Problems

1. Let $A \in \mathbb{R}^{n\times n}$ be a tridiagonal matrix where the diagonal entries are given by a_j for $j = 1, \dots, n$, the lower diagonal entries are b_j for $j = 2, \dots, n$ and the upper diagonal entries are c_j for $j = 1, \dots, n-1$.

(a) For $n = 3$, derive the LU factorization of the matrix A.

$$U = \begin{pmatrix} a_1 & c_1 & 0 \\ b_1 & a_2 & c_2 \\ 0 & b_2 & a_3 \end{pmatrix} \to \begin{pmatrix} a_1 & c_1 & 0 \\ 0 & a_2 - \frac{c_1 b_1}{a_1} & c_2 \\ 0 & b_2 & a_3 \end{pmatrix} \to \begin{pmatrix} a_1 & c_1 & 0 \\ 0 & a_2 - \frac{c_1 b_1}{a_1} & c_2 \\ 0 & 0 & a_3 - \frac{c_2 b_2}{a_2 - \frac{c_1 b_1}{a_1}} \end{pmatrix}$$

$$L = \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix} \to \begin{pmatrix} 1 & 0 & 0 \\ \frac{b_1}{a_1} & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix} \to \begin{pmatrix} 1 & 0 & 0 \\ \frac{b_1}{a_1} & 1 & 0 \\ 0 & \frac{b_2}{a_2 - \frac{c_1 b_1}{a_1}} & 1 \end{pmatrix}.$$

So, our LU factorization in $n = 3$ is given by

$$L = \begin{pmatrix} 1 & 0 & 0 \\ \frac{b_1}{a_1} & 1 & 0 \\ 0 & \frac{b_2}{a_2 - \frac{c_1 b_1}{a_1}} & 1 \end{pmatrix}$$

and

$$U = \begin{pmatrix} a_1 & c_1 & 0 \\ 0 & a_2 - \frac{c_1 b_1}{a_1} & c_2 \\ 0 & 0 & a_3 - \frac{c_2 b_2}{a_2 - \frac{c_1 b_1}{a_1}} \end{pmatrix}.$$

(b) What is the extension of the LU factorization for general n?

Looking at the $n = 3$ case, we can see that the next entry in U and L can be turned into an iterative process. The iteration process is as follows.

(1) Set U to be the zero $n \times n$ matrix and set L to be the $n \times n$ identity matrix.
(2) Set $U_{11} = a_1$.
(3) Set $k = 1$.
(4) Set $U_{k+1,k+1} = a_k - \frac{c_k b_k}{U_{k,k}}$.
(5) Set $U_{k,k+1} = c_k$.
(6) Set $L_{k+1,k} = \frac{b_k}{U_{k,k}}$
(7) Increase k by 1 and then repeat at step (4) until done.

(c) What is the operation count when applying Gaussian Elimination to a tridiagonal system without pivoting.

Looking at our operation count in part (b), we can see that step 1, 2, and 3, take 0 flops. Next, step 4 takes 3 flops. Because we are just doing Gaussian Elimination and we don't need to form LU, we can skip the rest of the steps except for the repeat step which occurs $n - 1$ times. Thus, the total cost is given by

$$3(n - 1) = 3n - 3 \text{ flops}.$$

2. Consider the linear system

$$\begin{aligned} 6x + 2y + 2z &= -2 \\ 2x + \frac{2}{3}y + \frac{1}{3}z &= 1 \\ x + 2y - z &= 0 \end{aligned}$$

(a) Verify that $(x, y, z) = (2.6, -3.8, -5)$ is the exact solution.

To verify the solution, let's first rewrite the LHS of the system and multiply by our vector to get

$$\begin{pmatrix} 6 & 2 & 2 \\ 2 & \frac{2}{3} & \frac{1}{3} \\ 1 & 2 & -1 \end{pmatrix} \begin{pmatrix} 2.6 \\ -3.8 \\ -5 \end{pmatrix} = \begin{pmatrix} -2 \\ 1 \\ 0 \end{pmatrix}$$

which shows the exact solution is given by $(x, y, z) = (2.6, -3.8, -5)$.

