Lendel Deguia

2.2.3

$$A=\begin{bmatrix}1&0&3&0\\0&1&3&-1\\3&-3&0&6\\0&2&4&-6\end{bmatrix} \Rightarrow \vec{S}=(3,3,6,6),\ \vec{l}=(1,2,3,4)$$

$K=1$: $\max\left\{\frac{|a_{11}|}{s_1}, \frac{|a_{21}|}{s_2}, \frac{|a_{31}|}{s_3}, \frac{|a_{41}|}{s_4}\right\}$

$=\max\left\{\frac{1}{3}, 0, \frac{1}{2}, 0\right\} \Rightarrow j=3 \Rightarrow l_K \leftrightarrow l_j$

$\Rightarrow \vec{l}=(3,2,1,4)$

$$\Rightarrow \begin{bmatrix}0&1&3&-2\\0&1&3&-1\\3&-3&0&6\\0&2&4&-6\end{bmatrix} \Rightarrow K=2.\ \max\left\{\frac{|a_{22}|}{s_2}, \frac{|a_{12}|}{s_1}, \frac{|a_{42}|}{s_4}\right\}$$

$= \max\left\{\frac{1}{3}, \frac{1}{3}, \frac{1}{3}\right\} \Rightarrow j=2$

$\Rightarrow \vec{l}=(3,2,1,4) \Rightarrow \begin{bmatrix}0&0&0&-1\\0&1&3&-1\\3&-3&0&6\\0&0&-2&-4\end{bmatrix}$

2.2.4 For

i) Naive Gaussian elimination, the next pivot element will be $\boxed{-0.0145}$

ii) For unscaled partial pivoting, the next pivot element will be $\max_{2\le i\le 4}|a_{i2}| = \boxed{102.7513}$

iii) For scaled partial pivoting, the next pivot element will be the one corresponding to $\max\left\{\frac{|a_{i2}|}{s_i} \,\middle|\, 2\le i\le 4\right\}$ where $\vec{S}=[987.6543, 833.3333, 102.7513, 9876.5432] \Rightarrow \max\left\{\frac{0.0145}{833.333}, \frac{102.7513}{102.7513}, \frac{1.3131}{9876.5432}\right\}$

$\Rightarrow \Rightarrow \boxed{102.7513}$

2.2.13

(a) The system yields the following matrix equation:

$$\begin{bmatrix} 3 & 4 & 3 \\ 1 & 5 & -1 \\ 6 & 3 & 7 \end{bmatrix}\begin{bmatrix} X_1 \\ X_2 \\ X_3 \end{bmatrix} = \begin{bmatrix} 10 \\ 7 \\ 15 \end{bmatrix} \Rightarrow \vec{S} = (4,5,7)$$

$\vec{l} = (1,2,3)$ ← initial index vector

$k=1 \Rightarrow \max\left\{\frac{3}{4}, \frac{1}{5}, \frac{6}{7}\right\} \Rightarrow j=3 \Rightarrow \boxed{\vec{l} = (3,2,1)}$

$$\Rightarrow \begin{bmatrix} 0 & 2.500 & -0.5000 \\ 0 & 4.500 & -2.167 \\ 6.000 & 3.000 & 7.000 \end{bmatrix}\begin{bmatrix} X_1 \\ X_2 \\ X_3 \end{bmatrix} = \begin{bmatrix} 2.500 \\ 4.500 \\ 15.00 \end{bmatrix}$$

switched l_k with l_j

$k=2 \Rightarrow \max\left\{\frac{|a_{l_2 2}|}{s_{l_2}}, \frac{|a_{l_3 2}|}{s_{l_3}}\right\} = \max\left\{\frac{|a_{22}|}{s_2}, \frac{|a_{12}|}{s_1}\right\}$ ($j=2$, $j=3$)

$= \max\left\{\frac{4.500}{5}, \frac{2.500}{4}\right\} \Rightarrow j=2 \Rightarrow l_k \leftrightarrow l_j \Rightarrow$ no change to $\vec{l}$

