

理 学 院 数 学 系

实 验 报 告

|  |  |
| --- | --- |
| **课程名称：** | 数值分析 |
| **实验学期：** | 2018-2019第二学期 |
| **实验班级：** | 软工1708 |
| **学生学号：** | 20174996 |
| **学生姓名：** | 汤元丰 |
| **指导教师：** | 盛莹 |

**理学院数学实验中心**

二〇一九年 五月 三 日

**理学院数学实验中心实验报告**

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 专业： | 软件工程 | | 姓名： | | 汤元丰 | | 学号： | 20174996 | 成绩： |  |
| 实验地点： | | 寝室 | |  | |  | | 指导教师签名： | |  |
|  | |  | |  | |  | |  | |  |
| 1. **实验名称**：   **研究Hilbert矩阵的病态问题**  二、**实验目的：**  1．实践通过编程解决数值分析问题。  2．熟悉Gauss消去法、Jacobi迭代法、Gauss-Seidel迭代法和SOR迭代法的算法原理。  3．了解Hilbert矩阵的各种性质。  4．能够通过编程实现矩阵求逆和矩阵各种运算。  5．掌握如何分析病态问题的求解方法。  6．了解如何对病态系数矩阵进行预处理。  **三、实验原理：**  **1.Gauss消去法：**  **原理：**通过逐次消元计算把需求解的线性方程组系数矩阵转化成上三角矩阵，从而使一般线性方程组的求解转化成等价的上三角形方程组的求解。  **实现方式：**   1. 将传入的A，b矩阵进行复制。 2. 从左到右遍历每一列，对每一列进行消元处理，并将消元结果同步作用于A，b矩阵的被消元行中。 3. 消元成上三角矩阵后，对x矩阵从下往上进行求解。 4. 返回求解后的x矩阵。 5. **Jacobi迭代法：**   **原理：**设Ax = b，其中A=D-L-U；Jacobi迭代算法：  ;  **实现方法：**   1. 通过编写的矩阵类实现Jacobi迭代算法公式。 2. 编写迭代器，将Jacobi类对象传入进行迭代。   **3．SOR迭代法：**  **原理：**设Ax = b，其中A=D-L-U；SOR迭代算法：  **实现方法：**   1. 通过编写的矩阵类实现SOR迭代算法公式。 2. 将SOR类对象传入迭代器进行迭代。 3. **Gauss-Seidel迭代法：**   **原理：**设Ax = b，其中A=D-L-U；Gauss-Seidel迭代算法：  **实现方法：**   1. 通过继承SOR迭代法类，将参数设置为1。 2. 将GS迭代法类对象传入迭代器进行迭代。   **四、实验数据记录及处理：**  使用Gauss消去法、Jacobi迭代法、Gauss-Seidel迭代法和SOR迭代法q求解线性方程组。指n维Hilbert矩阵。   * 1. 针对问题（1）给出每个方法对6维Hilbert系数矩阵方程组的解；   注：最大迭代次数设置为10000000，精度要求为小数点后两位，SOR迭代法参数设置为。  表4-1 当n=6时的各方法对Hilbert系数矩阵方程组的求解结果   |  |  |  |  |  | | --- | --- | --- | --- | --- | | 迭代次数 | 1 | 487 | 14507 | 168298 | | 精确解 | **Gauss消去** | **Jacobi迭代** | **GS迭代** | **SOR迭代** | | 1 | 0.99999 | NaN | 0.99988 | 1.00002 | | 1 | 1.00000 | NaN | 1.00161 | 0.99944 | | 1 | 0.99999 | NaN | 0.99574 | 1.00361 | | 1 | 1.00000 | NaN | 0.99936 | 0.99079 | | 1 | 0.99999 | NaN | 1.00999 | 1.00999 | | 1 | 1.00000 | NaN | 0.99329 | 0.99611 |  * 1. 针对问题（2）给出每个方法对逐渐增大维数的Hilbert系数矩阵方程组的解；   注：求解各参数同上。  表4-2 GS消去法对Hilbert系数矩阵方程组的求解结果   |  |  |  |  |  | | --- | --- | --- | --- | --- | | Hilbert阶数 | N = 5 | N = 10 | N = 15 | N = 20 | | 精确解 | 近似解 | | | | | 1 | 0.99999 | 0.99999 | 0.99999 | 0.99999 | | 1 | 1.00000 | 1.00000 | 1.00000 | 1.00007 | | 1 | 0.99999 | 0.99999 | 0.99999 | 0.99723 | | 1 | 1.00000 | 1.00002 | 0.99864 | 1.04300 | | 1 | 0.99999 | 0.99990 | 1.02566 | 0.65746 | | 1 |  | 1.00026 | 0.78193 | 2.48552 | | 1 |  | 0.99956 | 2.06684 | -2.21802 | | 1 |  | 1.00042 | -2.27903 | 2.01269 | | 1 |  | 0.99977 | 7.53239 | 11.91402 | | 1 |  | 1.00004 | -7.35504 | -20.72883 | | 1 |  |  | 7.38066 | 5.73221 | | 1 |  |  | -1.12904 | 34.23327 | | 1 |  |  | 0.42574 | -42.15362 | | 1 |  |  | 1.73328 | 18.72773 | | 1 |  |  | 0.81795 | 2.72154 | | 1 |  |  |  | -16.80337 | | 1 |  |  |  | 49.40574 | | 1 |  |  |  | -57.51627 | | 1 |  |  |  | 33.28834 | | 1 |  |  |  | -5.79874 |   表4-3 Jacobi迭代法对Hilbert系数矩阵方程组的求解结果   |  |  |  |  | | --- | --- | --- | --- | | Hilbert阶数 | N = 1 | N = 2 | N = 3 | | 迭代次数 | 0 | 33 | 1305 | | 精确解 | 近似解 | | | | 1 | 1.0 | 0.99248 | NaN | | 1 |  | 0.99248 | NaN | | 1 |  |  | NaN |   表4-4 GS迭代法对Hilbert系数矩阵方程组的求解结果   |  |  |  |  |  | | --- | --- | --- | --- | --- | | Hilbert阶数 | N = 5 | N = 10 | N = 15 | N = 20 | | 迭代次数 | 28221 | 25507 | 47390 | 144586 | | 精确解 | 近似解 | | | | | 1 | 0.99991 | 1.00012 | 0.99989 | 1.00003 | | 1 | 1.00156 | 0.99796 | 1.00218 | 0.99896 | | 1 | 0.99332 | 1.00648 | 0.99035 | 1.00593 | | 1 | 1.00999 | 0.99846 | 1.00999 | 0.99000 | | 1 | 0.99513 | 0.99116 | 1.00668 | 0.99969 | | 1 |  | 0.99702 | 0.99709 | 1.00602 | | 1 |  | 1.00570 | 0.99207 | 1.00527 | | 1 |  | 1.00907 | 0.99298 | 1.00128 | | 1 |  | 1.00388 | 0.99745 | 0.99748 | | 1 |  | 0.99000 | 1.00266 | 0.99544 | | 1 |  |  | 1.00644 | 0.99542 | | 1 |  |  | 1.00748 | 0.99693 | | 1 |  |  | 1.00515 | 0.99924 | | 1 |  |  | 0.99931 | 1.00164 | | 1 |  |  | 0.99011 | 1.00352 | | 1 |  |  |  | 1.00441 | | 1 |  |  |  | 1.00401 | | 1 |  |  |  | 1.00214 | | 1 |  |  |  | 0.99871 | | 1 |  |  |  | 0.99374 |   表4-5 SOR迭代法对Hilbert系数矩阵方程组的求解结果   |  |  |  |  |  | | --- | --- | --- | --- | --- | | Hilbert阶数 | N = 5 | N = 10 | N = 15 | N = 20 | | 迭代次数 | 19246 | 223764 | 55450 | 213961 | | 精确解 | 近似解 | | | | | 1 | 0.99990 | 0.99997 | 0.99990 | 1.00003 | | 1 | 1.00162 | 1.00059 | 1.00201 | 0.99909 | | 1 | 0.99322 | 0.99591 | 0.99100 | 1.00538 | | 1 | 1.00999 | 1.00999 | 1.00999 | 0.99000 | | 1 | 0.99519 | 0.99247 | 1.00458 | 1.00260 | | 1 |  | 0.99796 | 0.99796 | 1.00319 | | 1 |  | 0.99966 | 0.99304 | 1.00438 | | 1 |  | 1.00435 | 0.99399 | 1.00075 | | 1 |  | 1.00271 | 0.99760 | 0.99832 | | 1 |  | 0.99632 | 1.00222 | 0.99647 | | 1 |  |  | 1.00559 | 0.99641 | | 1 |  |  | 1.00665 | 0.99744 | | 1 |  |  | 1.00467 | 0.99925 | | 1 |  |  | 0.99949 | 1.00118 | | 1 |  |  | 0.99117 | 1.00227 | | 1 |  |  |  | 1.00359 | | 1 |  |  |  | 1.00335 | | 1 |  |  |  | 1.00186 | | 1 |  |  |  | 0.99902 | | 1 |  |  |  | 0.99479 |   **五、实验内容及步骤：**  **实验内容：**  1.编写程序对将Hilbert矩阵作为系数矩阵的方程组进行求解。  2.使用Gauss消去法、Jacobi迭代法、Gauss-Seidel迭代法和SOR迭代法对六维的Hilbert系数矩阵方程组进行求解，并比较每个方法的解之间的不同，总结出结论。  3.逐步增大Hilbert矩阵维数，对比每个维数解的不同，总结规律，得出结论。  4.