北京航空航天大学数学科学学院实验报告

|  |  |  |  |
| --- | --- | --- | --- |
| 课程名称：科学计算通识实验课 | | 实验名称：实验三： 线性方程组的迭代求解 | |
| 实验类型： 演示性实验□ 验证性实验☑ 综合性实验□ 设计性实验□ | | | |
| 班级：暑期班 | 姓名：涂磊 | | 学号：22377211 |
| 实验日期： 2024/7/7 | 指导教师：冯成亮 | | 实验成绩： |
| 实验环境：（所用仪器设备及软件）  Windows + VS-code, Ubuntu 20.04.6 + g++ | | | |
| 实验目的与实验内容：  【目的要求】  通过本实验使学生进一步熟悉个人电脑上C++代码的编写与调试，服务器上的代码编译与运行；熟悉求解线性方程组的Jacobi迭代法、Gauss-Seidel迭代法和SOR松弛迭代法；了解以上方法的算法的稳定性与收敛速度特点；了解以上方法的算法适用性与稳定性，并试着使用以上方法解决两点边值问题。  【实验内容】  实验1.1：（Jacobi迭代法求解线性方程组2）  针对方程组  采用Jacobi迭代法进行迭代求解，记录收敛终止条件下的迭代数据，如果结果不收敛，记录K=50终止时的绝对误差表。  实验1.2：（Jacobi迭代法求解线性方程组3）  针对方程组  采用Jacobi迭代法进行迭代求解，记录收敛终止条件下的迭代数据，如果结果不收敛，记录K=50终止时的绝对误差表。  实验2.1：（Gauss-Seidel迭代法求解线性方程组2）  针对方程组  采用Gauss-Seidel迭代法进行迭代求解，记录收敛终止条件下的迭代数据，如果结果不收敛，记录K=50终止时的绝对误差表。  实验2.2：（Gauss-Seidel迭代法求解线性方程组3）  针对方程组  采用Gauss-Seidel迭代法进行迭代求解，记录收敛终止条件下的迭代数据，如果结果不收敛，记录K=50终止时的绝对误差表。  实验3.1：（SOR松弛迭代法求解线性方程组2）  针对方程组  采用SOR松弛迭代法进行迭代求解，（），记录收敛终止条件下的迭代数据，如果结果不收敛，记录K=50终止时的绝对误差表。  实验3.2：（SOR松弛迭代法求解线性方程组3）  针对方程组  采用SOR松弛迭代法进行迭代求解，（），记录收敛终止条件下的迭代数据，如果结果不收敛，记录K=50终止时的绝对误差表。  实验4.1：使用迭代法对两点边值问题的有限差分求解    （精确解。）  对于进行均匀网格剖分，  ，.  得线性方程组：  即可求得N=10,20,40,80对应的，并可计算误差，。（尝试比较N=160时，直接求解法与迭代求解法的计算时间\*） | | | |
| 实验过程与结果：  实验1.1：（Jacobi迭代法求解线性方程组2）  #include <stdio.h>  #include <stdlib.h>  #include <math.h>  #include "error.hpp"  double max(double a, double b, double c) {  return a > b? (a > c? a : c) : (b > c? b : c);  }      int main(void) {  // 输入矩阵大小n  /\* int n;  printf("Enter the size of the matrix: ");  scanf("%d", &n); \*/  int n=3;  /\* //输入最大迭代次数  int max\_iter;  printf("Enter the maximum number of iterations: ");  scanf("%d", &max\_iter); \*/    /\* // 输入A  double \*\*A = NULL;  A = (double \*\*)malloc(n \* sizeof(double \*));  for (int i = 0; i < n; i++) {  A[i] = (double \*)malloc(n \* sizeof(double));  }  if(A == NULL) {  printf("Memory allocation failed.\n");  return 1;  }  printf("Enter the elements of the matrix:\n");  for (int i = 0; i < n; i++) {  for (int j = 0; j < n; j++) {  scanf("%lf", &A[i][j]);  }  printf("\n");  }    // 输入b  printf("Enter the number of iterations:\n");  double\* b = (double\*)malloc(n \* sizeof(double));  if(b == NULL) {  printf("Memory allocation failed.\n");  return 1;  }  for (int i = 0; i < n; i++) {  scanf("%lf", &b[i]);  }  \*/  // 进行Jacobi迭代  double A[3][3] = {{10.0, 3.0, 1.0}, {2.0, -10.0, 3.0}, {1.0, 3.0, 10.0}};  double b[3] = {14.0, -5.0, 14.0};  //计算D矩阵  double\*\* D = (double\*\*)malloc(n \* sizeof(double\*));  for (int i = 0; i < n; i++) {  D[i] = (double\*)malloc(n \* sizeof(double));  }  if(D == NULL) {  printf("Memory allocation failed.\n");  return 1;  }  for (int i = 0; i < n; i++){  for (int j = 0; j < n; j++) {  if (i == j) {  D[i][j] = A[i][j];  } else {  D[i][j] = 0.0;  }  }  }  /\* //计算L矩阵  double\*\* L = (double\*\*)malloc(n \* sizeof(double\*));  for (int i = 0; i < n; i++) {  L[i] = (double\*)malloc(n \* sizeof(double));  }  if(L == NULL) {  printf("Memory allocation failed.\n");  return 1;  }  for (int i = 0; i < n; i++) {  for (int j = 0; j < n; j++) {  if (i == j) {  L[i][j] = 0.0;  } else if (j > i) {  L[i][j] = -A[i][j];  } else {  L[i][j] = 0.0;  }  }  }    //计算U矩阵  double\*\* U = (double\*\*)malloc(n \* sizeof(double\*));  for (int i = 0; i < n; i++) {  U[i] = (double\*)malloc(n \* sizeof(double));  }  if(U == NULL) {  printf("Memory allocation failed.\n");  return 1;  }  for (int i = 0; i < n; i++) {  for (int j = 0; j < n; j++) {  if (i == j) {  U[i][j] = 0.0;  } else if (j < i) {  U[i][j] = -A[i][j];  } else {  U[i][j] = 0.0;  }  }  } \*/  /\* // 输出矩阵A  printf("The matrix A is:\n");  for (int i = 0; i < n; i++) {  for (int j = 0; j < n; j++) {  printf("%lf\t", A[i][j]);  }  printf("\n");  }    // 输出矩阵D  printf("The matrix D is:\n");  for (int i = 0; i < n; i++) {  for (int j = 0; j < n; j++) {  printf("%lf\t", D[i][j]);  }  printf("\n");  }  // 输出矩阵b  printf("The vector b is:\n");  for (int i = 0; i < n; i++) {  printf("%lf\t", b[i]);  }  printf("\n");  \*/  //设置数列来存储n维向量的n个分量迭代结果    // 定义最大迭代次数  int max\_iter = 50;  double\*\* x = (double\*\*)malloc((max\_iter+1) \* sizeof(double\*));  for (int i = 0; i < (max\_iter+1); i++) {  x[i] = (double\*)malloc(n \* sizeof(double));  }  if(x == NULL) {  printf("Memory allocation failed.\n");  return 1;  }  // 初始化n维向量的n个分量为0'  for (int i = 0; i < n; i++) {  x[0][i] = 0.0;  }  //存储L2范数的数组  double\* diff = (double\*)malloc((max\_iter+1) \* sizeof(double));  if(diff == NULL) {  printf("Memory allocation failed.\n");  return 1;  }  // 定义初始的L2范数  diff[0] = 9999;  // 开始迭代  int iter = 1;  double eps = 1e-5;    // 开始迭代  // 定义结果判断是否收敛    int result = 0;  while (iter <= max\_iter) {  for (int i = 0; i < n; i++) {  double sum = 0.0;  for (int j = 0; j < n; j++) {  if (j != i) {  sum += A[i][j] \* x[iter-1][j];  }  }  x[iter][i] = (b[i]-sum) / D[i][i];  }    // L2范数计算  diff[iter] = 0.0;  for (int i = 0; i < n; i++) {  diff[iter] += (x[iter][i] - x[iter-1][i])\*(x[iter][i] - x[iter-1][i]);  }  diff[iter] = sqrt(diff[iter]);      // 输出当前迭代结果  printf("Iteration %d:\n", iter);  printf("x = [");  for (int i = 0; i < n; i++) {  printf("%lf ", x[iter][i]);  }  printf("]\n");  printf("L2\_error = %lf\n", diff[iter]);  // 判断是否收敛  if (diff[iter] < eps) {  result = 1;  break;  }  iter++;  }  if(iter > max\_iter) {  iter--;  }  printf("----------\n");    double\* x\_real = (double\*)malloc(n \* sizeof(double));  if(x\_real == NULL) {  printf("Memory allocation failed.