

# 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

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

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

(1) 对  $n = 2, 3, 4, \dots$ , 计算条件数  $\text{cond}(H_n)_1$ , 分析  $\text{cond}(H_n)_1$  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:

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

其中, 矩阵的 1-范数 (列范数) 计算公式为:

$$\|A\|_1 = \max_j \sum_i |a_{ij}| \quad (2)$$

由于  $H_n$  是对称矩阵, 其 1-范数与  $\infty$ -范数 (行范数) 数值相等。计算  $n = 2, 3, \dots$  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：

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

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

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：

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

### 2.2.3 SOR 迭代法

选取松弛因子  $\omega$ ，迭代公式为：

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

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：

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

## 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）

```
1  function experiment()
2      % 病态Hilbert矩阵实验
3      clc; clear; close all;
4
5      %% 1. 条件数分析
6      fprintf('1. 条件数分析\n');
7      n_cond_range = 2:16;
8      conds = zeros(size(n_cond_range));
9
10     for i = 1:length(n_cond_range)
11         n = n_cond_range(i);
12         H = hilb(n);
13         % 1-范数条件数
14         conds(i) = cond(H, 1);
15     end
16
17     % 绘制条件数变化图
18     figure(1);
19     semilogy(n_cond_range, conds, '-bo', 'LineWidth', 1.5);
20     title('Hilbert矩阵条件数cond(H)_1随n的变化');
21     xlabel('矩阵阶数n');
22     ylabel('条件数（对数坐标）');
23     grid on;
24
25     %% 2. n=6时的求解与误差计算
26     fprintf('2. n=6时的求解误差\n');
27     n = 6;
28     [H, b, x_true] = setup_problem(n);
29
30     % (1) LU分解法
31     x_lu = solve_lu_manual(H, b);
32     err_lu = norm(x_lu - x_true, inf) / norm(x_true, inf);
33
34     % (2) Jacobi迭代法
```

```

35 [x_jac, iter_jac] = solve_jacobi(H, b);
36 err_jac = norm(x_jac - x_true, inf) / norm(x_true, inf);
37
38 % (3) SOR迭代法
39 omegas = [1.0, 1.25, 1.5];
40 err_sor = zeros(length(omegas), 1);
41 iters_sor = zeros(length(omegas), 1);
42 for i = 1:length(omegas)
43     [x_s, k_s] = solve_sor(H, b, omegas(i));
44     err_sor(i) = norm(x_s - x_true, inf) / norm(x_true, inf);
45     iters_sor(i) = k_s;
46 end
47
48 % (4) 共轭梯度法
49 [x_cg, iter_cg] = solve_cg(H, b);
50 err_cg = norm(x_cg - x_true, inf) / norm(x_true, inf);
51
52 % 输出结果表格
53 fprintf('%-15s %-10s %-15s\n', '方法', '迭代次数', '相对误差'
54 );
55 fprintf('%-15s %-10s %-.4e\n', 'LU分解', '-', err_lu);
56 fprintf('%-15s %-10d %-.4e\n', 'Jacobi', iter_jac, err_jac);
57 fprintf('%-15s %-10d %-.4e\n', 'SOR(w=1.0)', iters_sor(1),
58 err_sor(1));
59 fprintf('%-15s %-10d %-.4e\n', 'SOR(w=1.25)', iters_sor(2),
60 err_sor(2));
61 fprintf('%-15s %-10d %-.4e\n', 'SOR(w=1.5)', iters_sor(3),
62 err_sor(3));
63 fprintf('%-15s %-10d %-.4e\n', 'CG', iter_cg, err_cg);
64 fprintf('\n');
65
66 %% 3. 误差随n的变化
67 fprintf('3. 误差随n的变化');
68 n_max = 100;
69 err_history = zeros(n_max, 4);
70 for k = 2:n_max
71     [Hk, bk, xk_true] = setup_problem(k);
72
73     % LU
74     xk_lu = solve_lu_manual(Hk, bk);
75     err_history(k, 1) = norm(xk_lu - xk_true, inf) / norm(
76         xk_true, inf);
77
78     % Jacobi

```

```

74     [xk_jac, ~] = solve_jacobi(Hk, bk);
75     err_history(k, 2) = norm(xk_jac - xk_true, inf) / norm(
        xk_true, inf);
76
77     % SOR (w=1.25)
78     [xk_sor, ~] = solve_sor(Hk, bk, 1.25);
79     err_history(k, 3) = norm(xk_sor - xk_true, inf) / norm(
        xk_true, inf);
80
81     % CG
82     [xk_cg, ~] = solve_cg(Hk, bk);
83     err_history(k, 4) = norm(xk_cg - xk_true, inf) / norm(
        xk_true, inf);
84
85     % 检查误差是否达到100%
86     methods = {'LU', 'Jacobi', 'SOR(1.25)', 'CG'};
87     for m = 1:4
88         if err_history(k, m) >= 1.0 && err_history(k-1, m) <
            1.0
89             fprintf('当n = %d时, %s方法的误差达到100%%。\\n',
                ...
                k, methods{m});
90         end
91     end
92 end
93 end
94
95 % 绘制误差曲线
96 figure(2);
97 n_axis = 2:n_max;
98 semilogy(n_axis, err_history(2:end, 1), '-o', 'DisplayName',
    'LU分解'); hold on;
99 semilogy(n_axis, err_history(2:end, 2), '-s', 'DisplayName',
    'Jacobi');
100 semilogy(n_axis, err_history(2:end, 3), '-^', 'DisplayName',
    'SOR(w=1.25)');
101 semilogy(n_axis, err_history(2:end, 4), '-x', 'DisplayName',
    'CG');
102
103 yline(1.0, '--r', '误差 100%', 'LineWidth', 2);
104 title('各方法相对误差随n的变化');
105 xlabel('矩阵阶数n'); ylabel('相对误差 (对数坐标)');
106 legend('Location', 'southeast');
107 grid on;
108 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

