Exercise 1. (Component Skill 4.1)

Consider the following ridiculously simple linear system:

$$\begin{pmatrix} 1 & 0 \\ 0 & 2 \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \end{pmatrix} = \begin{pmatrix} 1 \\ 1 \end{pmatrix}$$

a. What is the solution of this linear system? You don't have to show your work.

b. Set up a Neumann iteration with initial guess x0 = (0, 0)T . Calculate x1, x2, x3, and x4. Show your work.

c. Based on your calculations in (b), what will xk equal for k > 0 when (i) k is odd and (ii) k is even? You don't have to show your work.

d. Based on your answer to (c), does the Neumann iteration converge?

e. Let M represent the matrix used in your Neumann iteration. Does M^n → 0 as n → ∞? What does this tell you about the convergence of the Neumann iteration method? Is this consistent with your answer to (d)?

Solution

a. The solution to the linear system:

$$\begin{pmatrix} x_1 \\ x_2 \end{pmatrix} = \begin{pmatrix} 1 \\ 0.5 \end{pmatrix}$$

b. Neumann iteration with initial guess x0 = (0, 0)T:

Let A =

$$\begin{pmatrix} 1 & 0 \\ 0 & 2 \end{pmatrix}$$

and b =

$$\begin{pmatrix} 1 \\ 1 \end{pmatrix}$$

Neumann iteration: $x_{k+1} = A^{-1}(b - Ax_k)$

$x_1 = A^{-1}(b - Ax_0) = A^{-1}b$

$$x_1 = \begin{pmatrix} 1 \\ 0.5 \end{pmatrix}$$

$x_2 = A^{-1}(b - Ax_1)$

$$x_2 = \begin{pmatrix} 1 \\ 0.5 \end{pmatrix}$$

$x_3 = A^{-1}(b - Ax_2)$

$$x_3 = \begin{pmatrix} 1 \\ 0.5 \end{pmatrix}$$

$x_4 = A^{-1}(b - Ax_3)$

$$x_4 = \begin{pmatrix} 1 \\ 0.5 \end{pmatrix}$$

c. For k > 0, (i) when k is odd, and (ii) when k is even:

$$x_k = \begin{pmatrix} 1 \\ 0.5 \end{pmatrix}$$

d. The Neumann iteration converges because the sequence of approximations x_k remains constant.

e. The matrix M used in the Neumann iteration is A^{-1}:

$$M = \begin{pmatrix} 1 & 0 \\ 0 & 0.5 \end{pmatrix}$$

M^n → 0 as n → ∞:

$$\lim_{n \to \infty} M^n = \begin{pmatrix} 1 & 0 \\ 0 & 0 \end{pmatrix}$$

This implies that the Neumann iteration method does not converge. However, this result is inconsistent with the answer to (d), as the iteration does converge

Exercise 2. (Component Skill 4.2)

For each linear system, decide whether Jacobi iteration is guaranteed or not guaranteed to converge to the solution. Justify each of your decisions in a sentence or two.

a.

$$\begin{pmatrix} 3 & 1 & 0 \\ 1 & 3 & 1 \\ 0 & 1 & 3 \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \\ x_3 \end{pmatrix} = \begin{pmatrix} 1 \\ 1 \\ 1 \end{pmatrix}$$

Guaranteed to converge because the matrix is strictly diagonally dominant.

b.

$$\begin{pmatrix} -3 & 1 & 0 \\ 1 & -3 & 1 \\ 0 & 1 & -3 \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \\ x_3 \end{pmatrix} = \begin{pmatrix} 1 \\ 1 \\ 1 \end{pmatrix}$$

Not guaranteed to converge because the matrix has negative diagonal elements and is not strictly diagonally dominant.

c.

$$\begin{pmatrix} 1 & 3 & 0 \\ 3 & 1 & 3 \\ 0 & 3 & 1 \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \\ x_3 \end{pmatrix} = \begin{pmatrix} 1 \\ 1 \\ 1 \end{pmatrix}$$

Not guaranteed to converge because the matrix is not diagonally dominant.

d.

$$\begin{pmatrix} 1 & -3 & 0 \\ -3 & 1 & -3 \\ 0 & -3 & 1 \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \\ x_3 \end{pmatrix} = \begin{pmatrix} 1 \\ 1 \\ 1 \end{pmatrix}$$

Not guaranteed to converge because the matrix is not diagonally dominant.

e.

$$\begin{pmatrix} 2 & 1 & 0 \\ 1 & 2 & 1 \\ 0 & 1 & 2 \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \\ x_3 \end{pmatrix} = \begin{pmatrix} 1 \\ 1 \\ 1 \end{pmatrix}$$

Guaranteed to converge because the matrix is strictly diagonally dominant.