\n");  return 1;  }  x\_real[0] = 1;  x\_real[1] = 1;  x\_real[2] = 1;      // 输出迭代过程  printf("--------------------------------------------\n");  printf("inter\tx1\t\tx2\t\tx3\t\tL\_infinity\tL\_2\_error\n");  printf("--------------------------------------------\n");  printf("%d\t%lf\t%lf\t%lf\t%lf\t%s\n", 0, x[0][0], x[0][1], x[0][2],max(fabs(x[0][0]-x\_real[0]),fabs(x[0][1]-x\_real[1]),fabs(x[0][2])-x\_real[2]), "initial error");  for (int i = 1; i <= iter; i++){  printf("%d\t%lf\t%lf\t%lf\t%lf\t%lf\n", i, x[i][0], x[i][1], x[i][2],max(fabs(x[i][0]-1),fabs(x[i][1]-1),fabs(x[i][2])-1), diff[i]);  }  printf("--------------------------------------------\n");    // 输出结果  if (result == 1) {  printf("The result was found within %d iterations.\n", iter);  } else {  //告诉用户结果未收敛  printf("The result was not found within %d iterations.\n", max\_iter);    // 输出绝对误差(需要知道真正结果)  printf("The absolute error is (%lf,%lf,%lf)'.\n",absolute\_error(x[iter][0],1),absolute\_error(x[iter][1],1),absolute\_error(x[iter][2],1));  // 输出x带入方程计算的L2范数误差（不需要知道真正结果）  double \*x\_dairu = (double\*)malloc(n \* sizeof(double));  if(x\_dairu == NULL) {  printf("Memory allocation failed.\n");  return 1;  }  for (int i = 0; i < n; i++) {  for (int j = 0; j < n; j++) {  x\_dairu[i] += A[i][j] \* x[max\_iter][j];  }  x\_dairu[i] -= b[i];  }  double error\_norm = 0.0;  for (int i = 0; i < n; i++) {  error\_norm += (x\_dairu[i] \* x\_dairu[i]);  }  error\_norm = sqrt(error\_norm);  printf("The L2 norm error when calculate result brought into the equation is %lf.\n", error\_norm);  }        free(D);  free(x);    free(diff);  return 0;  }    实验1.2：（Jacobi迭代法求解线性方程组3）  #include <stdio.h>  #include <stdlib.h>  #include <math.h>  #include "error.hpp"  double max(double a, double b, double c) {  return a > b? (a > c? a : c) : (b > c? b : c);  }      int main(void) {  // 输入矩阵大小n  /\* int n;  printf("Enter the size of the matrix: ");  scanf("%d", &n); \*/  int n=3;  /\* //输入最大迭代次数  int max\_iter;  printf("Enter the maximum number of iterations: ");  scanf("%d", &max\_iter); \*/    /\* // 输入A  double \*\*A = NULL;  A = (double \*\*)malloc(n \* sizeof(double \*));  for (int i = 0; i < n; i++) {  A[i] = (double \*)malloc(n \* sizeof(double));  }  if(A == NULL) {  printf("Memory allocation failed.\n");  return 1;  }  printf("Enter the elements of the matrix:\n");  for (int i = 0; i < n; i++) {  for (int j = 0; j < n; j++) {  scanf("%lf", &A[i][j]);  }  printf("\n");  }    // 输入b  printf("Enter the number of iterations:\n");  double\* b = (double\*)malloc(n \* sizeof(double));  if(b == NULL) {  printf("Memory allocation failed.\n");  return 1;  }  for (int i = 0; i < n; i++) {  scanf("%lf", &b[i]);  }  \*/  // 进行Jacobi迭代  double A[3][3] = {{4.0, -2.0, -4.0}, {-2.0, 17.0, 10.0}, {-4.0, 10.0, 9.0}};  double b[3] = {10.0, 3.0, -7.0};  //计算D矩阵  double\*\* D = (double\*\*)malloc(n \* sizeof(double\*));  for (int i = 0; i < n; i++) {  D[i] = (double\*)malloc(n \* sizeof(double));  }  if(D == NULL) {  printf("Memory allocation failed.\n");  return 1;  }  for (int i = 0; i < n; i++){  for (int j = 0; j < n; j++) {  if (i == j) {  D[i][j] = A[i][j];  } else {  D[i][j] = 0.0;  }  }  }  /\* //计算L矩阵  double\*\* L = (double\*\*)malloc(n \* sizeof(double\*));  for (int i = 0; i < n; i++) {  L[i] = (double\*)malloc(n \* sizeof(double));  }  if(L == NULL) {  printf("Memory allocation failed.\n");  return 1;  }  for (int i = 0; i < n; i++) {  for (int j = 0; j < n; j++) {  if (i == j) {  L[i][j] = 0.0;  } else if (j > i) {  L[i][j] = -A[i][j];  } else {  L[i][j] = 0.0;  }  }  }    //计算U矩阵  double\*\* U = (double\*\*)malloc(n \* sizeof(double\*));  for (int i = 0; i < n; i++) {  U[i] = (double\*)malloc(n \* sizeof(double));  }  if(U == NULL) {  printf("Memory allocation failed.\n");  return 1;  }  for (int i = 0; i < n; i++) {  for (int j = 0; j < n; j++) {  if (i == j) {  U[i][j] = 0.0;  } else if (j < i) {  U[i][j] = -A[i][j];  } else {  U[i][j] = 0.0;  }  }  } \*/  /\* // 输出矩阵A  printf("The matrix A is:\n");  for (int i = 0; i < n; i++) {  for (int j = 0; j < n; j++) {  printf("%lf\t", A[i][j]);  }  printf("\n");  }    // 输出矩阵D  printf("The matrix D is:\n");  for (int i = 0; i < n; i++) {  for (int j = 0; j < n; j++) {  printf("%lf\t", D[i][j]);  }  printf("\n");  }  // 输出矩阵b  printf("The vector b is:\n");  for (int i = 0; i < n; i++) {  printf("%lf\t", b[i]);  }  printf("\n");  \*/  //设置数列来存储n维向量的n个分量迭代结果    // 定义最大迭代次数  int max\_iter = 50;  double\*\* x = (double\*\*)malloc((max\_iter+1) \* sizeof(double\*));  for (int i = 0; i < (max\_iter+1); i++) {  x[i] = (double\*)malloc(n \* sizeof(double));  }  if(x == NULL) {  printf("Memory allocation failed.\n");  return 1;  }  // 初始化n维向量的n个分量为0'  for (int i = 0; i < n; i++) {  x[0][i] = 0.0;  }  //存储L2范数的数组  double\* diff = (double\*)malloc((max\_iter+1) \* sizeof(double));  if(diff == NULL) {  printf("Memory allocation failed.\n");  return 1;  }  // 定义初始的L2范数  diff[0] = 9999;  // 开始迭代  int iter = 1;  double eps = 1e-5;    // 开始迭代  // 定义结果判断是否收敛    int result = 0;  while (iter <= max\_iter) {  for (int i = 0; i < n; i++) {  double sum = 0.0;  for (int j = 0; j < n; j++) {  if (j != i) {  sum += A[i][j] \* x[iter-1][j];  }  }  x[iter][i] = (b[i]-sum) / D[i][i];  }    // L2范数计算  diff[iter] = 0.0;  for (int i = 0; i < n; i++) {  diff[iter] += (x[iter][i] - x[iter-1][i])\*(x[iter][i] - x[iter-1][i]);  }  diff[iter] = sqrt(diff[iter]);      // 输出当前迭代结果  printf("Iteration %d:\n", iter);  printf("x = [");  for (int i = 0; i < n; i++) {  printf("%lf ", x[iter][i]);  }  printf("]\n");  printf("L2\_error = %lf\n", diff[iter]);  // 判断是否收敛  if (diff[iter] < eps) {  result = 1;  break;  }  iter++;  }  if(iter > max\_iter) {  iter--;  }  printf("----------\n");    double\* x\_real = (double\*)malloc(n \* sizeof(double));  if(x\_real == NULL) {  printf("Memory allocation failed.