(b) Let's create our augmented matrix and begin Gaussian elimination

$$\begin{aligned} \left(\begin{array}{ccc|c} 6 & 2 & 2 & -2 \\ 2 & \frac{2}{3} & \frac{1}{3} & 1 \\ 1 & 2 & -1 & 0 \end{array}\right) &\sim \left(\begin{array}{ccc|c} 1 & 0.3333 & 0.3333 & -0.3333 \\ 2 & 0.6667 & 0.3333 & 1 \\ 1 & 2 & -1 & 0 \end{array}\right) \\ &\sim \left(\begin{array}{ccc|c} 1 & 0.3333 & 0.3333 & -2 \\ 0 & 0.0001 & -0.3333 & 1.666 \\ 0 & 1.666 & -1.333 & 0.3333 \end{array}\right) \\ &\sim \left(\begin{array}{ccc|c} 1 & 0.3333 & 0.3333 & -2 \\ 0 & 1 & -3333 & 16660 \\ 0 & 1.666 & -1.333 & 0.3333 \end{array}\right) \\ &\sim \left(\begin{array}{ccc|c} 1 & 0 & 1111 & -5554 \\ 0 & 1 & -3333 & 16660 \\ 0 & 0 & 5551 & -27740 \end{array}\right) \\ &\sim \left(\begin{array}{ccc|c} 1 & 0 & 1111 & -5554 \\ 0 & 1 & -3333 & 16660 \\ 0 & 0 & 1 & -4.997 \end{array}\right) \\ &\sim \left(\begin{array}{ccc|c} 1 & 0 & 0 & -3 \\ 0 & 1 & 0 & 10 \\ 0 & 0 & 1 & -4.997 \end{array}\right). \end{aligned}$$

So, our solution in 4 digit arithmetic without pivoting is given by $(x, y, z) = (-3, 10, -4.997)$ which has an absolute error of 14.893

(c) Repeat part (b) with partial pivoting.

$$\left(\begin{array}{ccc|c} 6 & 2 & 2 & -2 \\ 2 & \frac{2}{3} & \frac{1}{3} & 1 \\ 1 & 2 & -1 & 0 \end{array}\right) \sim \left(\begin{array}{ccc|c} 6 & 2 & 2 & -2 \\ 2 & 0.6667 & 0.3333 & 1 \\ 1 & 2 & -1 & 0 \end{array}\right)$$
$$\sim \left(\begin{array}{ccc|c} 6 & 2 & 2 & -2 \\ 0 & 0.0001 & 0.3333 & 1 \\ 0 & 1.666 & -1.333 & 0.3333 \end{array}\right)$$
$$\sim \left(\begin{array}{ccc|c} 6 & 2 & 2 & -2 \\ 0 & 1.666 & -1.333 & 0.3333 \\ 0 & 0.0001 & 0.3333 & 1 \end{array}\right)$$
$$\sim \left(\begin{array}{ccc|c} 6 & 2 & 2 & -2 \\ 0 & 1.666 & -1.333 & 0.3333 \\ 0 & 0 & 0.3333 & 1 \end{array}\right)$$

which implies $z = 3.000$, $y = 2.6$, and $x = -2.644$ which has an absolute error of 11.5091.

(d) Gaussian elimination with partial pivoting was slightly more accurate in this case and kept us from losing so many significant digits by reducing divisions by relatively small numbers.

3. Consider the system $Ax = b$ where

$$A = \begin{pmatrix} 4 & -1 & 0 & -1 & 0 & 0 \\ -1 & 4 & -1 & 0 & -1 & 0 \\ 0 & -1 & 4 & -1 & 0 & -1 \\ -1 & 0 & -1 & 4 & -1 & 0 \\ 0 & -1 & 0 & -1 & 4 & -1 \\ 0 & 0 & -1 & 0 & -1 & 4 \end{pmatrix} \quad b = \begin{pmatrix} 2 \\ 1 \\ 2 \\ 2 \\ 1 \\ 2 \end{pmatrix}.$$

All code used can be found at the end of the document With $x_0 = [111111]^T$,

(a) use Gauss-Jacobi iteration to approximate the solution to this problem $\varepsilon = 1e-7$.
Using Gauss-Jacobi iteration, my code converged in 37 iterations to a solution of

$$x = \begin{pmatrix} 1.166666550360919 \\ 1.208333174456728 \\ 1.458333174456728 \\ 1.458333174456728 \\ 1.208333174456728 \\ 1.166666550360919 \end{pmatrix}.$$

(b) use Gauss-Siedel iteration to approximate the solution to this problem $\varepsilon = 1e-7$.
Using Gauss-Siedel iteration, my code converged in 20 iterations to a solution of

$$x = \begin{pmatrix} 1.166666582106617 \\ 1.208333241597320 \\ 1.458333255911479 \\ 1.458333262516811 \\ 1.208333275046199 \\ 1.166666632739420 \end{pmatrix}.$$

