question 4 for $\omega = \omega_{\text{opt}}$. Include its output together with the figure for $\rho(R_{SOR})$ as a function of ω and the code producing it.

Solution: The corresponding MATLAB code follows:

```
clearvars; close all; clc; format long;

% The linear problem and setup:
 A = [2,-1,0;-1,2,-1;0,-1,2];
```

```
 b = [5;-2;2];
 L = tril(A,-1);                 % Strictly lower triangular ...
     matrix.
 U = triu(A,1);                  % Strictly upper triangular ...
     matrix.
 D = diag(diag(A));              % Diagonal matrix.

% For loop setup:
   nomg = 500;                   % # of sampling points.
omg_vec = linspace(0.5,1.8,nomg);% Create the vector for \omega ...
    's.
  rspec = zeros(nomg,1);         % Pre-allocate a vector.
for i = 1:nomg
      omg = omg_vec(i);
    invLD = inv(omg * L + D);
     cmat = invLD * ( ( 1-omg)*D - omg * U ); % SOR iteration ...
         matrix.
 rspec(i) = max(abs(eig(cmat)));              % Compute the radius.
end
% Plot the outcome:
figure;
set(gca,'FontSize',24,'Fontname','Times');
plot(omg_vec,rspec,'-k','LineWidth',2);
xlabel('$\omega  $','Interpreter','latex');
ylabel('$\rho  (R_{SOR})$','Interpreter','latex');
```

The plot showcasing the spectral radius as a function of ω is shown in Fig. 1. The optimal value of ω based on this plot is $\omega_{\text{opt}} \approx 1.172$ which corresponds to the value for which the spectral radius attains its minimum value. The MATLAB output of the script developed in Question 4 for this choice of ω follows:

```
k      ||r_{k}||_{2}
--------------------
0      5.744563e+00
1      1.436842e+00
2      7.299452e-01
3      1.017584e-01
4      3.838631e-02
5      5.693237e-03
6      1.601098e-03
7      2.431682e-04
8      5.965543e-05
9      9.125485e-06
10     2.070512e-06
11     3.162982e-07
12     6.823753e-08
```

```
13     1.035166e-08
14     2.156453e-09
15     3.233319e-10
```

A direct comparison with the results of Question 4 reveals that the number of iterations has been decreased by 5. This is something expected, i.e., less iterations are required to converge, since we found the optimal value of ω for the matrix A given. Note that the value of ω that we picked in Question 4 is not that far away from the optimal one.

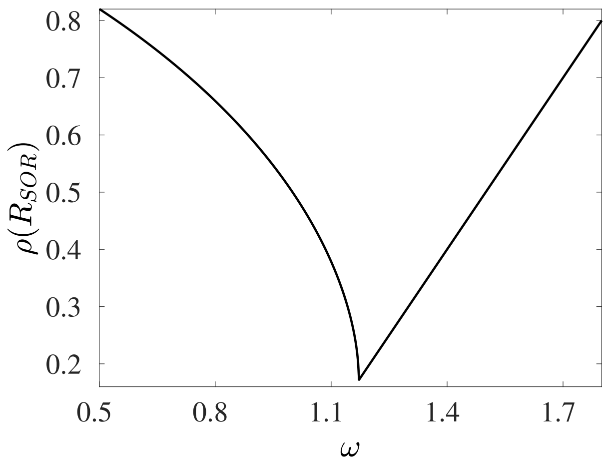

Figure 1: The spectral radius of the SOR method as a function of ω.

6. (20 points) Use your codes developed for the **Jacobi**, **Gauss-Seidel**, and **SOR** (with $\omega = 1.1$) iterative methods to solve the following linear system of equations:

$$\begin{bmatrix} 7 & 1 & -1 & 2 \\ 1 & 8 & 0 & -2 \\ -1 & 0 & 4 & -1 \\ 2 & -2 & -1 & 6 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \end{bmatrix} = \begin{bmatrix} 3 \\ -5 \\ 4 \\ -3 \end{bmatrix}.$$

Stop the iterations when $\|\mathbf{r}_k\|_2 \leq \text{tol}\,\|\mathbf{b}\|_2$ holds with tol $= 10^{-10}$. As per the initial guess (for **all** methods), use the **zero** vector, i.e., $\mathbf{x}_0 = [0\,0\,0\,0]^T$. Make a graph in a **semilog scale** showcasing the $\|\mathbf{r}_k\|_2$ against the number of iterations k in each case and compare your findings. Include **all** your codes, MATLAB output and solution.