$\Rightarrow \boxed{\vec{l} = (3,2,1)}$

$$\Rightarrow \begin{bmatrix} 0 & 0 & 0.7039 \\ 0 & 4.500 & -2.167 \\ 6.000 & 3.000 & 7.000 \end{bmatrix}\begin{bmatrix} X_1 \\ X_2 \\ X_3 \end{bmatrix} = \begin{bmatrix} 0 \\ 4.500 \\ 15.00 \end{bmatrix}$$

$\Rightarrow$ $0.7039 X_3 = 0 \Rightarrow X_3 = 0$

$4.500 X_2 - 2.167 X_3 = 4.500 \Rightarrow X_2 = \frac{4.500}{4.500} = 1$

$6.000 X_1 + 3.000 X_2 + 7.000 X_3 = 15.00 \Rightarrow X_1 = \frac{12.00}{6.000} = 2$

$\therefore$ $\boxed{\begin{aligned} X_1 &= 2 \\ X_2 &= 1 \\ X_3 &= 0 \end{aligned}}$

8.4.4 By definition of a subordinate matrix norm:

$$\|I\| = \sup\{\|I\vec{x}\| : \vec{x}\in\mathbb{R}^n \wedge \|\vec{x}\| = 1\}$$
$$= \sup\{\|x\| : \vec{x}\in\mathbb{R}^n \wedge \|\vec{x}\| = 1\}$$
$$= \sup\{1\} \Rightarrow \boxed{\|I\| = 1 ; (b)}$$

8.4.7 By the Jacobi and Gauss-Seidel convergence theorem, a sufficient condition for the convergence of the Jacobi method on $A\vec{x} = \vec{b}$ is that

$\boxed{A \text{ is diagonally dominant} ; (b)}$

8.4.8 By the Jacobi and Gauss-Seidel convergence theorem, a sufficient condition for the convergence of the Gauss-Seidel method on $A\vec{x} = \vec{b}$ is that $\boxed{A \text{ is diagonally dominant} ; (a)}$

8.4.11 Here, we use the L_1 norm for the matrix norm: $\|A\|_1 = \max_{1\le j\le n}\left(\sum_{i=1}^{n}|a_{ij}|\right)$

(a) $A = \begin{bmatrix} -2 & 1 & 0 \\ 1 & -2 & 1 \\ 0 & 1 & -2 \end{bmatrix} \Rightarrow A^{-1} = \begin{bmatrix} -3/4 & -1/2 & -1/4 \\ -1/2 & -1 & -1/2 \\ -1/4 & -1/2 & -3/4 \end{bmatrix} \Rightarrow \|A\|_1 = \max\{3,4\} = 4$

$\|A^{-1}\|_1 = \max\{\frac{3}{2}, 2\} = 2$

$\Rightarrow K(A) = \|A\|_1\|A^{-1}\|_1 \Rightarrow \boxed{K(A) = 8}$

(c) $A = \begin{bmatrix} 3 & 0 & 0 \\ 0 & 2 & 0 \\ 0 & 0 & 1 \end{bmatrix} \Rightarrow A^{-1} = \begin{bmatrix} 1/3 & 0 & 0 \\ 0 & 1/2 & 0 \\ 0 & 0 & 1 \end{bmatrix} \Rightarrow \|A\|_1 = \max\{1,2,3\} = 3$

$\|A^{-1}\|_1 = \max\{\frac{1}{3}, \frac{1}{2}, 1\} = 1$

$\Rightarrow K(A) = \|A\|_1\|A^{-1}\|_1 \Rightarrow \boxed{K(A) = 3}$

Computer Exercise 2.2.2

The following program will use Gaussian elimination with scaled partial pivoting to solve the following $\mathbf{Ax} = \mathbf{b}$ system:

$$\begin{bmatrix} 0.4096 & 0.1234 & 0.3678 & 0.2943 \\ 0.2246 & 0.3872 & 0.4015 & 0.1129 \\ 0.3645 & 0.1920 & 0.3781 & 0.0643 \\ 0.1784 & 0.4002 & 0.2786 & 0.3927 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \end{bmatrix} = \begin{bmatrix} 0.4043 \\ 0.1550 \\ 0.4240 \\ 0.2557 \end{bmatrix}$$

