# Problem 1 Question (a)

## 1. Theoretical Convergence Analysis

### The First System

#### 1. Calculation of and

* The iteration matrixfor the Jacobi method is calculated using:
* The iteration matrix for the Gauss-Seidel method is calculated using:

We have:

#### 2. Calculation of Spectral Radius

both the Jacobi and Gauss-Seidel methods should converge

### The Second System

#### 1. Calculation of and

#### 2. Calculation of Spectral Radius

Only the Gauss-Seidel method should converge.

### The Third System

#### 1. Calculation of and

#### 2. Calculation of Spectral Radius

Neither the Jacobi nor the Gauss-Seidel methods should converge.

## 2. MATLAB:

**Hence, we check that Jacobi method can only converge for the first system, and Gauss-Seidel method can only converge for the first system and second system.**

% Output

% For the first system of equations:

% Solution using Jacobi method: [-0.72727;0.72727;0.090909], Number of iterations: 18

% Solution using Gauss-Seidel method: [-0.72728;0.72727;0.090902], Number of iterations: 12

% For the second system of equations:

% Solution using Jacobi method: [-1.014e+204;-1.014e+204;-1.014e+204], Number of iterations: 1000

% Solution using Gauss-Seidel method: [5.7692;0.76922;-4.2307], Number of iterations: 37

% For the third system of equations:

% Solution using Jacobi method: [Inf;-Inf], Number of iterations: 1000

% Solution using Gauss-Seidel method: [-Inf;-Inf], Number of iterations: 1000

% Define matrices and vectors

A1 = [6, 2, -1; 1, 4, -2; -3, 2, 4];

b1 = [-3; 2; 4];

A2 = [1, 0.8, 0.8; 0.8, 1, 0.8; 0.8, 0.8, 1];

b2 = [3; 2; 1];

A3 = [1, 3; -7, 1];

b3 = [4; 6];

% Solve equations using Jacobi and Gauss-Seidel methods

[x1\_jacobi, iter1\_jacobi] = jacobi(A1, b1, 1e-5, 1000);

[x1\_gauss\_seidel, iter1\_gauss\_seidel] = gauss\_seidel(A1, b1, 1e-5, 1000);

[x2\_jacobi, iter2\_jacobi] = jacobi(A2, b2, 1e-5, 1000);

[x2\_gauss\_seidel, iter2\_gauss\_seidel] = gauss\_seidel(A2, b2, 1e-5, 1000);

[x3\_jacobi, iter3\_jacobi] = jacobi(A3, b3, 1e-5, 1000);

[x3\_gauss\_seidel, iter3\_gauss\_seidel] = gauss\_seidel(A3, b3, 1e-5, 1000);

% Display results

fprintf('For the first system of equations:\n');

fprintf('Solution using Jacobi method: %s, Number of iterations: %d\n', mat2str(x1\_jacobi, 5), iter1\_jacobi);

fprintf('Solution using Gauss-Seidel method: %s, Number of iterations: %d\n', mat2str(x1\_gauss\_seidel, 5), iter1\_gauss\_seidel);

fprintf('For the second system of equations:\n');

fprintf('Solution using Jacobi method: %s, Number of iterations: %d\n', mat2str(x2\_jacobi, 5), iter2\_jacobi);

fprintf('Solution using Gauss-Seidel method: %s, Number of iterations: %d\n', mat2str(x2\_gauss\_seidel, 5), iter2\_gauss\_seidel);

fprintf('For the third system of equations:\n');

fprintf('Solution using Jacobi method: %s, Number of iterations: %d\n', mat2str(x3\_jacobi, 5), iter3\_jacobi);

fprintf('Solution using Gauss-Seidel method: %s, Number of iterations: %d\n', mat2str(x3\_gauss\_seidel, 5), iter3\_gauss\_seidel);

# Problem 1 Question (b)

For the first system of equations, the Gauss-Seidel method converges faster (requiring 12 iterations) compared to the Jacobi method (requiring 16 iterations).

# Problem 1 Question (b)

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **System** |  |  | **Jacobi Solution** | **Jacobi Iterations** | **Gauss-Seidel Solution** | **Gauss-Seidel Iterations** |
| 1 | 0.486 | 0.354 |  | 16 |  | 12 |
| 2 | 1.6 | 0.716 | Does not converge | 1000 |  | 34 |
| 3 | 4.583 | 21 | Does not converge | 1000 | Does not converge | 1000 |

