实验报告

代卓远

2025210205，机械工程系，机研 52

（2026 年 1 月 4 日）

1 题目

实验目的：病态线性代数方程组的求解

问题：设 $H_n = [h_ij] \in R^{n\times n}$ 是 n 阶 Hilbert 矩阵，即

$$h_{ij} = \frac{1}{i+j-1},$$

$x = (1, 1, \cdots, 1)^T \in R^n$, $b_n = H_n x$.。

(1) 对 $n = 2, 3, 4, \cdots$, 计算条件数 cond$(H_n)_1$, 分析 cond$(H_n)_1$ 作为 n 的函数如何变化（用图表示）;

(2) 对 $n = 6$, 用 LU 分解方法、Jacobi 迭代法、SOR 迭代法（取 $\omega = 1, 1.25, 1.5$ 等）以及共轭梯度法求解 $H_n x = b_n$, 计算几种方法的误差，选用适当的范数，列表表示结果，结论如何;

(3) 逐步增大 n, 用 (2) 中的方法求解，观察误差的变化（n 多大时绝对误差达到 100%，即连一位有效数字都没有了）（选用适当的范数，用图和表表示）。

2 算法设计

2.1 条件数计算（第一问）

对于 n 阶 Hilbert 矩阵 $H_n = [h_{ij}] \in R^{n\times n}$（其中 $h_{ij} = \frac{1}{i+j-1}$），其 1-范数条件数定义为：

$$\text{cond}(H_n)_1 = \|H_n\|_1 \cdot \|H_n^{-1}\|_1 \tag{1}$$

其中，矩阵的 1-范数（列范数）计算公式为：

$$\|A\|_1 = \max_j \sum_i |a_{ij}| \tag{2}$$

由于 H_n 是对称矩阵，其 1-范数与 ∞-范数（行范数）数值相等。计算 $n = 2, 3, \ldots$ 时的条件数并作图分析。

2.2 方程组求解算法（第二问）

针对方程组 $H_n x = b_n$，采用以下方法求解：

2.2.1 LU 分解法

若 A 的各阶顺序主子式均非零，存在唯一的单位下三角矩阵 L 和上三角矩阵 U 使得 $A = LU$。分解公式如下：

$$u_{ij} = a_{ij} - \sum_{k=1}^{i-1} l_{ik} u_{kj}, \quad (j = i, \ldots, n) \tag{3}$$

$$l_{ji} = \frac{1}{u_{ii}} \left(a_{ji} - \sum_{k=1}^{i-1} l_{jk} u_{ki} \right), \quad (j = i+1, \ldots, n) \tag{4}$$

求解 $Ax = b$ 转化为求解 $Ly = b$ 和 $Ux = y$。

2.2.2 Jacobi 迭代法

迭代公式为：

$$x^{(k+1)} = D^{-1}(b + Lx^{(k)} + Ux^{(k)}) \tag{5}$$

2.2.3 SOR 迭代法

选取松弛因子 ω，迭代公式为：

$$x^{(k+1)} = (1-\omega)x^{(k)} + \omega D^{-1}(b + Lx^{(k+1)} + Ux^{(k)}) \tag{6}$$

收敛的必要条件是 $\omega \in (0, 2)$。

2.2.4 共轭梯度法

适用于对称正定矩阵。取初值 $x^{(0)}$，令 $p^{(0)} = r^{(0)} = b - Ax^{(0)}$。

第 k 步迭代步骤如下：

$$\begin{aligned}
\alpha_k &= \frac{(r^{(k)}, r^{(k)})}{(Ap^{(k)}, p^{(k)})} \\
x^{(k+1)} &= x^{(k)} + \alpha_k p^{(k)} \\
r^{(k+1)} &= r^{(k)} - \alpha_k A p^{(k)} \\
\beta_k &= \frac{(r^{(k+1)}, r^{(k+1)})}{(r^{(k)}, r^{(k)})} \\
p^{(k+1)} &= r^{(k+1)} + \beta_k p^{(k)}
\end{aligned} \tag{7}$$

2.3 误差分析（第三问）

定义数值解 $\tilde{x}$ 的相对误差为 $\frac{\|\tilde{x}-x^*\|}{\|x^*\|}$。根据误差分析定理，当系数矩阵存在扰动时，误差界与条件数 $\text{cond}(A)$ 成正比。对于 Hilbert 矩阵，由于 $\text{cond}(H_n)$ 极大，预计当 n 增大到一定程度（如 $n \geq 12$），计算结果将完全失效。