Here, procedures *Gauss* and *Solve* have been combined into one method, *gespp*, which outputs the solution $\mathbf{x}$ along with the final states of $\mathbf{A}$ and $\mathbf{b}$. Only the solution $\mathbf{x}$ will be displayed.

```
A = [0.4096, 0.1234, 0.3678, 0.2943;
    0.2246, 0.3872, 0.4015, 0.1129;
    0.3645, 0.1920, 0.3781, 0.0643;
    0.1784, 0.4002, 0.2786, 0.3927];
b = [0.4043, 0.1550, 0.4240, 0.2557]';
[Amod, bmod, x] = gespp(A,b);
x
```

```
x = 4×1
    3.4606
    1.5610
   -2.9342
   -0.4301
```

Just to make sure that this is the right solution, we can check that $\mathbf{Ax}$ provides the same $\mathbf{b} = \begin{bmatrix} 0.4043 \\ 0.1550 \\ 0.4240 \\ 0.2557 \end{bmatrix}$

```
A*x
```

```
ans = 4×1
    0.4043
    0.1550
    0.4240
    0.2557
```

Indeed, we end up with the same $\mathbf{b}$, so this is the correct solution.

```
function [Amod, bmod, x] = gespp(A,b)
    n = length(b);
    %set index vector
    l = (1:n);
    %set scale vector
    s = zeros(length(l), 1);
```

```
    for i = 1:n
        s(i) = max(abs(A(i, :)));
    end
    %forward elimination
    for k = 1:(n-1)
        max_r = 0;
        pivot_index = l(1);
        for i = k:n
            if (abs(A(l(i), k))/s(l(i))) > max_r
                pivot_index = i;
                max_r = (abs(A(l(i), k))/s(l(i)));
            end
        end
        a = l(pivot_index);
        l(pivot_index) = l(k);
        l(k) = a;
        for i = (k+1):n
            mult = A(l(i), k)/A(l(k), k);
            for j = k:n
                A(l(i), j) = A(l(i), j) - mult*A(l(k), j);
            end
            b(l(i)) = b(l(i)) - mult*b(l(k));
        end
    end
    Amod = A;
    bmod = b;
    %back substitution
    x = zeros(n, 1);
    x(n) = b(l(n))/A(l(n), n);
    for u = (n-1):-1:1
        sum = 0;
        for v = (u+1):n
            sum = sum + (A(l(u), v)*x(v));
        end
        x(u) = (b(l(u)) - sum)/(A(l(u), u));
    end
end
```

Computer Exercise 2.2.3

The following program will use Gaussian elimination with scaled partial pivoting to solve the following $\mathbf{Ax} = \mathbf{b}$ system:

$$\begin{bmatrix} 0.4096 & 0.1234 & 0.3678 & 0.2943 \\ 0.2246 & 0.3872 & 0.4015 & 0.1129 \\ \underline{\mathbf{0.3345}} & 0.1920 & 0.3781 & 0.0643 \\ 0.1784 & 0.4002 & 0.2786 & 0.3927 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \end{bmatrix} = \begin{bmatrix} 0.4043 \\ 0.1550 \\ 0.4240 \\ 0.2557 \end{bmatrix}$$

Here, the matrix equation is mostly the same as in computer exercise 2.2.2 except element a_{31} of $\mathbf{A}$ is modified from 0.3645 to 0.3345 (the modification is indicated by the boldface and underline), so as to simulate slight mistypes in data entry in order for us to study the effects of minor perturbations to the matrix system. We will compare the solutions of this system and that of 2.2.2 by taking the ratio of the respective components where we designate the solution of this system as $\mathbf{x}_{est}$ and the solution of 2.2.2 as $\mathbf{x}_{sol}$,

```
A = [0.4096, 0.1234, 0.3678, 0.2943;
    0.2246, 0.3872, 0.4015, 0.1129;
    0.3345, 0.1920, 0.3781, 0.0643;
    0.1784, 0.4002, 0.2786, 0.3927];
b = [0.4043, 0.1550, 0.4240, 0.2557]';
[Amod2, bmod2, xest] = gespp(A2,b2);
xest
```

```
xest = 4×1
    6.7831
    3.5914
   -6.4451
   -1.5179
```