(c) use SOR iteration with $\omega = 1.6735$ to approximate the solution to this problem $\varepsilon = 1e-7$.
Using SOR iteration, my code converged in 47 iterations to a solution of

$$x = \begin{pmatrix} 1.166666642205604 \\ 1.208333349651239 \\ 1.458333301323380 \\ 1.458333326774206 \\ 1.208333348402415 \\ 1.166666649988570 \end{pmatrix}.$$

(d) For this exact problem, Gauss-Siedel had the fastest convergence. In general, this will not be true, especially if we pick a nice ω for SOR. Furthermore, each of these methods is very sensitive to the input matrix which means performance of each method will very considerably from matrix to matrix.

(e) Set $c = \rho(B)$ (spectral radius). Use the following error estimate to derive error bounds for the last computed approximations with all methods.

$$\|x_{k+1} - x\| \leq \frac{c}{1-c}\|x_{k+1} - x_k\|$$

- For Gauss-Jacobi iteration, $B = -D^{-1}(L+U)$. Then, from my MATLAB code, the spectral radius is given of B is

  $$c = \rho(B) = 0.683012701892219$$

  which implies an error bound of

  $$\|x_{k+1}\| \leq \frac{c}{1-c}\|x_{k+1} - x_k\| = 2.1547\|x_{k+1} - x_k\|.$$

- For Gauss-Siedel iteration, $B = -(L+D)^{-1}U$. Then, from my MATLAB code, the spectral radius is given of B is

  $$c = \rho(B) = 0.480583134298243$$

  which implies an error bound of

  $$\|x_{k+1}\| \leq \frac{c}{1-c}\|x_{k+1} - x_k\| = 0.925236\|x_{k+1} - x_k\|.$$

- For SOR iteration, $B = -(D+\omega L)^{-1}(\omega U + (\omega - 1)D)$. Then, from my MATLAB code, the spectral radius is given of B is

  $$c = \rho(B) = 0.725728486720244$$

  which implies an error bound of

  $$\|x_{k+1}\| \leq \frac{c}{1-c}\|x_{k+1} - x_k\| = 2.64602\|x_{k+1} - x_k\|.$$

(f) What happens if you change the parameter ω for SOR?
Changing ω can change the convergence rate of SOR. As for the iterations, when ω is large, the iterations can jump further which can cause an oscillatory convergence pattern. If ω is too small, then the iterations move slower and slower but are also more predictable. For fastest convergence rates, we need some ω that balances movement with stability.

4. The linear system of equation
$$\begin{pmatrix} 1 & -a \\ -a & 1 \end{pmatrix} x = b$$
where $a \in \mathbb{R}$ under certain conditions can be solved by the iterative method
$$\begin{pmatrix} 1 & 0 \\ -\omega a & 1 \end{pmatrix} x_{k+1} = \begin{pmatrix} 1-\omega & \omega a \\ 0 & 1-\omega \end{pmatrix}$$

(a) For which values of a is the method convergent for $\omega = 1$?
First, let's compute the spectral radius of
$$\begin{pmatrix} 1 & 0 \\ -a & 1 \end{pmatrix}^{-1} \begin{pmatrix} 0 & a \\ 0 & 0 \end{pmatrix} = \begin{pmatrix} 1 & 0 \\ a & 1 \end{pmatrix} \begin{pmatrix} 0 & a \\ 0 & 0 \end{pmatrix} = \begin{pmatrix} 0 & a \\ 0 & a^2 \end{pmatrix}.$$
which has eigenvalues $\lambda = 0$ and $\lambda = a^2$ which implies the spectral radius is
$$\sigma(B) = a^2.$$
Then, for convergence, we must have $\sigma(B) < 1$ which implies $a^2 < 1$ or $a \in (-1, 1)$ for convergence.

(b) For $a = 0.5$, find the value of $\omega \in \{0.8, 0.9, 1.0, 1.1, 1.2, 1.3\}$ which minimizes the spectral radius of the matrix
$$\begin{pmatrix} 1 & 0 \\ -\omega a & 1 \end{pmatrix}^{-1} \begin{pmatrix} 1-\omega & \omega a \\ 0 & 1-\omega \end{pmatrix}.$$
First, let's expand this matrix out to get
$$\begin{pmatrix} 1 & 0 \\ -\omega a & 1 \end{pmatrix}^{-1} \begin{pmatrix} 1-\omega & \omega a \\ 0 & 1-\omega \end{pmatrix} = \begin{pmatrix} 1-\omega & \frac{1}{2}\omega \\ \frac{1}{2}\omega(1-\omega) & \frac{1}{4}(w-2)^2 \end{pmatrix}.$$
Then, using my MATLAB code, we can see that the spectral radius of our matrix is minimized when $\omega = 1.1$ which gives our matrix a spectral radius of 0.1.