3 求解

3.1 程序（注释）

```
function experiment()
    % 病态Hilbert矩阵实验
    clc; clear; close all;

    %% 1. 条件数分析
    fprintf('1. 条件数分析\n');
    n_cond_range = 2:16;
    conds = zeros(size(n_cond_range));

    for i = 1:length(n_cond_range)
        n = n_cond_range(i);
        H = hilb(n);
        % 1-范数条件数
        conds(i) = cond(H, 1);
    end

    % 绘制条件数变化图
    figure(1);
    semilogy(n_cond_range, conds, '-bo', 'LineWidth', 1.5);
    title('Hilbert矩阵条件数cond(H)_1随n的变化');
    xlabel('矩阵阶数n');
    ylabel('条件数（对数坐标）');
    grid on;

    %% 2. n=6时的求解与误差计算
    fprintf('2. n=6时的求解误差\n');
    n = 6;
    [H, b, x_true] = setup_problem(n);

    % (1) LU分解法
    x_lu = solve_lu_manual(H, b);
    err_lu = norm(x_lu - x_true, inf) / norm(x_true, inf);

    % (2) Jacobi迭代法
```

```
[x_jac, iter_jac] = solve_jacobi(H, b);
err_jac = norm(x_jac - x_true, inf) / norm(x_true, inf);

% (3) SOR迭代法
omegas = [1.0, 1.25, 1.5];
err_sor = zeros(length(omegas), 1);
iters_sor = zeros(length(omegas), 1);
for i = 1:length(omegas)
    [x_s, k_s] = solve_sor(H, b, omegas(i));
    err_sor(i) = norm(x_s - x_true, inf) / norm(x_true, inf);
    iters_sor(i) = k_s;
end

% (4) 共轭梯度法
[x_cg, iter_cg] = solve_cg(H, b);
err_cg = norm(x_cg - x_true, inf) / norm(x_true, inf);

% 输出结果表格
fprintf('%-15s %-10s %-15s\n', '方法', '迭代次数', '相对误差');
fprintf('%-15s %-10s %-.4e\n', 'LU分解', '-', err_lu);
fprintf('%-15s %-10d %-.4e\n', 'Jacobi', iter_jac, err_jac);
fprintf('%-15s %-10d %-.4e\n', 'SOR(w=1.0)', iters_sor(1), err_sor(1));
fprintf('%-15s %-10d %-.4e\n', 'SOR(w=1.25)', iters_sor(2), err_sor(2));
fprintf('%-15s %-10d %-.4e\n', 'SOR(w=1.5)', iters_sor(3), err_sor(3));
fprintf('%-15s %-10d %-.4e\n', 'CG', iter_cg, err_cg);
fprintf('\n');

%% 3. 误差随n的变化
fprintf('3. 误差随n的变化');
n_max = 100;
err_history = zeros(n_max, 4);
for k = 2:n_max
    [Hk, bk, xk_true] = setup_problem(k);

    % LU
    xk_lu = solve_lu_manual(Hk, bk);
    err_history(k, 1) = norm(xk_lu - xk_true, inf) / norm(xk_true, inf);

    % Jacobi
```

```
            [xk_jac, ~] = solve_jacobi(Hk, bk);
            err_history(k, 2) = norm(xk_jac - xk_true, inf) / norm(
                xk_true, inf);

            % SOR (w=1.25)
            [xk_sor, ~] = solve_sor(Hk, bk, 1.25);
            err_history(k, 3) = norm(xk_sor - xk_true, inf) / norm(
                xk_true, inf);

            % CG
            [xk_cg, ~] = solve_cg(Hk, bk);
            err_history(k, 4) = norm(xk_cg - xk_true, inf) / norm(
                xk_true, inf);

            % 检查误差是否达到100%
            methods = {'LU', 'Jacobi', 'SOR(1.25)', 'CG'};
            for m = 1:4
                if err_history(k, m) >= 1.0 && err_history(k-1, m) <
                    1.0
                    fprintf('当n = %d时, %s方法的误差达到100%%。\n',
                        ...
                        k, methods{m});
                end
            end
        end

        % 绘制误差曲线
        figure(2);
        n_axis = 2:n_max;
        semilogy(n_axis, err_history(2:end, 1), '-o', 'DisplayName',
            'LU分解'); hold on;
        semilogy(n_axis, err_history(2:end, 2), '-s', 'DisplayName',
            'Jacobi');
        semilogy(n_axis, err_history(2:end, 3), '-^', 'DisplayName',
            'SOR(w=1.25)');
        semilogy(n_axis, err_history(2:end, 4), '-x', 'DisplayName',
            'CG');

        yline(1.0, '--r', '误差 100%', 'LineWidth', 2);
        title('各方法相对误差随n的变化');
        xlabel('矩阵阶数n'); ylabel('相对误差 (对数坐标) ');
        legend('Location', 'southeast');
        grid on;
    end
```

```
109
110 %% 辅助函数定义
111 function [H, b, x_true] = setup_problem(n)
112     H = hilb(n);
113     x_true = ones(n, 1);
114     b = H * x_true;
115 end
116
117 % 1. LU分解
118 function x = solve_lu_manual(A, b)
119     n = length(b);
120     L = eye(n); U = zeros(n);
121
122     for i = 1:n
123         for j = i:n
124             U(i,j) = A(i,j) - L(i,1:i-1)*U(1:i-1,j);
125         end
126         for j = i+1:n
127             L(j,i) = (A(j,i) - L(j,1:i-1)*U(1:i-1,i)) / U(i,i);
128         end
129     end
130
131     % 前代解
132     y = zeros(n, 1);
133     for i = 1:n
134         y(i) = b(i) - L(i, 1:i-1) * y(1:i-1);
135     end
136
137     % 回代解
138     x = zeros(n, 1);
139     for i = n:-1:1
140         x(i) = (y(i) - U(i, i+1:n) * x(i+1:n)) / U(i,i);
141     end
142 end
143
144 % 2. Jacobi
145 function [x, k] = solve_jacobi(A, b)
146     n = length(b);
147     x = zeros(n, 1);
148     D = diag(A);
149     R = A - diag(D);
150     max_iter = 5000;
151     tol = 1e-6;
152
```

```
        for k = 1:max_iter
            x_new = (b - R*x) ./ D;
            if norm(x_new - x, inf) < tol
                x = x_new; return;
            end
            x = x_new;
        end
    end

    % 3. SOR
    function [x, k] = solve_sor(A, b, w)
        n = length(b);
        x = zeros(n, 1);
        max_iter = 5000;
        tol = 1e-6;

        for k = 1:max_iter
            x_old = x;
            for i = 1:n
                % sigma = sum(A(i,j) * x(j)) for j != i
                sigma = A(i, :) * x - A(i, i) * x(i);

                % Gauss-Seidel更新值
                x_gs = (b(i) - sigma) / A(i, i);

                % SOR加权更新
                x(i) = (1 - w) * x(i) + w * x_gs;
            end

            if norm(x - x_old, inf) < tol
                return;
            end
        end
    end

    % 4. 共轭梯度法
    function [x, k] = solve_cg(A, b)
        n = length(b);
        x = zeros(n, 1);
        r = b - A*x;
        p = r;
        max_iter = 200;
        tol = 1e-6;