Now, we compare this solution to the solution from 2.2.2 by determining the following component wise ratio $|\mathbf{x}_{est}^{(i)}/\mathbf{x}_{sol}^{(i)}|,\ i \in \{1,2,3,4\}$.

```
%acquire x from problem 2.2.2
xsol = x;
factordif = abs(xest./xsol)
```

```
factordif = 4×1
    1.9601
    2.3008
    2.1965
    3.5296
```

It turns out that each element of $\mathbf{x}_{est}$ is overall at least twice as much as $\mathbf{x}_{sol}$. We can explore why this is by considering the condition number of the original matrix from 2.2.2:

```
%original matrix
A1 = [0.4096, 0.1234, 0.3678, 0.2943;
    0.2246, 0.3872, 0.4015, 0.1129;
```

```
    0.3645, 0.1920, 0.3781, 0.0643;
    0.1784, 0.4002, 0.2786, 0.3927];

n1a = norm(A1, 2);
n1b = norm(inv(A1), 2);
condition_number = n1a*n1b
```

```
ans = 46.1393
```

We see that the condition number is mildly far away from one, but it isn't overwhelmingly large. However, considering the effects of just a single data point (matrix entry) on the final solution, we might as well consider it ill conditioned enough since we can easily imagine this much of a difference leading to catastrophic consequences (e.g. perhaps a collapsing of a bridge resulting from a slightly erroneous solution for a mathematical model of some dynamical system). We can explore this further by considering the determinant of the original matrix

```
det(A1)
```

```
ans = -0.0024
```

The determinant is fairly close to zero which makes the near singluarity aspect of the original matrix clearer, so it is no wonder, then, why just a slight modification resulted in a different final solution.

```
function [Amod, bmod, x] = gespp(A,b)
    n = length(b);
    %set index vector
    l = (1:n);
    %set scale vector
    s = zeros(length(l), 1);
    for i = 1:n
        s(i) = max(abs(A(i, :)));
    end
    %forward elimination
    for k = 1:(n-1)
        max_r = 0;
        pivot_index = l(1);
        for i = k:n
            if (abs(A(l(i), k))/s(l(i))) > max_r
                pivot_index = i;
                max_r = (abs(A(l(i), k))/s(l(i)));
            end
        end
        a = l(pivot_index);
        l(pivot_index) = l(k);
        l(k) = a;
        for i = (k+1):n
            mult = A(l(i), k)/A(l(k), k);
            for j = k:n
                A(l(i), j) = A(l(i), j) - mult*A(l(k), j);
            end
            b(l(i)) = b(l(i)) - mult*b(l(k));
        end
```

```
    end
    Amod = A;
    bmod = b;
    %back substitution
    x = zeros(n, 1);
    x(n) = b(l(n))/A(l(n), n);
    for u = (n-1):-1:1
        sum = 0;
        for v = (u+1):n
            sum = sum + (A(l(u), v)*x(v));
        end
        x(u) = (b(l(u)) - sum)/(A(l(u), u));
    end
end
```

Computer Exercise 8.4.3

The following program will produce approximate solutions to the following $\mathbf{Ax} = \mathbf{b}$ system

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

using the Jacobi method, the Gauss-Seidel method and the SOR method (with $\omega = 1.4$). In each case, a maximum amount of iterations $k_{\max}$ (selected based on testing each method) and an error tolerance of 10^{-4} (four decimal places of accuracy); the error is determined by $\text{error} = \max_{1 \le i \le n} |\mathbf{x}_{exact}^{(i)} - \mathbf{x}_{approx}^{(i)}|, i \in \{1, \ldots, n\}$ (here, the "i" indicates the component i^{th} of the corresponding vectors) where $\mathbf{x}_{exact} = [-1, 1, -1, 1]^T$ and in this case, $n = 4$. Each algorithm corresponding to each method is designed so that the program will exit when $\text{error} \le 10^{-4}$ or when $k \ge k_{\max}$ (however, the value for $k_{\max}$ is picked such that only the former condition is met).The error and iteration number evaluated upon exit of the program will be displayed along with $\mathbf{x}_{approx}$.

```
A = [7, 3, -1, 2; 3, 8, 1, -4; -1, 1, 4, -1; 2, -4, -1, 6];
b = [-1, 0, -3, 1]';
x0 = [0, 0, 0, 0]';
x_exact = [-1, 1, -1, 1]';


err_tol = (10^(-4));
kmax = 80;
xapprox = jacobi(A, b, x0, kmax, x_exact, err_tol)
```

```
error tolerance satisfied
exited at k = 73, error: 8.79977e-05
xapprox = 4×1
   -0.9999
    0.9999
   -1.0000
    0.9999
```

```
err_tol = (10^(-4));
kmax = 40;
xapprox = gaussseidel(A, b, x0, kmax, x_exact, err_tol)
```

```
error tolerance satisfied
exited at k = 37, error: 9.98589e-05
xapprox = 4×1
   -0.9999
    0.9999
   -1.0000
    0.9999
```

```
err_tol = (10^(-4));
kmax = 20;
w = 1.4;
```

```
xapprox = sor(A, b, x0, kmax, x_exact, err_tol, w)
```

```
error tolerance satisfied
exited at k = 12, error: 7.88288e-05
xapprox = 4×1
   -1.0001
    1.0001
   -1.0000
    1.0001
```