\n");  return 1;  }  x\_real[0] = 2;  x\_real[1] = 1;  x\_real[2] = -1;      // 输出迭代过程  printf("--------------------------------------------\n");  printf("inter\tx1\t\tx2\t\tx3\t\tL\_infinity\tL\_2\_error\n");  printf("--------------------------------------------\n");  printf("%d\t%lf\t%lf\t%lf\t%lf\t%s\n", 0, x[0][0], x[0][1], x[0][2],max(fabs(x[0][0]-x\_real[0]),fabs(x[0][1]-x\_real[1]),fabs(x[0][2])-x\_real[2]), "initial error");  for (int i = 1; i <= iter; i++){  printf("%d\t%lf\t%lf\t%lf\t%lf\t%lf\n", i, x[i][0], x[i][1], x[i][2],max(fabs(x[i][0]-1),fabs(x[i][1]-1),fabs(x[i][2])-1), diff[i]);  }  printf("--------------------------------------------\n");    // 输出结果  if (result == 1) {  printf("The result was found within %d iterations.\n", iter);  } else {  //告诉用户结果未收敛  printf("The result was not found within %d iterations.\n", max\_iter);    // 输出绝对误差(需要知道真正结果)  printf("The absolute error is (%lf,%lf,%lf)'.\n",absolute\_error(x[iter][0],1),absolute\_error(x[iter][1],1),absolute\_error(x[iter][2],1));  // 输出x带入方程计算的L2范数误差（不需要知道真正结果）  double \*x\_dairu = (double\*)malloc(n \* sizeof(double));  if(x\_dairu == NULL) {  printf("Memory allocation failed.\n");  return 1;  }  for (int i = 0; i < n; i++) {  for (int j = 0; j < n; j++) {  x\_dairu[i] += A[i][j] \* x[max\_iter][j];  }  x\_dairu[i] -= b[i];  }  double error\_norm = 0.0;  for (int i = 0; i < n; i++) {  error\_norm += (x\_dairu[i] \* x\_dairu[i]);  }  error\_norm = sqrt(error\_norm);  printf("The L2 norm error when calculate result brought into the equation is %lf.\n", error\_norm);  }    // 释放内存  for (int i = 0; i < n; i++) {  free(A[i]);  free(D[i]);  free(x[i]);  }    free(D);  free(x);    free(diff);  return 0;  }      实验2.1：（Gauss-Seidel迭代法求解线性方程组2）  #include <stdio.h>  #include <stdlib.h>  #include <math.h>  #include "error.hpp"  double max(double a, double b, double c) {  return a > b? (a > c? a : c) : (b > c? b : c);  }      int main(void) {  // 输入矩阵大小n  /\* int n;  printf("Enter the size of the matrix: ");  scanf("%d", &n); \*/  int n=3;  /\* //输入最大迭代次数  int max\_iter;  printf("Enter the maximum number of iterations: ");  scanf("%d", &max\_iter); \*/    /\* // 输入A  double \*\*A = NULL;  A = (double \*\*)malloc(n \* sizeof(double \*));  for (int i = 0; i < n; i++) {  A[i] = (double \*)malloc(n \* sizeof(double));  }  if(A == NULL) {  printf("Memory allocation failed.\n");  return 1;  }  printf("Enter the elements of the matrix:\n");  for (int i = 0; i < n; i++) {  for (int j = 0; j < n; j++) {  scanf("%lf", &A[i][j]);  }  printf("\n");  }    // 输入b  printf("Enter the number of iterations:\n");  double\* b = (double\*)malloc(n \* sizeof(double));  if(b == NULL) {  printf("Memory allocation failed.\n");  return 1;  }  for (int i = 0; i < n; i++) {  scanf("%lf", &b[i]);  }  \*/  // 进行Jacobi迭代  double A[3][3] = {{10.0, 3.0, 1.0}, {2.0, -10.0, 3.0}, {1.0, 3.0, 10.0}};  double b[3] = {14.0, -5.0, 14.0};  //计算D矩阵  double\*\* D = (double\*\*)malloc(n \* sizeof(double\*));  for (int i = 0; i < n; i++) {  D[i] = (double\*)malloc(n \* sizeof(double));  }  if(D == NULL) {  printf("Memory allocation failed.\n");  return 1;  }  for (int i = 0; i < n; i++){  for (int j = 0; j < n; j++) {  if (i == j) {  D[i][j] = A[i][j];  } else {  D[i][j] = 0.0;  }  }  }  /\* //计算L矩阵  double\*\* L = (double\*\*)malloc(n \* sizeof(double\*));  for (int i = 0; i < n; i++) {  L[i] = (double\*)malloc(n \* sizeof(double));  }  if(L == NULL) {  printf("Memory allocation failed.\n");  return 1;  }  for (int i = 0; i < n; i++) {  for (int j = 0; j < n; j++) {  if (i == j) {  L[i][j] = 0.0;  } else if (j > i) {  L[i][j] = -A[i][j];  } else {  L[i][j] = 0.0;  }  }  }    //计算U矩阵  double\*\* U = (double\*\*)malloc(n \* sizeof(double\*));  for (int i = 0; i < n; i++) {  U[i] = (double\*)malloc(n \* sizeof(double));  }  if(U == NULL) {  printf("Memory allocation failed.\n");  return 1;  }  for (int i = 0; i < n; i++) {  for (int j = 0; j < n; j++) {  if (i == j) {  U[i][j] = 0.0;  } else if (j < i) {  U[i][j] = -A[i][j];  } else {  U[i][j] = 0.0;  }  }  } \*/  /\* // 输出矩阵A  printf("The matrix A is:\n");  for (int i = 0; i < n; i++) {  for (int j = 0; j < n; j++) {  printf("%lf\t", A[i][j]);  }  printf("\n");  }    // 输出矩阵D  printf("The matrix D is:\n");  for (int i = 0; i < n; i++) {  for (int j = 0; j < n; j++) {  printf("%lf\t", D[i][j]);  }  printf("\n");  }  // 输出矩阵b  printf("The vector b is:\n");  for (int i = 0; i < n; i++) {  printf("%lf\t", b[i]);  }  printf("\n");  \*/  //设置数列来存储n维向量的n个分量迭代结果    // 定义最大迭代次数  int max\_iter = 50;  double\*\* x = (double\*\*)malloc((max\_iter+1) \* sizeof(double\*));  for (int i = 0; i < (max\_iter+1); i++) {  x[i] = (double\*)malloc(n \* sizeof(double));  }  if(x == NULL) {  printf("Memory allocation failed.\n");  return 1;  }  // 初始化n维向量的n个分量为0'  for (int i = 0; i < n; i++) {  x[0][i] = 0.0;  }  //存储L2范数的数组  double\* diff = (double\*)malloc((max\_iter+1) \* sizeof(double));  if(diff == NULL) {  printf("Memory allocation failed.