HW4/HW4.m

```
% Mason Wheeler
% 4/25/2023

% Exercise 1
format long;

A = [3, 1, 0; 1, 3, 1; 0, 1, 3];
b = [1; 1; 1];
x0 = [0; 0; 0];
N = 10;

% a.
x10 = JacobiHW4(A, b, x0, N);
A1 = x10;

% b.
A2 = A \ b;

% c.
E = max(abs(x10 - A2));
A3 = E;

% Exercise 2
% a.
x10_gs = GaussSeidelHW4(A, b, x0, N);
A4 = x10_gs;

% b.
E_gs = max(abs(x10_gs - A2));
A5 = E_gs;

% Exercise 3
omega = 1.09;

% a.
x10_sor = SORHW4(A, b, x0, N, omega);
A6 = x10_sor;

% b.
E_sor = max(abs(x10_sor - A2));
A7 = E_sor;

% Exercise 4
threshold = 1e-10;

% a.
[x_jacobi_cauchy, N_jacobi] = JacobiCauchyHW4(A, b, x0, threshold);
A8 = N_jacobi;

% b.
[x_gs_cauchy, N_gs] = GaussSeidelCauchyHW4(A, b, x0, threshold);
A9 = N_gs;
```

```
% c.
[x_sor_cauchy, N_sor] = SORCauchyHW4(A, b, x0, threshold, omega);
A10 = N_sor;

% Function definitions
function [x1, N] = JacobiCauchyHW4(A, b, x0, threshold)
    % INPUTS: A is a nxn coefficient matrix
    % b is a nx1 column vector of knowns
    % x0 is the nx1 initial guess of the solution
    % threshold is the Cauchy error threshold
    % OUTPUTS: x1 is the nx1 solution
    % N is the number of iterations

    D = diag(diag(A));
    L = tril(A, -1);
    U = triu(A, 1);

    x1 = D \ (-(L + U) * x0 + b);
    err = max(abs(x1 - x0));
    N = 1;

    while err > threshold
        xtemp = x1;
        x1 = D \ (-(L + U) * x1 + b);
        x0 = xtemp;
        err = max(abs(x1 - x0));
        N = N + 1;
    end
end


function [x1, N] = GaussSeidelCauchyHW4(A, b, x0, threshold)
    n = length(b);
    x1 = x0;
    err = threshold + 1;
    N = 0;

    while err > threshold
        x_prev = x1;
        for i = 1:n
            x1(i) = (1 / A(i, i)) * (b(i) - A(i, [1:i-1, i+1:n]) * x1([1:i-1, i+1:n]));
        end
        err = max(abs(x1 - x_prev));
        N = N + 1;
    end
end

function [x1, N] = SORCauchyHW4(A, b, x0, threshold, omega)
    n = length(b);
    x1 = x0;
    err = threshold + 1;
    N = 0;

    while err > threshold
        x_prev = x1;
```

```
        for i = 1:n
            x1(i) = (1 - omega) * x1(i) + (omega / A(i, i)) * (b(i) - A(i, :) * x1 + A(i,
i) * x1(i));
        end
        err = max(abs(x1 - x_prev));
        N = N + 1;
    end
end

% Function definition
function [x1] = SORHW4(A, b, x0, N, omega)
    n = length(b);
    x1 = x0;
    for k = 1:N
        for i = 1:n
            x1(i) = (1 - omega) * x1(i) + (omega / A(i, i)) * (b(i) - A(i, :) * x1 + A(i,
i) * x1(i));
        end
    end
end


% Function definition
function [x1] = GaussSeidelHW4(A, b, x0, N)
    n = length(b);
    x1 = x0;
    for k = 1:N
        for i = 1:n
            x1(i) = (b(i) - A(i, :) * x1 + A(i, i) * x1(i)) / A(i, i);
        end
    end
end


% Function definition
function [x1] = JacobiHW4(A, b, x0, N)
    D = diag(diag(A));
    L = tril(A, -1);
    U = triu(A, 1);

    for j = 1:N
        x1 = D \ (-(L + U) * x0 + b);
        x0 = x1;
    end
end
```