Solution: The MATLAB script employing the Jacobi, Gauss-Seidel as well as SOR methods together with its output is given next:

```
clearvars; close all; clc; format long;

% Matrix containing the coefficients:
        A = [7,  1, -1,  2;
             1,  8,  0, -2; ...
            -1,  0,  4, -1; ...
             2, -2, -1,  6];
% Right-hand-side column vector:
        b = [3;-5;4;3];

% Executable statements:
D = diag(diag(A));                    % Diagonal matrix from A.
U = triu(A,1);                        % Strictly upper ...
   triangular matrix.
L = tril(A,-1);                       % Strictly lower ...
   triangular matrix.

% Construct the iteration matrices:
    invD = inv(D);                    % Compute it once!
   invDL = inv(D+L);                  % Compute it once!
      RJ = - invD * ( L + U );        % For Jacobi method.
     RGS = -invDL * U;                % For Gauss-Seidel method.
     omg = 1.1;                       % \omega   for SOR.
  invDoL = inv(D+omg*L);              % Compute it once!
    RSOR = invDoL*...                 % For SOR.
                ( (1-omg) * D - omg * U );

 % "Global variables":
      x0 = [0;0;0;0];                 % Initial iterate/guess.
    nmax = 1000;                      % Maximum number of ...
       iterations allowed.
     tol = 1e-10;                     % Tolerance.

% For Jacobi:
% =========================
fprintf('Jacobi:\n  ');
fprintf('%s \n  ','k      ||r_{k}||_{2}');
fprintf('%s \n  ','--------------------');
      x0J = x0;
  resJ(1) = norm(b - A * x0J,2);           % Current residual.
fprintf ('%d     %e     \n  ',0, resJ(1) );
ik=2;
for k = 1:nmax
      xkJ = RJ * x0J + invD * b;           % Jacobi's method.
      x0J = xkJ;                           % Update solution.
        p = norm(b - A * xkJ,2);           % Calculate norm.
 resJ(ik) = p;                             % Store residual.
        fprintf ('%d     %e     \n  ',k, p );
        ik = ik + 1;                       % Counter.
```

```
        if( p <= tol * norm(b,2) )          % Checkpoint.
            break
        end
end

% For Gauss-Seidel:
% =========================
fprintf('Gauss-Seidel:\n  ');
fprintf('%s \n  ','k       ||r_{k}||_{2}');
fprintf('%s \n  ','--------------------');
      x0GS = x0;
  resGS(1) = norm(b - A * x0GS,2);          % Current residual.
fprintf ('%d     %e     \n  ',0, resGS(1) );
ik=2;
for k = 1:nmax
      xkGS = RGS * x0GS +  invDL * b;       % Gauss-Seidel method.
      x0GS = xkGS;                          % Update solution.
         p = norm(b - A * xkGS,2);          % Calculate norm.
 resGS(ik) = p;                             % Store residual.
        fprintf ('%d     %e     \n  ',k, p );
        ik = ik + 1;                        % Counter.
        if( p <= tol * norm(b,2) )          % Checkpoint.
            break
        end
end

% For SOR:
% =========================
fprintf('SOR:\n  ');
fprintf('%s \n  ','k       ||r_{k}||_{2}');
fprintf('%s \n  ','--------------------');
      x0SOR = x0;
  resSOR(1) = norm(b - A * x0SOR,2);             % Current residual.
fprintf ('%d     %e     \n  ',0, resSOR(1) );
ik=2;
for k = 1:nmax

      xkSOR = RSOR * x0SOR +  omg * invDoL * b; % SOR method:
      x0SOR = xkSOR;                             % Update solution.
          p = norm(b - A * xkSOR,2);             % Calculate norm.
 resSOR(ik) = p;                                 % Store residual.
        fprintf ('%d     %e     \n  ',k, p );
        ik = ik + 1;                             % Counter.
        if( p <= tol * norm(b,2) )               % Checkpoint.
            break
        end
end

% Plot the outcome:
```

```
figure;
nj = length(resJ);
semilogy(0:nj-1,resJ,'linewidth',3);
hold on;
nGS = length(resGS);
semilogy(0:nGS-1,resGS,'linewidth',3);
hold on;
nSOR = length(resSOR);
semilogy(0:nSOR-1,resSOR,'linewidth',3);
set(gca,'fontsize',24,'fontname','times');
ht = legend('$\textrm {Jacobi}$', ...
                     '$\textrm {Gauss-Seidel}$', ...
                     '$\textrm {SOR}$');
set(ht,'interpreter','latex');
xlabel('$k$','interpreter','latex');
ylabel('$\|\mathbf {r}_{k}\|_{2}$','interpreter','latex');
xticks([0:10:40]);
yticks([10^(-10) 10^(-8) 10^(-6) 10^(-4) 10^(-2) 10^0]);
```

```
Jacobi:
 k      ||r_{k}||_{2}
---------------------
 0      7.681146e+00
 1      1.674863e+00
 2      7.337381e-01
 3      3.111902e-01
 4      1.673694e-01
 5      7.703977e-02
 6      4.103820e-02
 7      1.902076e-02
 8      1.011627e-02
 9      4.692206e-03
10      2.495123e-03
11      1.157400e-03
12      6.154456e-04
13      2.854860e-04
14      1.518065e-04
15      7.041834e-05
16      3.744477e-05
17      1.736948e-05
18      9.236173e-06
19      4.284376e-06
20      2.278206e-06
21      1.056790e-06
```

```
22      5.619451e-07
23      2.606690e-07
24      1.386101e-07
25      6.429695e-08
26      3.418973e-08
27      1.585957e-08
28      8.433283e-09
29      3.911940e-09
30      2.080164e-09
31      9.649235e-10
32      5.130960e-10

Gauss-Seidel:
 k       ||r_{k}||_{2}
---------------------
 0      7.681146e+00
 1      1.353182e+00
 2      9.926316e-02
 3      2.273975e-02
 4      5.599091e-03
 5      1.380927e-03
 6      3.406173e-04
 7      8.401699e-05
 8      2.072373e-05
 9      5.111742e-06
10      1.260869e-06
11      3.110074e-07
12      7.671347e-08
13      1.892224e-08
14      4.667383e-09
15      1.151263e-09
16      2.839717e-10

SOR:
 k       ||r_{k}||_{2}
---------------------
 0      7.681146e+00
 1      1.595471e+00
 2      2.440322e-01
 3      2.575564e-02
 4      1.945923e-03
```

```
 5      8.535639e-05
 6      1.139828e-05
 7      2.181574e-06
 8      2.618846e-07
 9      2.294761e-08
10      1.260682e-09
11      7.352819e-11
```