\n");  return 1;  }  // 定义初始的L2范数  diff[0] = 9999;  // 开始迭代  int iter = 1;  double eps = 1e-5;    // 开始迭代  // 定义结果判断是否收敛    int result = 0;  while (iter <= max\_iter) {  for (int i = 0; i < n; i++) {  double sum = 0.0;  for (int j = 0; j < n; j++) {  if (j != i) {  if (j < i) {  sum += A[i][j] \* x[iter][j];  } else {  sum += A[i][j] \* x[iter-1][j];  }  }  }  x[iter][i] = (b[i]-sum) / D[i][i];  }    // L2范数计算  diff[iter] = 0.0;  for (int i = 0; i < n; i++) {  diff[iter] += (x[iter][i] - x[iter-1][i])\*(x[iter][i] - x[iter-1][i]);  }  diff[iter] = sqrt(diff[iter]);      // 输出当前迭代结果  printf("Iteration %d:\n", iter);  printf("x = [");  for (int i = 0; i < n; i++) {  printf("%lf ", x[iter][i]);  }  printf("]\n");  printf("L2\_error = %lf\n", diff[iter]);  // 判断是否收敛  if (diff[iter] < eps) {  result = 1;  break;  }  iter++;  }  if(iter > max\_iter) {  iter--;  }  printf("----------\n");    double\* x\_real = (double\*)malloc(n \* sizeof(double));  if(x\_real == NULL) {  printf("Memory allocation failed.\n");  return 1;  }  x\_real[0] = 1;  x\_real[1] = 1;  x\_real[2] = 1;      // 输出迭代过程  printf("--------------------------------------------\n");  printf("inter\tx1\t\tx2\t\tx3\t\tL\_infinity\tL\_2\_error\n");  printf("--------------------------------------------\n");  printf("%d\t%lf\t%lf\t%lf\t%lf\t%s\n", 0, x[0][0], x[0][1], x[0][2],max(fabs(x[0][0]-x\_real[0]),fabs(x[0][1]-x\_real[1]),fabs(x[0][2])-x\_real[2]), "initial error");  for (int i = 1; i <= iter; i++){  printf("%d\t%lf\t%lf\t%lf\t%lf\t%lf\n", i, x[i][0], x[i][1], x[i][2],max(fabs(x[i][0]-1),fabs(x[i][1]-1),fabs(x[i][2])-1), diff[i]);  }  printf("--------------------------------------------\n");    // 输出结果  if (result == 1) {  printf("The result was found within %d iterations.\n", iter);  } else {  //告诉用户结果未收敛  printf("The result was not found within %d iterations.\n", max\_iter);    // 输出绝对误差(需要知道真正结果)  printf("The absolute error is (%lf,%lf,%lf)'.\n",absolute\_error(x[iter][0],1),absolute\_error(x[iter][1],1),absolute\_error(x[iter][2],1));  // 输出x带入方程计算的L2范数误差（不需要知道真正结果）  double \*x\_dairu = (double\*)malloc(n \* sizeof(double));  if(x\_dairu == NULL) {  printf("Memory allocation failed.\n");  return 1;  }  for (int i = 0; i < n; i++) {  for (int j = 0; j < n; j++) {  x\_dairu[i] += A[i][j] \* x[max\_iter][j];  }  x\_dairu[i] -= b[i];  }  double error\_norm = 0.0;  for (int i = 0; i < n; i++) {  error\_norm += (x\_dairu[i] \* x\_dairu[i]);  }  error\_norm = sqrt(error\_norm);  printf("The L2 norm error when calculate result brought into the equation is %lf.\n", error\_norm);  }    // 释放内存  for (int i = 0; i < n; i++) {  free(A[i]);  free(D[i]);  free(x[i]);  }    free(D);  free(x);    free(diff);  return 0;  }    实验2.2：（Gauss-Seidel迭代法求解线性方程组3）  #include <stdio.h>  #include <stdlib.h>  #include <math.h>  #include "error.hpp"  double max(double a, double b, double c) {  return a > b? (a > c? a : c) : (b > c? b : c);  }      int main(void) {  // 输入矩阵大小n  /\* int n;  printf("Enter the size of the matrix: ");  scanf("%d", &n); \*/  int n=3;  /\* //输入最大迭代次数  int max\_iter;  printf("Enter the maximum number of iterations: ");  scanf("%d", &max\_iter); \*/    /\* // 输入A  double \*\*A = NULL;  A = (double \*\*)malloc(n \* sizeof(double \*));  for (int i = 0; i < n; i++) {  A[i] = (double \*)malloc(n \* sizeof(double));  }  if(A == NULL) {  printf("Memory allocation failed.\n");  return 1;  }  printf("Enter the elements of the matrix:\n");  for (int i = 0; i < n; i++) {  for (int j = 0; j < n; j++) {  scanf("%lf", &A[i][j]);  }  printf("\n");  }    // 输入b  printf("Enter the number of iterations:\n");  double\* b = (double\*)malloc(n \* sizeof(double));  if(b == NULL) {  printf("Memory allocation failed.\n");  return 1;  }  for (int i = 0; i < n; i++) {  scanf("%lf", &b[i]);  }  \*/  // 进行Jacobi迭代  double A[3][3] = {{4.0, -2.0, -4.0}, {-2.0, 17.0, 10.0}, {-4.0, 10.0, 9.0}};  double b[3] = {10.0, 3.0, -7.0};  //计算D矩阵  double\*\* D = (double\*\*)malloc(n \* sizeof(double\*));  for (int i = 0; i < n; i++) {  D[i] = (double\*)malloc(n \* sizeof(double));  }  if(D == NULL) {  printf("Memory allocation failed.\n");  return 1;  }  for (int i = 0; i < n; i++){  for (int j = 0; j < n; j++) {  if (i == j) {  D[i][j] = A[i][j];  } else {  D[i][j] = 0.0;  }  }  }  /\* //计算L矩阵  double\*\* L = (double\*\*)malloc(n \* sizeof(double\*));  for (int i = 0; i < n; i++) {  L[i] = (double\*)malloc(n \* sizeof(double));  }  if(L == NULL) {  printf("Memory allocation failed.\n");  return 1;  }  for (int i = 0; i < n; i++) {  for (int j = 0; j < n; j++) {  if (i == j) {  L[i][j] = 0.0;  } else if (j > i) {  L[i][j] = -A[i][j];  } else {  L[i][j] = 0.0;  }  }  }    //计算U矩阵  double\*\* U = (double\*\*)malloc(n \* sizeof(double\*));  for (int i = 0; i < n; i++) {  U[i] = (double\*)malloc(n \* sizeof(double));  }  if(U == NULL) {  printf("Memory allocation failed.\n");  return 1;  }  for (int i = 0; i < n; i++) {  for (int j = 0; j < n; j++) {  if (i == j) {  U[i][j] = 0.0;  } else if (j < i) {  U[i][j] = -A[i][j];  } else {  U[i][j] = 0.0;  }  }  } \*/  /\* // 输出矩阵A  printf("The matrix A is:\n");  for (int i = 0; i < n; i++) {  for (int j = 0; j < n; j++) {  printf("%lf\t", A[i][j]);  }  printf("\n");  }    // 输出矩阵D  printf("The matrix D is:\n");  for (int i = 0; i < n; i++) {  for (int j = 0; j < n; j++) {  printf("%lf\t", D[i][j]);  }  printf("\n");  }  // 输出矩阵b  printf("The vector b is:\n");  for (int i = 0; i < n; i++) {  printf("%lf\t", b[i]);  }  printf("\n");  \*/  //设置数列来存储n维向量的n个分量迭代结果    // 定义最大迭代次数  int max\_iter = 50;  double\*\* x = (double\*\*)malloc((max\_iter+1) \* sizeof(double\*));  for (int i = 0; i < (max\_iter+1); i++) {  x[i] = (double\*)malloc(n \* sizeof(double));  }  if(x == NULL) {  printf("Memory allocation failed.