讨论求解病态问题的算法。  **实验步骤：**  **通过JAVA实现：**   * + - 1. 编写矩阵类，定义相关方法，如，矩阵加减乘和求逆。       2. 继承矩阵类实现Hilbert矩阵类。       3. 编写IterationMethod接口。       4. 通过实现的矩阵类实现J迭代法公式、SOR迭代法公式从而实现Jacobi类和SOR类并实现IterationMethod接口。       5. GaussSeidel类通过继承SOR类，并将参数置为1。       6. 编写Iteration类作为迭代器，设置有最大迭代次数，迭代精度等属性，需要传入IterationMethod接口对象进行迭代。       7. 在Test类中编写Main()函数，进行测试。   **六、实验结果及分析：**  **1.针对Gauss消去法进行分析：**  Gauss消去法实现简单，并且不需要迭代，消耗资源小，若在精度许可的情况下，应该优先选择。  通过表4-2的高斯消去法在多个维度的Hilbert矩阵求解结果可知：  高斯消去法在低维度（如，n = 5）拥有较高的精度，并且消耗资源相较于迭代法小很多，且获得精度比迭代法高。  但是在高维度（如，n = 15）的求解就不是那么令人满意，解中的前几个解分量能保持较高精度，但是之后的解分量的精度随着维数的增大严重失真，误差甚至达到了10的量级。  图6-1 高斯消去法在不同维度下的误差    我们通过折线图6-1可以看到高斯消去法在高维度时的最大误差已经使结果完全失真，前半段呈指数上升趋势，在n=13之后趋于稳定。  图6-2 高斯消去法在1-14维度下的误差    通过图6-2可以看出高斯消去法在1-11维下保持在的以上高精度，但是随着维数的增大误差大小呈指数上升趋势。  所以**高斯消去法适合求解11维含11维以下的Hilbert系数矩阵方程组。**  **2．针对Jacobi迭代法进行分析：**  通过表4-3 Jacobi迭代法对Hilbert系数矩阵方程组的求解结果可知，Jacobi迭代法只在Hilbert矩阵1维和2维时才收敛，3维以上就发散了。  所以**Jacobi迭代法不适合求解Hilbert系数矩阵方程组，因为其不收敛。**  **3. 针对Gauss-Seidel迭代法进行分析：**  通过表4-4 GS迭代法对Hilbert系数矩阵方程组的求解结果可知，无论对于高维或者低维的Hilbert系数矩阵都有不错的收敛性，对于设置的精度为小数点后两位的结果都能迭代出来。这里研究不同维度对GS迭代法迭代次数的影响。  图6-3 高斯迭代法在不同维度的迭代次数    通过图6-3可知，高斯迭代法的迭代次数随着维度的变大并不是线性的增大，而是呈锯齿状的递增趋势，这可能是由于Hilbert矩阵的高病态性质导致的。  所以可以得出**高斯迭代法对于Hilbert系数矩阵方程组收敛，适合求解Hilbert系数矩阵方程组。**  **4. 针对SOR迭代法进行分析：**  通过表4-5 GS迭代法对Hilbert系数矩阵方程组的求解结果可知，同GS迭代法，SOR迭代法无论对于高维或者低维的Hilbert系数矩阵都有不错的收敛性。这里研究不同维度对GS迭代法迭代次数的影响。  图6-4 SOR迭代法在不同维度的迭代次数()    通过图6-4可知，SOR迭代法与GS迭代法一样迭代次数呈锯齿状递增。  所以可以得出**SOR迭代法对于Hilbert系数矩阵方程组收敛，适合求解Hilbert系数矩阵方程组。**  **5．综合分析：**  通过上文的分析可知，对于求解Hilbert系数矩阵的方程组，在1-11维可以使用高斯消去法，因为在此维度内，高斯消去法有较高的精度，且对于资源的消耗很少，因为它不需要迭代。但是高斯消去法前半段的误差呈指数级的递增趋势，所以在n=11以上的维度将使结果完全失真，所以高维度的求解不适合高斯消去法。  高维度的求解可以使用SOR迭代法或者GS迭代法，这两个算法对于求解Hilbert系数矩阵的方程组是收敛的，**这是由于Hilbert矩阵是对称正定矩阵决定的**，但是随着矩阵维度的升高，二者的迭代次数也在不断呈锯齿状增加，所以这两个算法对于超高维的Hilbert矩阵求解消耗的资源巨大，不是很友好。  **6.讨论病态问题的求解方法：**  病态矩阵方程组会因为原始数据的微小扰动而产生严重失真的解。Hilbert矩阵就是一个典型的病态矩阵。  图6-5 Hilbert矩阵在1-15维的1-条件数    图6-6 Hilbert矩阵在1-100维的1-条件数    通过图6-4、图6-5可以看出Hilbert矩阵的1-条件数在1-14维呈指数增大，达到了的量级，继续增大维度，1-条件数不再呈指数增大，而是在上下波动，**这可能也是导致高斯消去法误差在n=13之后开始不再呈指数增大而是反复波动的原因**。  想降低解病态方程组的难度，可以对病态系数矩阵进行预条件处理，降低系数矩阵的条件数，即降低系数矩阵的病态程度。  图6-7 预条件处理后的Hilbert矩阵在1-15维的1-条件数(蓝色线)    图6-8 预条件处理后的Hilbert矩阵在1-100维的1-条件数(蓝色线)    对Hilbert矩阵进行预条件处理后，通过图6-7与图6-8可以看出经过预条件处理后的Hilbert矩阵的1-条件数在同维度的条件下下降了一至两个量级。**说明预条件处理对于病态矩阵有降低其病态程度的效果。**  **同时还可以对线性方程组进行迭代改善**。这里就不再讨论。  **所以可以对病态方程组进行预条件处理和迭代改善，提高解的精度和降低系数矩阵的病态程度。**  **A．附录：**  **代码通过Java实现（分为三个包，有较多类）**  **图片通过Python读取Java输出至txt文本中的数据进行打印，这里就不给出Java输出数据代码和Python打印图代码。