```

```

153     for k = 1:max_iter
154         x_new = (b - R*x) ./ D;
155         if norm(x_new - x, inf) < tol
156             x = x_new; return;
157         end
158         x = x_new;
159     end
160 end
161
162 % 3. SOR
163 function [x, k] = solve_sor(A, b, w)
164     n = length(b);
165     x = zeros(n, 1);
166     max_iter = 5000;
167     tol = 1e-6;
168
169     for k = 1:max_iter
170         x_old = x;
171         for i = 1:n
172             % sigma = sum(A(i,j) * x(j)) for j != i
173             sigma = A(i, :) * x - A(i, i) * x(i);
174
175             % Gauss-Seidel更新值
176             x_gs = (b(i) - sigma) / A(i, i);
177
178             % SOR加权更新
179             x(i) = (1 - w) * x(i) + w * x_gs;
180         end
181
182         if norm(x - x_old, inf) < tol
183             return;
184         end
185     end
186 end
187
188 % 4. 共轭梯度法
189 function [x, k] = solve_cg(A, b)
190     n = length(b);
191     x = zeros(n, 1);
192     r = b - A*x;
193     p = r;
194     max_iter = 200;
195     tol = 1e-6;
196

```

```

197     for k = 1:max_iter
198         if norm(r) < tol, return; end
199
200         Ap = A * p;
201         alpha = (r' * r) / (p' * Ap);
202         x = x + alpha * p;
203         r_new = r - alpha * Ap;
204
205         beta = (r_new' * r_new) / (r' * r);
206         p = r_new + beta * p;
207
208         r = r_new;
209     end
210 end

```

## 3.2 结果及分析

### 3.2.1 条件数的变化趋势

如图 1 所示，计算矩阵阶数  $n$  从 2 到 16 时的 1-范数条件数  $\text{cond}(H_n)_1$ 。

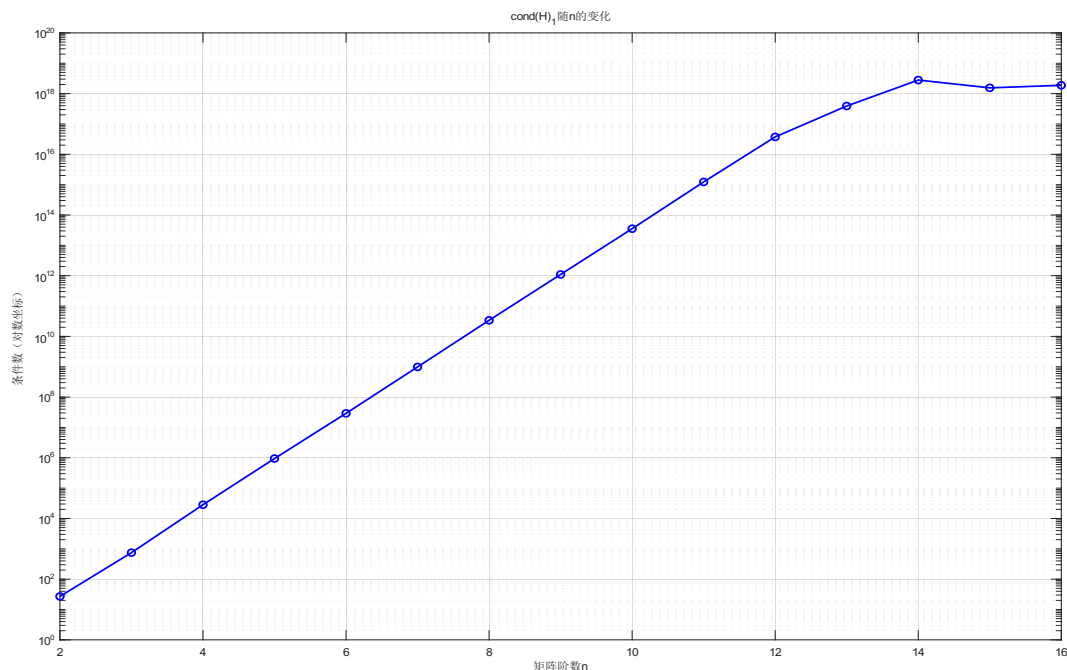


图 1 Hilbert 矩阵条件数随阶数  $n$  的变化 (半对数坐标)