The iterations required for convergence (based on the tolerance criteria of the problem) are 32, 16 and 11 for Jacobi, Gauss-Seidel and SOR methods, respectively. Finally, the $\|\mathbf{r}_k\|_2$ as functions of k for each method are shown in Fig. 2 (see the legend therein). Clearly, the SOR performs better than the other two methods!

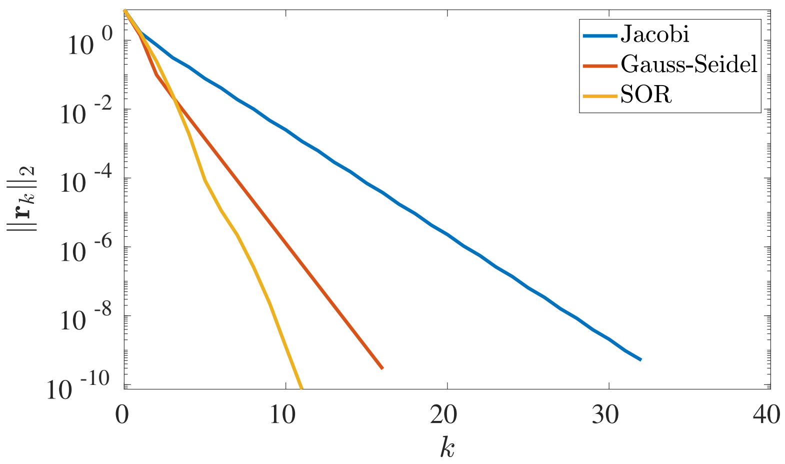

Figure 2: $\|\mathbf{r}_k\|_2$ as functions of k (see legend).