**  **Matrix类：**  package org.neu.dataStructure;  public class Matrix {  private double [][] matrix;  private int row;  private int col;  public Matrix(int row, int col) {  this.row = row;  this.col = col;  matrix = new double[row][col];  }  public Matrix(double[][] matrix) {  this.setMatrix(matrix);  }  public int getRow() {  return row;  }  public int getCol() {  return col;  }  public double[][] getMatrix() {  return matrix;  }  public void setMatrix(double[][] matrix) {  this.matrix = matrix;  this.row = matrix.length;  this.col = matrix[0].length;  }  /\*\*  \* Function:Matrix Add  \* @param a:Matrix  \* @param b:Matrix  \* @return :Matrix  \*/  public static Matrix add(Matrix a, Matrix b) throws Exception {  int row = a.getRow();  int col = a.getCol();  if (row != b.getRow() || col != b.getCol()) {  throw new Exception("AddFailed,Can't Add");  }  Matrix result = new Matrix(row, col);  for (int i = 0; i < row; i++) {  for (int j = 0; j < col; j++) {  result.getMatrix()[i][j] = a.getMatrix()[i][j] + b.getMatrix()[i][j];  }  }  return result;  }  /\*\*  \* Function:Matrix Sub  \* @param a:Matrix  \* @param b:Matrix  \* @return :Matrix  \*/  public static Matrix sub(Matrix a, Matrix b) throws Exception {  int row = a.getRow();  int col = a.getCol();  if (row != b.getRow() || col != b.getCol()) {  throw new Exception("SubtractFailed,Can't Subtract");  }  Matrix result = new Matrix(row, col);  for (int i = 0; i < row; i++) {  for (int j = 0; j < col; j++) {  result.getMatrix()[i][j]= a.getMatrix()[i][j] - b.getMatrix()[i][j];  }  }  return result;  }  /\*\*  \* Function:Matrix Multiple Matrix  \* @param a:Matrix  \* @param b:Matrix  \* @return :Matrix  \*/  public static Matrix mul(Matrix a, Matrix b) throws Exception {  int row = a.getRow();  int col = b.getCol();  if (a.getCol() != b.getRow()) {  throw new Exception("MultipleFailed,Can't Multiple");  }  Matrix result = new Matrix(row, col);  for (int i = 0; i < row; i++) {  for (int j = 0; j < col; j++) {  for (int k = 0; k < a.getCol(); k++) {  result.getMatrix()[i][j] += a.getMatrix()[i][k]\*b.getMatrix()[k][j];  }  }  }  return result;  }  /\*\*  \* Function:Matrix Multiple Number  \* @param a Matrix  \* @param b double  \* @return :Matrix  \*/  public static Matrix mul(Matrix a, double b) {  int row = a.getRow();  int col = a.getCol();  Matrix result = new Matrix(row, col);  for (int i = 0; i < row; i++) {  for (int j = 0; j < col; j++) {  result.getMatrix()[i][j] = a.getMatrix()[i][j] \* b;  }  }  return result;  }  /\*\*  \* Function:Square Matrix Inversion  \* @param a :Matrix  \* @return :Matrix  \*/  public static Matrix inver(Matrix a) throws Exception {  int row = a.getRow();  if(a.getCol() != a.getRow()){ //方阵才能使用初等行变换求逆  throw new Exception("InversionFailed,Not Square");  }  double[][] copy = copyMatrix(a);  Matrix result = new Matrix(row, row);  double[][] resultM = result.getMatrix();  for(int i = 0; i < row; i++){ //将result矩阵置为单位矩阵  resultM[i][i] = 1;  }  for(int i = 0; i < row; i++){ //遍历每一列,从左到右  double aii = copy[i][i];  for(int j = 0; j < row; j++){ //遍历行中每个元素，除以aii  resultM[i][j] /= aii;  copy[i][j] /= aii;  }  for(int j = i+1; j < row; j++){ //遍历本行以下的每一行  elementaryRowTrans(row, copy, resultM, i, j);  }  }  for(int i = row-1; i >= 0 ; i--){ //遍历每一列,从右到左  for(int j = i-1; j >= 0; j--){ //遍历本行以上的每一行  elementaryRowTrans(row, copy, resultM, i, j);  }  }  return result;  }  /\*\*  \* Function:CopyMatrix  \* @param a:Matrix  \* @return :double[][]  \*/  public static double[][] copyMatrix(Matrix a) {  double[][] copy = new double[a.getMatrix().length][]; //复制矩阵  for (int i = 0; i < copy.length; i++) {  copy[i] = a.getMatrix()[i].clone();  }  return copy;  }  /\*\*  \*Function:初等行变换  \*/  private static void elementaryRowTrans(int row, double[][] copy, double[][] resultM, int i, int j) {  double coef = -copy[j][i]; //每行系数  for (int k = 0; k < row; k++) { //遍历行中每个元素  resultM[j][k] += coef \* resultM[i][k];  copy[j][k] += coef \* copy[i][k];  }  }  /\*\*  \* Function:Print Matrix  \* @param matrix :Matrix  \*/  public static void print(Matrix matrix) {  double [][] mat = matrix.getMatrix();  for (int i = 0; i < mat.length; i++) {  System.out.print("[");  for (int j = 0; j < mat[0].length; j++) {  System.out.print(mat[i][j]);  if (j != mat[0].length - 1) {  System.out.print(", ");  }  }  System.out.print("]\n");  }  }  /\*\*  \* Function:生成全一矩阵  \*/  public static Matrix ones(int row, int col){  Matrix a = new Matrix(row, col);  double[][] matrix = a.getMatrix();  for (int i = 0; i < matrix.length; i++) {  for (int j = 0; j < matrix[0].length; j++) {  matrix[i][j] = 1;  }  }  return a;  }  /\*\*  \* Function:生成全0矩阵  \*/  public static Matrix zeros(int row, int col){  Matrix a = new Matrix(row, col);  double[][] matrix = a.getMatrix();  for (int i = 0; i < matrix.length; i++) {  for (int j = 0; j < matrix[0].length; j++) {  matrix[i][j] = 0;  }  }  return a;  }  /\*\*  \* Function:生成无对角线的上三角矩阵  \*/  public static Matrix upperTriangularMatrix(Matrix