```

```
        for k = 1:max_iter
            if norm(r) < tol, return; end

            Ap = A * p;
            alpha = (r' * r) / (p' * Ap);
            x = x + alpha * p;
            r_new = r - alpha * Ap;

            beta = (r_new' * r_new) / (r' * r);
            p = r_new + beta * p;

            r = r_new;
        end
    end
```

3.2 结果及分析

1. **条件数增长**：随着 n 增大，$\text{cond}(H_n)_1$ 呈指数级增长。例如 $n = 3$ 时，$\text{cond}(H_3)_1 = 748$ [cite: 949]，而当 $n = 6$ 时已达到 10^7 量级。

2. **求解误差** ($n = 6$)：

   - LU 分解法保持了较高的精度（误差 $\sim 10^{-12}$），但在病态严重时受舍入误差影响较大 [cite: 967]。
   - Jacobi 迭代法在 $n = 6$ 时收敛极慢甚至发散，因为 H_n 虽正定但不严格对角占优 [cite: 618-620]。
   - CG 方法对正定矩阵有效，但受条件数影响，收敛步数增加。

3. **失效点**：当 $n \approx 13$ 时，条件数接近 10^{18}，超过了双精度浮点数的机器精度，导致绝对误差达到 100%，解完全不可信。

4 总结（可以有收获与建议）

- **病态性理解**：通过 Hilbert 矩阵实验，深刻理解了条件数 $cond(A)$ 对数值稳定性的决定性作用。当 $cond(A)$ 很大时，微小的扰动（如计算机舍入误差）会被显著放大 [cite: 922]。
- **算法选择**：对于病态方程组，直接法（如 LU）通常比简单的迭代法更可靠，但仍需结合预处理技术或使用高精度计算 [cite: 953]。
- **课件关联**：实验验证了课件中关于“病态方程组求解须小心进行”的论述 [cite: 952]，以及 LU 分解的存在性定理 [cite: 768]。

Listing 1　Matlab Code Example (示例代码)

```
function main()
    % 中文注释：微软雅黑
    % English Comment: Consolas
    disp('Hello World');
end
```

4.1　Results (结果)

如图 1 所示（假设有图），题注中的英文也已应用新罗马字体。

图 1　Error Analysis Plot (误差分析图)