21374389 牛鹏军 210711

理论分析

2. 用 C 语言实现高斯消去算法，并用 5、10、15 阶 Hilbert 矩阵 H_n 求解 $H_n X = \begin{pmatrix} 1 \\ \vdots \\ 1 \end{pmatrix}$，验证算法的有效性。

对于此题，考虑封装两个函数：其一为生成 Hilbert 矩阵，其二为高斯消元法，高斯消元法分为消元与回代两个步骤。验证算法的有效性环节，我决定用 Matlab 编写 LU 求解，主要原因 Matlab 中函数比较齐全且更适合处理矩阵数据，对比两种编译环境下的输出值，得到结论。

算法设计

严格按照 Hilbert 矩阵定义以及高斯消元法定义来就行，具体见编程实现部分封装的两个函数。

编程实现

C 语言代码：

```
#include <stdio.h>
#include <stdlib.h>

double** generateHilbertMatrix(int n) {
    // 动态分配内存以保存 Hilbert 矩阵
    double** hilbertMatrix = (double**)malloc(n * sizeof(double*));

    for (int i = 0; i < n; i++) {
        // 为每一行分配内存
        hilbertMatrix[i] = (double*)malloc(n * sizeof(double));

        for (int j = 0; j < n; j++) {
            // 通过公式计算 Hilbert 矩阵的每个元素的值
            hilbertMatrix[i][j] = 1.0 / (i + j + 1);
        }
    }

    return hilbertMatrix;
}

void gaussianElimination(double** matrix, double* vector, int n) {
    for (int k = 0; k < n - 1; k++) {
        for (int i = k + 1; i < n; i++) {
```

```
                // 计算消元因子
                double factor = matrix[i][k] / matrix[k][k];

                for (int j = k; j < n; j++) {
                    // 执行消元操作
                    matrix[i][j] -= factor * matrix[k][j];
                }

                // 更新向量
                vector[i] -= factor * vector[k];
            }
        }

        // 解向量的数组
        double* solution = (double*)malloc(n * sizeof(double));

        for (int i = n - 1; i >= 0; i--) {
            double sum = 0.0;

            for (int j = i + 1; j < n; j++) {
                // 计算解向量的部分和
                sum += matrix[i][j] * solution[j];
            }

            // 计算解向量的每个元素的值
            solution[i] = (vector[i] - sum) / matrix[i][i];
        }

        printf("Solution:\n");

        for (int i = 0; i < n; i++) {
            // 打印解向量的每个元素
            printf("x%d = %lf\n", i + 1, solution[i]);
        }

        // 释放解向量的内存
        free(solution);
}

int main() {
        int n;

        printf("请输入 Hilbert 矩阵的大小： ");
```

```
    scanf("%d", &n);

    // 生成 Hilbert 矩阵
    double** hilbertMatrix = generateHilbertMatrix(n);

    // 动态分配内存以保存向量
    double* vector = (double*)malloc(n * sizeof(double));

    // 初始化向量的元素
    for (int i = 0; i < n; i++) {
        vector[i] = 1.0;
    }

    // 调用高斯消元算法解线性方程组
    gaussianElimination(hilbertMatrix, vector, n);

    // 释放内存
    for (int i = 0; i < n; i++) {
        free(hilbertMatrix[i]);
    }
    free(hilbertMatrix);
    free(vector);

    return 0;
}
```

为了便于验证有效性，采用 Matlab 编写 LU 分解程序：

```
n = input('请输入 Hilbert 矩阵的大小：');
hilbertMatrix = hilb(n);
vector = ones(n, 1);

[L, U, P] = lu(hilbertMatrix);
y = P * vector;
solution = U \ (L \ y);

disp('Solution:');
for i = 1:n
disp(['x', num2str(i), ' = ', num2str(solution(i))]);
end
```

测试分析

得到以下输出（左为 C 语言输出，右为 Matlab 输出，依次为 5、10、15 阶）：

请输入Hilbert矩阵的大小: 5 Solution: x1 = 5.000000 x2 = -120.000000 x3 = 630.000000 x4 = -1120.000000 x5 = 630.000000	请输入Hilbert矩阵的大小：5 Solution: x1 = 5 x2 = -120 x3 = 630 x4 = -1120 x5 = 630
请输入Hilbert矩阵的大小: 10 Solution: x1 = -9.997645 x2 = 989.797120 x3 = -23755.688353 x4 = 240200.872382 x5 = -1261073.641735 x6 = 3783268.334666 x7 = -6725881.404654 x8 = 7000470.344871 x9 = -3937795.284733 x10 = 923686.666474	请输入Hilbert矩阵的大小：10 Solution: x1 = -9.9981 x2 = 989.8337 x3 = -23756.462 x4 = 240207.8631 x5 = -1261106.8229 x6 = 3783359.1775 x7 = -6726029.9353 x8 = 7000613.4567 x9 = -3937870.2236 x10 = 923703.1094
请输入Hilbert矩阵的大小: 15 Solution: x1 = 9.936520 x2 = -1441.536379 x3 = 50540.555842 x4 = -743443.644592 x5 = 5629285.756843 x6 = -23839955.216380 x7 = 56529931.781714 x8 = -66300636.023581 x9 = 22288817.347147 x10 = -40304905.513102 x11 = 285485432.739901 x12 = -593691042.678635 x13 = 589262482.573590 x14 = -293895016.678170 x15 = 59530127.653554	请输入Hilbert矩阵的大小：15 Solution: x1 = 14.0115 x2 = -2025.3725 x3 = 71133.4363 x4 = -1052639.9064 x5 = 8034412.609 x6 = -34097248.0767 x7 = 77573838.1062 x8 = -59312669.8192 x9 = -141266240.1972 x10 = 432410872.6603 x11 = -463501279.2447 x12 = 142885684.4169 x13 = 140364301.7942 x14 = -137971560.8564 x15 = 35863593.4837

结论

高斯消去算法在 C 语言的实现中，对于低阶 Hilbert 矩阵的情况比较适用，此时误差比较小。高阶的场景下，由于连环的消元与回代，在双精度浮点数存储数据中下有精度损失。