\n");  return 1;  }  // 初始化n维向量的n个分量为0'  for (int i = 0; i < n; i++) {  x[0][i] = 0.0;  }  //存储L2范数的数组  double\* diff = (double\*)malloc((max\_iter+1) \* sizeof(double));  if(diff == NULL) {  printf("Memory allocation failed.\n");  return 1;  }  // 定义初始的L2范数  diff[0] = 9999;  // 开始迭代  int iter = 1;  double eps = 1e-5;    // 开始迭代  // 定义结果判断是否收敛    int result = 0;  while (iter <= max\_iter) {  for (int i = 0; i < n; i++) {  double sum = 0.0;  for (int j = 0; j < n; j++) {  if (j != i) {  if (j < i) {  sum += A[i][j] \* x[iter][j];  } else {  sum += A[i][j] \* x[iter-1][j];  }  }  }  x[iter][i] = (b[i]-sum) / D[i][i];  }    // L2范数计算  diff[iter] = 0.0;  for (int i = 0; i < n; i++) {  diff[iter] += (x[iter][i] - x[iter-1][i])\*(x[iter][i] - x[iter-1][i]);  }  diff[iter] = sqrt(diff[iter]);      // 输出当前迭代结果  printf("Iteration %d:\n", iter);  printf("x = [");  for (int i = 0; i < n; i++) {  printf("%lf ", x[iter][i]);  }  printf("]\n");  printf("L2\_error = %lf\n", diff[iter]);  // 判断是否收敛  if (diff[iter] < eps) {  result = 1;  break;  }  iter++;  }  if(iter > max\_iter) {  iter--;  }  printf("----------\n");    double\* x\_real = (double\*)malloc(n \* sizeof(double));  if(x\_real == NULL) {  printf("Memory allocation failed.\n");  return 1;  }  x\_real[0] = 2;  x\_real[1] = 1;  x\_real[2] = -1;      // 输出迭代过程  printf("--------------------------------------------\n");  printf("inter\tx1\t\tx2\t\tx3\t\tL\_infinity\tL\_2\_error\n");  printf("--------------------------------------------\n");  printf("%d\t%lf\t%lf\t%lf\t%lf\t%s\n", 0, x[0][0], x[0][1], x[0][2],max(fabs(x[0][0]-x\_real[0]),fabs(x[0][1]-x\_real[1]),fabs(x[0][2])-x\_real[2]), "initial error");  for (int i = 1; i <= iter; i++){  printf("%d\t%lf\t%lf\t%lf\t%lf\t%lf\n", i, x[i][0], x[i][1], x[i][2],max(fabs(x[i][0]-x\_real[0]),fabs(x[i][1]-x\_real[1]),fabs(x[i][2])-x\_real[2]), diff[i]);  }  printf("--------------------------------------------\n");    // 输出结果  if (result == 1) {  printf("The result was found within %d iterations.\n", iter);  } else {  //告诉用户结果未收敛  printf("The result was not found within %d iterations.\n", max\_iter);    // 输出绝对误差(需要知道真正结果)  printf("The absolute error is (%lf,%lf,%lf)'.\n",absolute\_error(x[iter][0],1),absolute\_error(x[iter][1],1),absolute\_error(x[iter][2],1));  // 输出x带入方程计算的L2范数误差（不需要知道真正结果）  double \*x\_dairu = (double\*)malloc(n \* sizeof(double));  if(x\_dairu == NULL) {  printf("Memory allocation failed.\n");  return 1;  }  for (int i = 0; i < n; i++) {  for (int j = 0; j < n; j++) {  x\_dairu[i] += A[i][j] \* x[max\_iter][j];  }  x\_dairu[i] -= b[i];  }  double error\_norm = 0.0;  for (int i = 0; i < n; i++) {  error\_norm += (x\_dairu[i] \* x\_dairu[i]);  }  error\_norm = sqrt(error\_norm);  printf("The L2 norm error when calculate result brought into the equation is %lf.\n", error\_norm);  }    // 释放内存      free(D);  free(x);    free(diff);  return 0;  }    实验3.1：（SOR松弛迭代法求解线性方程组2）  #include <stdio.h>  #include <stdlib.h>  #include <math.h>  #include "error.hpp"  double max(double a, double b, double c) {  return a > b? (a > c? a : c) : (b > c? b : c);  }      int main(void) {  // 输入矩阵大小n  /\* int n;  printf("Enter the size of the matrix: ");  scanf("%d", &n); \*/  int n=3;  /\* //输入最大迭代次数  int max\_iter;  printf("Enter the maximum number of iterations: ");  scanf("%d", &max\_iter); \*/    /\* // 输入A  double \*\*A = NULL;  A = (double \*\*)malloc(n \* sizeof(double \*));  for (int i = 0; i < n; i++) {  A[i] = (double \*)malloc(n \* sizeof(double));  }  if(A == NULL) {  printf("Memory allocation failed.\n");  return 1;  }  printf("Enter the elements of the matrix:\n");  for (int i = 0; i < n; i++) {  for (int j = 0; j < n; j++) {  scanf("%lf", &A[i][j]);  }  printf("\n");  }    // 输入b  printf("Enter the number of iterations:\n");  double\* b = (double\*)malloc(n \* sizeof(double));  if(b == NULL) {  printf("Memory allocation failed.\n");  return 1;  }  for (int i = 0; i < n; i++) {  scanf("%lf", &b[i]);  }  \*/  // 进行Jacobi迭代  double A[3][3] = {{10.0, 3.0, 1.0}, {2.0, -10.0, 3.0}, {1.0, 3.0, 10.0}};  double b[3] = {14.0, -5.0, 14.0};  //计算D矩阵  double\*\* D = (double\*\*)malloc(n \* sizeof(double\*));  for (int i = 0; i < n; i++) {  D[i] = (double\*)malloc(n \* sizeof(double));  }  if(D == NULL) {  printf("Memory allocation failed.\n");  return 1;  }  for (int i = 0; i < n; i++){  for (int j = 0; j < n; j++) {  if (i == j) {  D[i][j] = A[i][j];  } else {  D[i][j] = 0.0;  }  }  }  /\* //计算L矩阵  double\*\* L = (double\*\*)malloc(n \* sizeof(double\*));  for (int i = 0; i < n; i++) {  L[i] = (double\*)malloc(n \* sizeof(double));  }  if(L == NULL) {  printf("Memory allocation failed.\n");  return 1;  }  for (int i = 0; i < n; i++) {  for (int j = 0; j < n; j++) {  if (i == j) {  L[i][j] = 0.0;  } else if (j > i) {  L[i][j] = -A[i][j];  } else {  L[i][j] = 0.0;  }  }  }    //计算U矩阵  double\*\* U = (double\*\*)malloc(n \* sizeof(double\*));  for (int i = 0; i < n; i++) {  U[i] = (double\*)malloc(n \* sizeof(double));  }  if(U == NULL) {  printf("Memory allocation failed.\n");  return 1;  }  for (int i = 0; i < n; i++) {  for (int j = 0; j < n; j++) {  if (i == j) {  U[i][j] = 0.0;  } else if (j < i) {  U[i][j] = -A[i][j];  } else {  U[i][j] = 0.0;  }  }  } \*/  /\* // 输出矩阵A  printf("The matrix A is:\n");  for (int i = 0; i < n; i++) {  for (int j = 0; j < n; j++) {  printf("%lf\t", A[i][j]);  }  printf("\n");  }    // 输出矩阵D  printf("The matrix D is:\n");  for (int i = 0; i < n; i++) {  for (int j = 0; j < n; j++) {  printf("%lf\t", D[i][j]);  }  printf("\n");  }  // 输出矩阵b  printf("The vector b is:\n");  for (int i = 0; i < n; i++) {  printf("%lf\t", b[i]);  }  printf("\n");  \*/  //设置数列来存储n维向量的n个分量迭代结果    // 定义最大迭代次数  int max\_iter = 50;  double\*\* x = (double\*\*)malloc((max\_iter+1) \* sizeof(double\*));  for (int i = 0; i < (max\_iter+1); i++) {  x[i] = (double\*)malloc(n \* sizeof(double));  }  if(x == NULL) {  printf("Memory allocation failed.