**21374389 牛鹏军 210711**

**理论分析**

|  |
| --- |
| **32a1dee7986ee28f9ab162dc822e924** |

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）：

|  |  |
| --- | --- |
|  |  |
|  |  |
|  |  |

**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。