Code Used

```
%%
% APPM 5600 Iterative Solvers
% Author: Caleb Jacobs
% Date Last Modified: 01-10-2021

%% Problem parameters
A = [4,-1,0,-1,0,0;   ...
     -1,4,-1,0,-1,0;  ...
     0,-1,4,-1,0,-1;  ...
     -1,0,-1,4,-1,0;  ...
     0,-1,0,-1,4,-1;  ...
     0,0,-1,0,-1,4];
b = [2;1;2;2;1;2];

%% Settings
format long
x0 = [1;1;1;1;1;1];
maxIts = 100;
tol = 1e-7;
omega = 1.6735;

%% Driver for iterative methods
GJ(x0, A, b, tol, maxIts)
GS(x0, A, b, tol, maxIts)
SOR(x0, A, b, omega, tol, maxIts)

%% Question 3 Spectral Radii
D = diag(diag(A));  % Diagonal entries of A
L = tril(A,-1);       % Lower triangular entries of A
U = triu(A,1);       % Upper triangular entries of A
jacSpec   = abs(eigs(-D\(L + U), 1, 'largestabs'))
seidSpec = abs(eigs(-(L + D) \ U, 1, 'largestabs'))
SORSpec   = abs(eigs(-(D + omega * L) \ (omega * U + (omega - 1) * D),
   1, 'largestabs'))

%% Question 4 Spectral Radii
omega = 0.8 : 0.1 : 1.3;
A = @(w) [1 - w, w / 2; w .* (1 - w) / 2, (w - 2).^2 / 4];
spec = zeros(length(omega), 1);
for i = 1:length(omega)
    spec(i) = eigs(A(omega(i)), 1, 'largestabs');
end
[sig, idx] = min(abs(spec));
sig                           % Minimum spectral radius
omega(idx)                    % Omega spectrum minimizer

%% Gauss-Jacobi iterative solver
function x = GJ(x0, A, b, tol, maxIts)
```

```
    fprintf('Gauss-Jacobi Iteration\n')
    k = 0;
    xf = x0;
    n = length(x0);


    while 1
        xi = xf;

        for i = 1:n
            tmp = 0;

            for j = 1:n
                if j ~= i
                    tmp = tmp + A(i,j) * xi(j);
                end
            end

            xf(i) = (b(i) - tmp) / A(i,i);
        end

        if norm(xf - xi,inf) < tol
            x = xf;
            k
            break;
        end
        if k >= maxIts
            x = NaN;
            break;
        end
        k = k + 1;
    end
end

%% Gauss-Seidel iterative solver
function x = GS(x0, A, b, tol, maxIts)
    fprintf('Gauss-Siedel Iteration\n')
    k = 0;
    xf = x0;
    n = length(x0);


    while 1
        xi = xf;

        for i = 1:n
            tmp = 0;

            for j = 1 : i-1
                tmp = tmp + A(i,j) * xf(j);
            end
```

```

                for j = i+1 : n
                    tmp = tmp + A(i,j) * xf(j);
                end

                xf(i) = (b(i) - tmp) / A(i,i);
            end

            if norm(xf - xi, inf) < tol
                x = xf;
                k
                break;
            end
            if k >= maxIts
                x = NaN;
                break;
            end
            k = k + 1;
        end
end

%% SOR Iterative Solver
function x = SOR(x0, A, b, omega, tol, maxIts)
    fprintf('SOR Iteration\n')
    k = 0;
    xf = x0;
    n = length(x0);

    while 1
        xi = xf;

        for i = 1:n
            tmp = 0;

            for j = 1:n
                if j ~= i
                    tmp = tmp + A(i,j) * xf(j);
                end
            end

            xf(i) = (1 - omega) * xf(i) + omega * (b(i) - tmp) / A(i,i)
    ;
        end

        if norm(xf - xi, inf) < tol
            x = xf;
            k
            break;
        end
        if k >= maxIts
            x = NaN;
```

```
                break;
            end
            k = k + 1;
        end
    end
```