\n");  return 1;  }  // 初始化n维向量的n个分量为0'  for (int i = 0; i < n; i++) {  x[0][i] = 0.0;  }  //存储L2范数的数组  double\* diff = (double\*)malloc((max\_iter+1) \* sizeof(double));  if(diff == NULL) {  printf("Memory allocation failed.\n");  return 1;  }  // 定义初始的L2范数  diff[0] = 9999;  // 开始迭代  int iter = 1;  double eps = 1e-5;    // 开始迭代  // 定义结果判断是否收敛  double omiga = 1.46;  int result = 0;  while (iter <= max\_iter) {  for (int i = 0; i < n; i++) {  double sum = 0.0;  for (int j = 0; j < n; j++) {  if (j != i) {  if (j < i) {  sum += A[i][j] \* x[iter][j];  } else {  sum += A[i][j] \* x[iter-1][j];  }  }  }  x[iter][i] = (((b[i]-sum) / D[i][i] ) - x[iter-1][i]) \* omiga + x[iter-1][i];  }    // L2范数计算  diff[iter] = 0.0;  for (int i = 0; i < n; i++) {  diff[iter] += (x[iter][i] - x[iter-1][i])\*(x[iter][i] - x[iter-1][i]);  }  diff[iter] = sqrt(diff[iter]);      // 输出当前迭代结果  printf("Iteration %d:\n", iter);  printf("x = [");  for (int i = 0; i < n; i++) {  printf("%lf ", x[iter][i]);  }  printf("]\n");  printf("L2\_error = %lf\n", diff[iter]);  // 判断是否收敛  if (diff[iter] < eps) {  result = 1;  break;  }  iter++;  }  if(iter > max\_iter) {  iter--;  }  printf("----------\n");    double\* x\_real = (double\*)malloc(n \* sizeof(double));  if(x\_real == NULL) {  printf("Memory allocation failed.\n");  return 1;  }  x\_real[0] = 1;  x\_real[1] = 1;  x\_real[2] = 1;      // 输出迭代过程  printf("--------------------------------------------\n");  printf("inter\tx1\t\tx2\t\tx3\t\tL\_infinity\tL\_2\_error\n");  printf("--------------------------------------------\n");  printf("%d\t%lf\t%lf\t%lf\t%lf\t%s\n", 0, x[0][0], x[0][1], x[0][2],max(fabs(x[0][0]-x\_real[0]),fabs(x[0][1]-x\_real[1]),fabs(x[0][2])-x\_real[2]), "initial error");  for (int i = 1; i <= iter; i++){  printf("%d\t%lf\t%lf\t%lf\t%lf\t%lf\n", i, x[i][0], x[i][1], x[i][2],max(fabs(x[i][0]-1),fabs(x[i][1]-1),fabs(x[i][2])-1), diff[i]);  }  printf("--------------------------------------------\n");    // 输出结果  if (result == 1) {  printf("The result was found within %d iterations.\n", iter);  } else {  //告诉用户结果未收敛  printf("The result was not found within %d iterations.\n", max\_iter);    // 输出绝对误差(需要知道真正结果)  printf("The absolute error is (%lf,%lf,%lf)'.\n",absolute\_error(x[iter][0],1),absolute\_error(x[iter][1],1),absolute\_error(x[iter][2],1));  // 输出x带入方程计算的L2范数误差（不需要知道真正结果）  double \*x\_dairu = (double\*)malloc(n \* sizeof(double));  if(x\_dairu == NULL) {  printf("Memory allocation failed.\n");  return 1;  }  for (int i = 0; i < n; i++) {  for (int j = 0; j < n; j++) {  x\_dairu[i] += A[i][j] \* x[max\_iter][j];  }  x\_dairu[i] -= b[i];  }  double error\_norm = 0.0;  for (int i = 0; i < n; i++) {  error\_norm += (x\_dairu[i] \* x\_dairu[i]);  }  error\_norm = sqrt(error\_norm);  printf("The L2 norm error when calculate result brought into the equation is %lf.\n", error\_norm);  }    // 释放内存      free(D);  free(x);    free(diff);  return 0;  }      实验3.2：（SOR松弛迭代法求解线性方程组3）  #include <stdio.h>  #include <stdlib.h>  #include <math.h>  #include "error.hpp"  double max(double a, double b, double c) {  return a > b? (a > c? a : c) : (b > c? b : c);  }      int main(void) {  // 输入矩阵大小n  /\* int n;  printf("Enter the size of the matrix: ");  scanf("%d", &n); \*/  int n=3;  /\* //输入最大迭代次数  int max\_iter;  printf("Enter the maximum number of iterations: ");  scanf("%d", &max\_iter); \*/    /\* // 输入A  double \*\*A = NULL;  A = (double \*\*)malloc(n \* sizeof(double \*));  for (int i = 0; i < n; i++) {  A[i] = (double \*)malloc(n \* sizeof(double));  }  if(A == NULL) {  printf("Memory allocation failed.\n");  return 1;  }  printf("Enter the elements of the matrix:\n");  for (int i = 0; i < n; i++) {  for (int j = 0; j < n; j++) {  scanf("%lf", &A[i][j]);  }  printf("\n");  }    // 输入b  printf("Enter the number of iterations:\n");  double\* b = (double\*)malloc(n \* sizeof(double));  if(b == NULL) {  printf("Memory allocation failed.\n");  return 1;  }  for (int i = 0; i < n; i++) {  scanf("%lf", &b[i]);  }  \*/  // 进行Jacobi迭代  double A[3][3] = {{4.0, -2.0, -4.0}, {-2.0, 17.0, 10.0}, {-4.0, 10.0, 9.0}};  double b[3] = {10.0, 3.0, -7.0};  //计算D矩阵  double\*\* D = (double\*\*)malloc(n \* sizeof(double\*));  for (int i = 0; i < n; i++) {  D[i] = (double\*)malloc(n \* sizeof(double));  }  if(D == NULL) {  printf("Memory allocation failed.\n");  return 1;  }  for (int i = 0; i < n; i++){  for (int j = 0; j < n; j++) {  if (i == j) {  D[i][j] = A[i][j];  } else {  D[i][j] = 0.0;  }  }  }  /\* //计算L矩阵  double\*\* L = (double\*\*)malloc(n \* sizeof(double\*));  for (int i = 0; i < n; i++) {  L[i] = (double\*)malloc(n \* sizeof(double));  }  if(L == NULL) {  printf("Memory allocation failed.\n");  return 1;  }  for (int i = 0; i < n; i++) {  for (int j = 0; j < n; j++) {  if (i == j) {  L[i][j] = 0.0;  } else if (j > i) {  L[i][j] = -A[i][j];  } else {  L[i][j] = 0.0;  }  }  }    //计算U矩阵  double\*\* U = (double\*\*)malloc(n \* sizeof(double\*));  for (int i = 0; i < n; i++) {  U[i] = (double\*)malloc(n \* sizeof(double));  }  if(U == NULL) {  printf("Memory allocation failed.