# Problem 2 Question (a)

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  |  |  |  |  |
| 1 | -1.000000 | 0.000000 | -0.500000 |  |
| 2 | -1.000000 | -0.500000 | -0.750000 |  |
| 3 | -0.750000 | -0.500000 | -0.625000 |  |
| 4 | -0.625000 | -0.500000 | -0.562500 |  |
| 5 | -0.625000 | -0.562500 | -0.593750 |  |
| 6 | -0.593750 | -0.562500 | -0.578125 |  |
| 7 | -0.578125 | -0.562500 | -0.570312 |  |
| 8 | -0.570312 | -0.562500 | -0.566406 |  |
| 9 | -0.570312 | -0.566406 | -0.568359 |  |
| 10 | -0.568359 | -0.566406 | -0.567383 |  |
| 11 | -0.567383 | -0.566406 | -0.566895 |  |
| 12 | -0.567383 | -0.566895 | -0.567139 |  |
| 13 | -0.567383 | -0.567139 | -0.567261 |  |
| 14 | -0.567261 | -0.567139 | -0.567200 |  |
| 15 | -0.567200 | -0.567139 | -0.567169 |  |
| 16 | -0.567169 | -0.567139 | -0.567154 |  |
| 17 | -0.567154 | -0.567139 | -0.567146 |  |
| 18 | -0.567146 | -0.567139 | -0.567142 |  |
| 19 | -0.567146 | -0.567142 | -0.567144 |  |
| 20 | -0.567144 | -0.567142 | -0.567143 |  |
| 21 | -0.567143 | -0.567142 | -0.567143 |  |
| 22 | -0.567143 | -0.567143 | -0.567143 |  |

% Define the function f(x) = x + exp(x)

f = @(x) x + exp(x);

% Initialize parameters

a = -1;

b = 0;

N\_0 = 22;

i = 1;

FA = f(a);

% Create an empty array to store the results

results = zeros(N\_0, 5);

% Start the bisection algorithm

while i <= N\_0

% Compute p

p = (a + b) / 2;

% Compute f(p)

FP = f(p);

% Store the results in the array

results(i, :) = [i, a, b, p, FP];

% Update i

i = i + 1;

% Update a, b, and FA accordingly

if FA \* FP > 0

a = p;

FA = FP;

else

b = p;

end

end

% Convert the results to a table and display them

resultsTable = array2table(results, 'VariableNames', {'n', 'a\_n', 'b\_n', 'c\_n', 'f\_c\_n'});

disp(resultsTable)

# Problem 2 Question (b)

|  |  |  |  |
| --- | --- | --- | --- |
|  |  |  |  |
| **0** | -1 | **11** | -0.566415 |
| **1** | -0.367879 | **12** | -0.567557 |
| **2** | -0.692201 | **13** | -0.566909 |
| **3** | -0.500474 | **14** | -0.567276 |
| **4** | -0.606244 | **15** | -0.567068 |
| **5** | -0.545396 | **16** | -0.567186 |
| **6** | -0.579612 | **17** | -0.567119 |
| **7** | -0.560115 | **18** | -0.567157 |
| **8** | -0.571143 | **19** | -0.567135 |
| **9** | -0.564879 | **20** | -0.567148 |
| **10** | -0.568429 | **21** | -0.567141 |

% Fixed-Point Iteration to find the root of x + e^x = 0

clear; clc;

% Function definition for g(x) = -e^x

g = @(x) -exp(x);

% Initialization

x\_0 = -1;

tolerance = 1e-6;

max\_iterations = 100;

x\_n = x\_0;

% Display header

fprintf('n\tx\_n\n');

fprintf('----------------------\n');

fprintf('0\t%.6f\n', x\_0);

% Fixed-Point Iteration Algorithm

for n = 1:max\_iterations

% Compute x\_{n+1}

x\_next = g(x\_n);

% Display the result

fprintf('%d\t%.6f\n', n, x\_next);

% Check the stopping criteria

if abs(x\_next - x\_n) < tolerance

break;

end

% Update x\_n for the next iteration

x\_n = x\_next;

end

# Problem 2 Question (c)

|  |  |  |  |
| --- | --- | --- | --- |
| **Iteration** |  |  |  |
| 1 | -1.000000 | -0.537883 |  |
| 2 | -0.537883 | -0.566987 |  |
| 3 | -0.566987 | -0.567143 |  |
| 4 | -0.567143 | -0.567143 |  |

% Initialize parameters

clear; clc;

p0 = -1;

epsilon = 1e-6;

N0 = 22;

i = 1;

% Display headers for the table

fprintf('Iteration \t p0 \t\t p \t\t |p - p0|\n');

% Start Newton's method

while i <= N0

% Compute f(p0) and f'(p0)

fp0 = p0 + exp(p0);

f\_prime\_p0 = 1 + exp(p0);

% Compute new p

p = p0 - fp0 / f\_prime\_p0;

% Display the results

fprintf('%d \t\t %.6f \t %.6f \t %.6e\n', i, p0, p, abs(p - p0));

% Check the stopping criteria

if abs(p - p0) < epsilon

fprintf('The method found a root after %d iterations: p = %.6f\n', i, p);

break;

end

% Update i and p0

i = i + 1;

p0 = p;

end

% If the method didn't find a root, print a failure message

if i > N0

fprintf('The method failed after %d iterations.\n', N0);

end