在单对数坐标系下，条件数曲线呈现明显的线性增长趋势，说明 Hilbert 矩阵的条件数随阶数  $n$  呈指数级增长。

当  $n = 13$  左右时，条件数达到了  $10^{18}$  量级。此时矩阵的数值性质已极度不稳定，计算出的条件数出现波动，不再严格单调。

### 3.2.2 $n=6$ 时的求解误差对比

当  $n = 6$  时，各方法的求解结果及误差如下表所示：

表 1 不同方法求解  $H_6x = b_6$  的误差对比 ( $n = 6$ )

方法	迭代次数	相对误差	状态分析
LU 分解	-	$2.2057 \times 10^{-10}$	有效，但精度受限
Jacobi	5000	NaN	发散
SOR ( $\omega = 1.0$ )	5000	$2.5376 \times 10^{-2}$	收敛停滞
SOR ( $\omega = 1.25$ )	5000	$2.8683 \times 10^{-2}$	效果变差
SOR ( $\omega = 1.5$ )	5000	$3.2851 \times 10^{-2}$	效果更差
CG (共轭梯度)	5	$5.4680 \times 10^{-3}$	假性收敛

Jacobi 发散，结果出现 NaN，Hilbert 矩阵不满足 Jacobi 方法收敛所需的严格对角占优条件，导致迭代过程中数值溢出。

LU 分解精度最高，在  $n = 6$  时，矩阵  $\text{cond} \approx 10^7$  较大，LU 分解仍能保持约 10 位有效数字的精度。

迭代法效果有限，SOR 和共轭梯度法最终误差停留在  $10^{-2}$  到  $10^{-3}$ 。

### 3.2.3 误差随阶数 $n$ 的变化

随着  $n$  增大，各方法相对误差的变化曲线如图 2 所示。

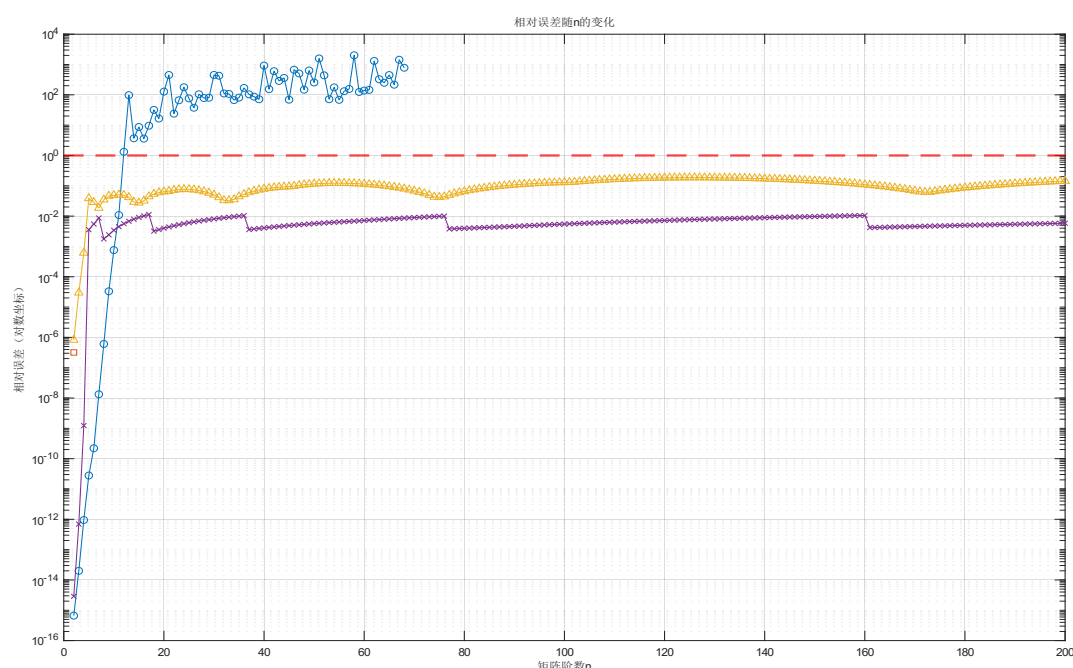


图 2 各方法相对误差随矩阵阶数  $n$  的变化趋势

LU 分解（蓝色）在  $n < 10$  时，误差随  $n$  迅速上升，从  $10^{-15}$  上升至  $10^{-2}$ 。当  $n = 12$  时，相对误差超过 100%，计算结果已完全失去有效数字，数值求解失效。

迭代法（黄色和紫色）曲线始终位于  $10^{-2}$  左右，精度停滞，难以继续提升。

## 4 总结（可以有收获与建议）

1. 通过 Hilbert 矩阵实验，深刻理解了条件数  $\text{cond}(A)$  对数值稳定性的决定性作用。当  $\text{cond}(A)$  很大时，微小的扰动（如计算机舍入误差）会被显著放大。
2. 对于病态方程组，直接法（如 LU）比简单的迭代法更可靠，但需要使用高精度计算。