\n");  return 1;  }  for (int i = 0; i < n; i++) {  for (int j = 0; j < n; j++) {  if (i == j) {  U[i][j] = 0.0;  } else if (j < i) {  U[i][j] = -A[i][j];  } else {  U[i][j] = 0.0;  }  }  } \*/  /\* // 输出矩阵A  printf("The matrix A is:\n");  for (int i = 0; i < n; i++) {  for (int j = 0; j < n; j++) {  printf("%lf\t", A[i][j]);  }  printf("\n");  }    // 输出矩阵D  printf("The matrix D is:\n");  for (int i = 0; i < n; i++) {  for (int j = 0; j < n; j++) {  printf("%lf\t", D[i][j]);  }  printf("\n");  }  // 输出矩阵b  printf("The vector b is:\n");  for (int i = 0; i < n; i++) {  printf("%lf\t", b[i]);  }  printf("\n");  \*/  //设置数列来存储n维向量的n个分量迭代结果    // 定义最大迭代次数  int max\_iter = 50;  double\*\* x = (double\*\*)malloc((max\_iter+1) \* sizeof(double\*));  for (int i = 0; i < (max\_iter+1); i++) {  x[i] = (double\*)malloc(n \* sizeof(double));  }  if(x == NULL) {  printf("Memory allocation failed.\n");  return 1;  }  // 初始化n维向量的n个分量为0'  for (int i = 0; i < n; i++) {  x[0][i] = 0.0;  }  //存储L2范数的数组  double\* diff = (double\*)malloc((max\_iter+1) \* sizeof(double));  if(diff == NULL) {  printf("Memory allocation failed.\n");  return 1;  }  // 定义初始的L2范数  diff[0] = 9999;  // 开始迭代  int iter = 1;  double eps = 1e-5;    // 开始迭代  // 定义结果判断是否收敛  double omiga = 1.46;  int result = 0;  while (iter <= max\_iter) {  for (int i = 0; i < n; i++) {  double sum = 0.0;  for (int j = 0; j < n; j++) {  if (j != i) {  if (j < i) {  sum += A[i][j] \* x[iter][j];  } else {  sum += A[i][j] \* x[iter-1][j];  }  }  }  x[iter][i] = (((b[i]-sum) / D[i][i] ) - x[iter-1][i]) \* omiga + x[iter-1][i];  }    // L2范数计算  diff[iter] = 0.0;  for (int i = 0; i < n; i++) {  diff[iter] += (x[iter][i] - x[iter-1][i])\*(x[iter][i] - x[iter-1][i]);  }  diff[iter] = sqrt(diff[iter]);      // 输出当前迭代结果  printf("Iteration %d:\n", iter);  printf("x = [");  for (int i = 0; i < n; i++) {  printf("%lf ", x[iter][i]);  }  printf("]\n");  printf("L2\_error = %lf\n", diff[iter]);  // 判断是否收敛  if (diff[iter] < eps) {  result = 1;  break;  }  iter++;  }  if(iter > max\_iter) {  iter--;  }  printf("----------\n");    double\* x\_real = (double\*)malloc(n \* sizeof(double));  if(x\_real == NULL) {  printf("Memory allocation failed.\n");  return 1;  }  x\_real[0] = 2;  x\_real[1] = 1;  x\_real[2] = -1;      // 输出迭代过程  printf("--------------------------------------------\n");  printf("inter\tx1\t\tx2\t\tx3\t\tL\_infinity\tL\_2\_error\n");  printf("--------------------------------------------\n");  printf("%d\t%lf\t%lf\t%lf\t%lf\t%s\n", 0, x[0][0], x[0][1], x[0][2],max(fabs(x[0][0]-x\_real[0]),fabs(x[0][1]-x\_real[1]),fabs(x[0][2])-x\_real[2]), "initial error");  for (int i = 1; i <= iter; i++){  printf("%d\t%lf\t%lf\t%lf\t%lf\t%lf\n", i, x[i][0], x[i][1], x[i][2],max(fabs(x[i][0]-x\_real[0]),fabs(x[i][1]-x\_real[1]),fabs(x[i][2])-x\_real[2]), diff[i]);  }  printf("--------------------------------------------\n");    // 输出结果  if (result == 1) {  printf("The result was found within %d iterations.\n", iter);  } else {  //告诉用户结果未收敛  printf("The result was not found within %d iterations.\n", max\_iter);    // 输出绝对误差(需要知道真正结果)  printf("The absolute error is (%lf,%lf,%lf)'.\n",absolute\_error(x[iter][0],1),absolute\_error(x[iter][1],1),absolute\_error(x[iter][2],1));  // 输出x带入方程计算的L2范数误差（不需要知道真正结果）  double \*x\_dairu = (double\*)malloc(n \* sizeof(double));  if(x\_dairu == NULL) {  printf("Memory allocation failed.\n");  return 1;  }  for (int i = 0; i < n; i++) {  for (int j = 0; j < n; j++) {  x\_dairu[i] += A[i][j] \* x[max\_iter][j];  }  x\_dairu[i] -= b[i];  }  double error\_norm = 0.0;  for (int i = 0; i < n; i++) {  error\_norm += (x\_dairu[i] \* x\_dairu[i]);  }  error\_norm = sqrt(error\_norm);  printf("The L2 norm error when calculate result brought into the equation is %lf.\n", error\_norm);  }    // 释放内存      free(D);  free(x);  free(diff);  return 0;  }    实验4.1：使用迭代法对两点边值问题的有限差分求解  #include <stdio.h>  #include <stdlib.h>  #include <math.h>  #include <time.h>  void jisuan\_zhijie(int n, double \* er\_1, double \* er\_2, double \* time\_used) {  clock\_t start, end;    start = clock();  double pi = 3.14159265358979323846;  /\* int n;  printf("Enter the size of the matrix: ");  scanf("%d", &n); \*/    double h = (double)1.0 / n;  n=n+1;  // 输入矩阵A和b  double \*\*A = NULL;      A = (double \*\*)malloc(n \* sizeof(double \*));  for (int i = 0; i < n; i++) {  A[i] = (double \*)malloc(n \* sizeof(double));  }  if(A == NULL) {  printf("Memory allocation failed.\n");  exit(1);  }        for (int i = 0; i < n; i++) {  for (int j = 0; j < n; j++) {  A[i][j] = 0;  }  }  A[0][0]=1;  A[n-1][n-1] = 1;  for (int i = 1; i < n-1; i++) {  A[i][i-1] = (double)(2.0) + h\*(1.0+i\*h)\*(1.0+i\*h);  A[i][i] = -1\*(double)4.0 - 2\*h\*h \* exp(-1\*i\*h);  A[i][i+1] = 2.0 - h\*(1+i\*h)\*(1+i\*h);  }  double\* b = (double\*)malloc(n \* sizeof(double));  if(b == NULL) {  printf("Memory allocation failed.\n");  exit(1);  }  for (int i = 0; i < n; i++) {  b[i] = 2 \*h\*h\*( (1-(1+i\*h)\*(1+i\*h))\*exp(i\*h) - pi\*pi\*cos(pi\*i\*h) + pi\*(1+i\*h)\*(1+i\*h)\*sin(pi\*i\*h) - 1 -exp(-1\*i\*h)\*cos(pi\*i\*h) );  }  b[0] = 2;  b[n-1] = exp(1)-1;    //列主元Gauss消去法  for (int i = 0; i < n; i++) {  int max\_index = i;  for (int j = i + 1; j < n; j++) {//找列最大值  if (fabs(A[j][i]) > fabs(A[max\_index][i])) {  max\_index = j;  }  }  if (A[max\_index][i] == 0) {//如果最大值为0，则矩阵为奇异矩阵  printf("The matrix is singular.