The SOR method provides the fastest convergence where it exits at $k = 12$; the Jacobi method and Gauss-Seidel method exit at $k = 73$ and $k = 37$ respectively.

```
function x = jacobi(A, b, x0, kmax, x_exact, err_tol)
    xkm1 = x0;
    n = length(b);
    k = 1;
    error = ones(n, 1);
    while (k < kmax) && (max(error) > err_tol)
        xk = zeros(n, 1);
        for i = 1:n
            sum = 0;
            for j = 1:n
                if j ~= i
                    sum = sum + ((A(i,j)/A(i,i))*xkm1(j));
                end
            end
            xk(i) = (b(i)/A(i, i)) - sum;
        end
        xkm1 = xk;
        k = k + 1;
        error = abs(xk - x_exact);
        if k >= kmax
            disp('max iterations reached')
            fprintf('exited at k = %d, error: %5.5e \n', k, max(error))
        elseif max(error) <= err_tol
            disp('error tolerance satisfied')
            fprintf('exited at k = %d, error: %5.5e \n', k, max(error))
        end
    end
    x = xk;
end

function x = gaussseidel(A, b, x0, kmax, x_exact, err_tol)
    xkm1 = x0;
    n = length(b);
    k = 1;
    error = ones(n, 1);
    while (k < kmax) && (max(error) > err_tol)
        xk = zeros(n, 1);
        for i = 1:n
            sum = 0;
            for j = 1:n
                if j < i
```

```
                    sum = sum + ((A(i,j)/A(i,i))*xk(j));
                elseif j > i
                    sum = sum + ((A(i,j)/A(i,i))*xkm1(j));
                end
            end
            xk(i) = (b(i)/A(i, i)) - sum;
        end
        xkm1 = xk;
        k = k + 1;
        error = abs(xk - x_exact);
        if k >= kmax
            disp('max iterations reached')
            fprintf('exited at k = %d, error: %5.5e \n', k, max(error))
        elseif max(error) <= err_tol
            disp('error tolerance satisfied')
            fprintf('exited at k = %d, error: %5.5e \n', k, max(error))
        end
    end
    x = xk;
end

function x = sor(A, b, x0, kmax, x_exact, err_tol, w)
    xkm1 = x0;
    n = length(b);
    k = 1;
    error = ones(n, 1);
    while (k < kmax) && (max(error) > err_tol)
        xk = zeros(n, 1);
        for i = 1:n
            sum = 0;
            for j = 1:n
                if j < i
                    sum = sum + ((A(i,j)/A(i,i))*xk(j));
                elseif j > i
                    sum = sum + ((A(i,j)/A(i,i))*xkm1(j));
                end
            end
            xk(i) = w*((b(i)/A(i, i)) - sum) + (1-w)*xkm1(i);
        end
        xkm1 = xk;
        k = k + 1;
        error = abs(xk - x_exact);
        if k >= kmax
            disp('max iterations reached')
            fprintf('exited at k = %d, error: %5.5e \n', k, max(error))
        elseif max(error) <= err_tol
            disp('error tolerance satisfied')
            fprintf('exited at k = %d, error: %5.5e \n', k, max(error))
        end
    end
    x = xk;
end
```