a){  Matrix result = new Matrix(a.getRow(), a.getCol());  for (int i = 0; i < a.getRow(); i++) {  for (int j = i+1; j < a.getCol(); j++) {  result.getMatrix()[i][j] = a.getMatrix()[i][j];  }  }  return result;  }  /\*\*  \* Function:生成无对角线的下三角矩阵  \*/  public static Matrix lowerTriangularMatrix(Matrix a){  Matrix result = new Matrix(a.getRow(), a.getCol());  for (int i = 0; i < a.getCol(); i++) {  for (int j = i+1; j < a.getRow(); j++) {  result.getMatrix()[j][i] = a.getMatrix()[j][i];  }  }  return result;  }  /\*\*  \* Function:生成对角矩阵  \*/  public static Matrix diagonalMatrix(Matrix a) throws Exception {  if(a.getCol() != a.getRow()){  throw new Exception("DiagonalMatrixFailed,Not Square");  }  Matrix result = new Matrix(a.getRow(), a.getCol());  for (int i = 0; i < a.getRow(); i++) {  result.getMatrix()[i][i] = a.getMatrix()[i][i];  }  return result;  }  /\*\*  \* Function:求解矩阵1-条件数  \*/  public static double cond1(Matrix A) throws Exception {  return norm1(A)\*norm1(Matrix.inver(A));  }  /\*\*  \* Function:求解矩阵1-范数  \*/  public static double norm1(Matrix A){  double max = 0;  for (int i = 0; i < A.getCol(); i++) {  double temp = 0;  for (int j = 0; j < A.getRow(); j++) {  temp += A.getMatrix()[j][i];  }  if(max < temp){  max = temp;  }  }  return max;  }  /\*\*  \* Function:对对称正定矩阵进行预处理  \*/  public static Matrix preTreatment(Matrix A) throws Exception {  Matrix C = Matrix.diagonalMatrix(A);  for (int i = 0; i < C.getRow(); i++) {  C.getMatrix()[i][i] = Math.sqrt(C.getMatrix()[i][i]);  }  return Matrix.mul(Matrix.mul(Matrix.inver(C),A),Matrix.inver(C));  }  }  **Hilbert类：**  package org.neu.dataStructure;  public class Hilbert extends Matrix{  public Hilbert(int row) {  super(row, row);  for (int i = 0; i < row; i++) {  for (int j = 0; j < row; j++) {  this.getMatrix()[i][j] = (double)1/(i+j+1);  }  }  }  }  **GaussElimination类：**  package org.neu.operation;  import org.neu.dataStructure.Matrix;  public class GaussElimination { //高斯消去法  public static Matrix gaussElimination(Matrix A, Matrix b) throws Exception {  Matrix x = new Matrix(b.getRow(), b.getCol());  int row = A.getRow();  if(A.getCol() != A.getRow()){  throw new Exception("GaussEliminationFailed");  }  Matrix bcopy = new Matrix(Matrix.copyMatrix(b));  double[][] a = Matrix.copyMatrix(A); //复制矩阵  for(int i = 0; i < row; i++){ //遍历每一列,从左到右  for (int j = i+1; j < row; j++) { //从本行往下一行开始遍历  double lij = a[j][i]/a[i][i];  for (int k = i; k < row; k++) { //遍历一行  a[j][k] -= lij\*a[i][k];  }  bcopy.getMatrix()[j][0] -= lij\*bcopy.getMatrix()[i][0];  }  }  for (int i = row-1; i >= 0; i--) { //求解  double sum = 0;  for (int j = i; j < row; j++) {  sum += a[i][j]\*x.getMatrix()[j][0];  }  x.getMatrix()[i][0] = (bcopy.getMatrix()[i][0] - sum)/a[i][i];  }  return x;  }  }  **Jacobi类：**  package org.neu.operation;  import org.neu.dataStructure.Matrix;  public class GaussElimination { //高斯消去法  public static Matrix gaussElimination(Matrix A, Matrix b) throws Exception {  Matrix x = new Matrix(b.getRow(), b.getCol());  int row = A.getRow();  if(A.getCol() != A.getRow()){  throw new Exception("GaussEliminationFailed");  }  Matrix bcopy = new Matrix(Matrix.copyMatrix(b));  double[][] a = Matrix.copyMatrix(A); //复制矩阵  for(int i = 0; i < row; i++){ //遍历每一列,从左到右  for (int j = i+1; j < row; j++) { //从本行往下一行开始遍历  double lij = a[j][i]/a[i][i];  for (int k = i; k < row; k++) { //遍历一行  a[j][k] -= lij\*a[i][k];  }  bcopy.getMatrix()[j][0] -= lij\*bcopy.getMatrix()[i][0];  }  }  for (int i = row-1; i >= 0; i--) { //求解  double sum = 0;  for (int j = i; j < row; j++) {  sum += a[i][j]\*x.getMatrix()[j][0];  }  x.getMatrix()[i][0] = (bcopy.getMatrix()[i][0] - sum)/a[i][i];  }  return x;  }  }  **GaussSeidel类：**  package org.neu.operation;  import org.neu.dataStructure.Matrix;  public class GaussSeidel extends SOR{  /\*\*  \* Function: GaussSeidel迭代法  \* @param A 系数矩阵  \* @param x 初始解  \* @param b 常数矩阵  \*/  public GaussSeidel(Matrix A, Matrix x, Matrix b) throws Exception {  super(A, x, b, 1);  }  public Matrix process() throws Exception {  return super.process();  }  }  **SOR类：**  package org.neu.operation;  import org.neu.dataStructure.Matrix;  public class GaussSeidel extends SOR{  /\*\*  \* Function: GaussSeidel迭代法  \* @param A 系数矩阵  \* @param x 初始解  \* @param b 常数矩阵  \*/  public GaussSeidel(Matrix A, Matrix x, Matrix b) throws Exception {  super(A, x, b, 1);  }  public Matrix process() throws Exception {  return super.process();  }  }  **IterationMethod接口：**  package org.neu.operation;  import org.neu.dataStructure.Matrix;  public interface IterationMethod {  Matrix process() throws Exception;  void setX(Matrix x);  }  **Iteration类：**  package org.neu.operation;  import org.neu.dataStructure.Matrix;  public class Iteration {  private int scale; //小数点后的位数要求  private int k = 0; //迭代次数  private int kMax = 10000000; //最大迭代次数  private IterationMethod iter; //迭代函数  private Matrix xAcc; //精确解  public Iteration(int scale, IterationMethod iter, Matrix xAcc) {  if (scale < 0) {  throw new IllegalArgumentException(  "The scale must be a positive integer or zero");  }  this.scale = scale;  this.iter = iter;  this.xAcc = xAcc;  }  /\*\*  \*Function:迭代  \*/  public Matrix iteration() throws Exception {  Matrix xn = null;  while(k < kMax){  xn = iter.process();  if(checkPoint(xn, xAcc)){  return xn;  }  iter.setX(xn);  k ++;  }  return xn;  }  /\*\*  \* Function:检查精度  \*/  private boolean checkPoint(Matrix xn, Matrix xAcc){  double max = 0;  double temp;  for (int i = 0; i < xAcc.getRow(); i++) {  if(max < (temp = Math.abs(xn.getMatrix()[i][0] - xAcc.getMatrix()[i][0]))){  max = temp;  }  }  double scaleDouble = creatScaleNum(scale);  return (max < scaleDouble);  }  /\*\*  \* Function:创建精度小数  \*/  private double creatScaleNum(int scale){  String s = "0.";  for (int i = 0; i < scale-1; i++) {  s += "0";  }  s += "1";  return Double.valueOf(s);  }  public int getK() {  return k;  }  }  **Test类：**  package org.neu.main;  import org.neu.dataStructure.Hilbert;  import org.neu.dataStructure.Matrix;  import org.neu.operation.\*;  import java.util.Scanner;  public class Test {  public static void main(String[] args) {  try {  while(true){  System.out.println("Please enter the dimension:");  Scanner scan = new Scanner(System.in);  int dimen = scan.nextInt();  doOneTime(dimen, 2, 1.46);  }  }catch (Exception e) {  e.printStackTrace();  }  }  public static void doOneTime(int dimension, int scale, double w) throws Exception {  Hilbert hilbert = new Hilbert(dimension);  Matrix xAcc = Matrix.ones(dimension,1);  Matrix xi = Matrix.zeros(dimension,1);  Matrix b = Matrix.mul(hilbert, xAcc);  System.out.println(dimension+"维高斯消去法结果：");  Matrix.print(GaussElimination.gaussElimination(hilbert, b));  System.out.println("\n");  System.out.println(dimension+"维，精度"+scale+"，Jacobi迭代法结果：");  Iteration iter = new Iteration(scale, new Jacobi(hilbert, xi, b), xAcc);  Matrix.print(iter.iteration());  System.out.println("迭代次数："+iter.getK());  System.out.println("\n");  System.out.println(dimension+"维，精度："+scale+"，GaussSeidel迭代法结果：");  iter = new Iteration(scale, new GaussSeidel(hilbert, xi, b), xAcc);  Matrix.print(iter.iteration());  System.out.println("迭代次数："+iter.getK());  System.out.println("\n");  System.out.println(dimension+"维，精度："+scale+"，参数w："+w+"，SOR迭代法结果：");  iter = new Iteration(scale, new SOR(hilbert, xi, b, w), xAcc);  Matrix.print(iter.iteration());  System.out.println("迭代次数："+iter.getK());  System.out.println("\n");  }  } | | | | | | | | | | |