7. (20 points) Implement the Conjugate Gradient (CG) method in MATLAB (or in any other scientific programming language). To do so, write an m-file `my_cg.m`, the first line of which should be:

```
function xk = my_cg( A, b, x0, tol, nmax )
```

Test your code with the linear system given by

$$A = \begin{bmatrix} 2 & -1 & 0 \\ -1 & 2 & -1 \\ 0 & -1 & 2 \end{bmatrix}, \quad \mathbf{b} = \begin{bmatrix} 5 \\ -2 \\ 2 \end{bmatrix},$$

using initial vector $\mathbf{x}_0 = [0\,0\,0]^T$. Stop the iterations when $\langle \mathbf{r}_k, \mathbf{r}_k \rangle \le \text{tol}^2 \langle \mathbf{b}, \mathbf{b} \rangle$ holds and $\text{tol} = 10^{-10}$. Include **all** your codes and MATLAB output.

Solution: The main MATLAB script together with the function utilizing the conjugate gradient method and MATLAB output follow:

```
clearvars; close all; clc; format long;

  A = [ 2,-1,0;-1,2,-1;0,-1,2]; % Matrix.
  b = [5;-2;2];                 % Rhs vector.
 x0 = [0;0;0];                  % Initial vector.

% Parameters:
 tol = 1e-10; % Tolerance.
nmax = 100;    % Max # of iterations.

% Call the conjugate gradient method:
[ xk ] = my_cg( A, b, x0, tol, nmax );
xk

function [ xk ] = my_cg( A, b, x0, tol, nmax )
%
% This function utilizes the conjugate
% gradient (CG) method.
%
% Input:  1) nxn matrix A and rhs vector b.
%         2) Initial iterate x0.
%         3) User-prescribed tolerance tol.
%         4) Maximum number of iterations allowed.
%
% Output: 1) Solution vector xk.
%

% Initializations:
  bd = b'*b;
  r0 = b - A * u0;
  d0 = r0'*r0;
  p0 = r0;

 iflag = 0;
fprintf('%s \n  ','k       ||r_{k}||_{2}');
fprintf('%s \n  ','--------------------');
for k = 1:nmax

       fprintf ('%d     %e     \n  ',k-1,norm(r0,2) )
       sk = A * p0;
       ak = d0 / (p0'*sk);          % Find the minimizer.
       xk = x0 + ak * p0;           % Update x.
       rk = r0 - ak * sk;           % Update the residual.
       dk = rk'*rk;
       pk = rk + ( dk / d0 ) * p0; % Search directions.
```

```
        if( dk <= tol^2 * bd )      % Checkpoint
            fprintf ('%d     %e     \n   ',k,norm(rk,2) );
            iflag = 1;
            break
        end
        % Substitutions:
        x0 = xk;
        r0 = rk;
        p0 = pk;
        d0 = dk;
end
if(k==nmax&&iflag==0)
    disp('CG did not converge--increase iterations');
end
end
```

```
k      ||r_{k}||_{2}
--------------------
0     5.744563e+00
1     2.024102e+00
2     1.268727e+00
3     1.700355e-15

xk =

   3.250000000000000
   1.500000000000000
   1.750000000000000
```

Just 3 iterations are required to converge to the exact solution! Recall that using SOR with $\omega = 1.3$ and even with $\omega = \omega_{\text{opt}}$, 20 and 15 iterations were required to converge to the numerically exact solution, respectively.

Date: March 5, 2020

Mathematics Department, California Polytechnic State University, San Luis Obispo, CA 93407-0403, USA
Email address: echarala@calpoly.edu