\n");    }  if (max\_index != i) {//交换两行  for(int j = i; j < n; j++) {  double temp = A[i][j];  A[i][j] = A[max\_index][j];  A[max\_index][j] = temp;  }  double temp\_b = b[i];  b[i] = b[max\_index];  b[max\_index] = temp\_b;  }  for (int j = i + 1; j < n; j++) {//消去法  double factor = A[j][i] / A[i][i];  A[j][i] = 0;  for (int k = i + 1; k < n; k++) {  A[j][k] -= factor \* A[i][k];  }  b[j] -= factor \* b[i];  }  }    /\* / 输出变换后的A和b  for (int i = 0; i < n; i++) {  for (int j = 0; j < n; j++) {  printf("%lf ", A[i][j]);  }  printf("\n");  }  printf("\n");  for(int i = 0; i < n; i++) {  printf("%lf ", b[i]);  }  printf("\n"); \*/  double\* x = (double\*)malloc(n \* sizeof(double));  if(x == NULL) {  printf("Memory allocation failed.\n");  exit(1);  }  // 解线性方程组  for (int i = n - 1; i >= 0; i--) {  double ad = b[i];  for (int j = n-1; j >= i+1; j--) {  ad -= A[i][j] \* x[j];  }  x[i] = ad/A[i][i];  }  /\* // 输出解  printf("The solution is:\n");  for(int i = 0; i < n; i++) {  printf("%lf\n", x[i]);  }  printf("\n"); \*/    n=n-1;  // 计算误差  double \* u = (double\*)malloc((n+1) \* sizeof(double));  if(u == NULL) {  printf("Memory allocation failed.\n");  exit(1);  }  for (int i = 0; i < n+1; i++) {  u[i] = exp(i\*h) + cos(pi\*i\*h);  }  double err\_1 = fabs(x[1] - u[1]);  for (int i = 2; i < n; i++) {  err\_1 = err\_1 > fabs(x[i] - u[i]) ? err\_1 : fabs(x[i] - u[i]);  }  double err\_2 = 0;  for (int i = 1; i < n; i++) {  err\_2 += h\*(x[i] - u[i])\*(x[i] - u[i]);  }  err\_2 = sqrt(err\_2);  \*er\_2 = err\_2;  \*er\_1 = err\_1;  // 计算运行时间  end = clock();  \*time\_used = (double)(end - start) / CLOCKS\_PER\_SEC;  // 释放内存  for (int i = 0; i < n; i++) {  free(A[i]);  }  free(A);  free(b);  free(x);  free(u);  }  void jisuan\_diedai(int n, double \* er\_1, double \* er\_2, double \* time\_used) {  clock\_t start, end;  start = clock();  // 输入矩阵A和b  double pi = 3.14159265358979323846;  /\* int n;  printf("Enter the size of the matrix: ");  scanf("%d", &n); \*/    double h = (double)1.0 / n;  n=n+1;  // 输入矩阵A和b  double \*\*A = NULL;      A = (double \*\*)malloc(n \* sizeof(double \*));  for (int i = 0; i < n; i++) {  A[i] = (double \*)malloc(n \* sizeof(double));  }  if(A == NULL) {  printf("Memory allocation failed.\n");  exit(1);  }        for (int i = 0; i < n; i++) {  for (int j = 0; j < n; j++) {  A[i][j] = 0;  }  }  A[0][0]=1;  A[n-1][n-1] = 1;  for (int i = 1; i < n-1; i++) {  A[i][i-1] = (double)(2.0) + h\*(1.0+i\*h)\*(1.0+i\*h);  A[i][i] = -1\*(double)4.0 - 2\*h\*h \* exp(-1\*i\*h);  A[i][i+1] = 2.0 - h\*(1+i\*h)\*(1+i\*h);  }  double\* b = (double\*)malloc(n \* sizeof(double));  if(b == NULL) {  printf("Memory allocation failed.\n");  exit(1);  }  for (int i = 0; i < n; i++) {  b[i] = 2 \*h\*h\*( (1-(1+i\*h)\*(1+i\*h))\*exp(i\*h) - pi\*pi\*cos(pi\*i\*h) + pi\*(1+i\*h)\*(1+i\*h)\*sin(pi\*i\*h) - 1 -exp(-1\*i\*h)\*cos(pi\*i\*h) );  }  b[0] = 2;  b[n-1] = exp(1)-1;  // 迭代法求解  //设置数列来存储n维向量的n个分量迭代结果    // 定义最大迭代次数  int max\_iter = 999999;  double\*\* x = (double\*\*)malloc((max\_iter+1) \* sizeof(double\*));  for (int i = 0; i < (max\_iter+1); i++) {  x[i] = (double\*)malloc(n \* sizeof(double));  }  if(x == NULL) {  printf("Memory allocation failed.\n");  exit(1);  }  // 初始化n维向量的n个分量为0'  for (int i = 0; i < n; i++) {  x[0][i] = 0.0;  }  //存储L2范数的数组  double\* diff = (double\*)malloc((max\_iter+1) \* sizeof(double));  if(diff == NULL) {  printf("Memory allocation failed.\n");  exit(1);  }  // 定义初始的L2范数  diff[0] = 9999;  // 开始迭代  int iter = 1;  double eps = 1e-7;    // 开始迭代  // 定义结果判断是否收敛  double omiga = 1.959;  int result = 0;  while (iter <= max\_iter) {  for (int i = 0; i < n; i++) {  double sum = 0.0;  for (int j = 0; j < n; j++) {  if (j != i) {  if (j < i) {  sum += A[i][j] \* x[iter][j];  } else {  sum += A[i][j] \* x[iter-1][j];  }  }  }  x[iter][i] = (((b[i]-sum) / A[i][i] ) - x[iter-1][i]) \* omiga + x[iter-1][i];  }    // L2范数计算  diff[iter] = 0.0;  for (int i = 0; i < n; i++) {  diff[iter] += (x[iter][i] - x[iter-1][i])\*(x[iter][i] - x[iter-1][i]);  }  diff[iter] = sqrt(diff[iter]);    // 判断是否收敛  if (diff[iter] < eps) {  result = 1;  break;  }  iter++;  }  if(iter > max\_iter) {  iter--;  }  n=n-1;  // 计算误差  double \* u = (double\*)malloc((n+1) \* sizeof(double));  if(u == NULL) {  printf("Memory allocation failed.\n");  exit(1);  }  for (int i = 0; i < n+1; i++) {  u[i] = exp(i\*h) + cos(pi\*i\*h);  }    double err\_1 = fabs(x[iter][1] - u[1]);  for (int i = 2; i < n; i++) {  err\_1 = err\_1 > fabs(x[iter][i] - u[i]) ? err\_1 : fabs(x[iter][i] - u[i]);  }  double err\_2 = 0;  for (int i = 1; i < n; i++) {  err\_2 += h\*(x[iter][i] - u[i])\*(x[iter][i] - u[i]);  }  err\_2 = sqrt(err\_2);  \*er\_2 = err\_2;  \*er\_1 = err\_1;  // 计算运行时间  end = clock();  \*time\_used = (double)(end - start) / CLOCKS\_PER\_SEC;  // 释放内存  for (int i = 0; i < (max\_iter+1); i++) {  free(x[i]);  }  free(x);  free(diff);  for (int i = 0; i < n; i++) {  free(A[i]);  }  free(A);  free(b);  free(u);  }  int main(void) {  int n1 = 10;  int n2 = 20;  int n3 = 40;  int n4 = 80;  int n5 = 160;  double err\_1\_1 = 0;  double err\_2\_1 = 0;  double err\_1\_2 = 0;  double err\_2\_2 = 0;  double err\_1\_3 = 0;  double err\_2\_3 = 0;  double err\_1\_4 = 0;  double err\_2\_4 = 0;  double err\_1\_5 = 0;  double err\_2\_5 = 0;  double time\_used\_1 = 0;  double time\_used\_2 = 0;  double time\_used\_3 = 0;  double time\_used\_4 = 0;  double time\_used\_5 = 0;    jisuan\_zhijie(n1, &err\_1\_1, &err\_2\_1, &time\_used\_1);  jisuan\_zhijie(n2, &err\_1\_2, &err\_2\_2, &time\_used\_2);  jisuan\_zhijie(n3, &err\_1\_3, &err\_2\_3, &time\_used\_3);  jisuan\_zhijie(n4, &err\_1\_4, &err\_2\_4, &time\_used\_4);  jisuan\_zhijie(n5, &err\_1\_5, &err\_2\_5, &time\_used\_5);  printf("Direct solution method\n");  printf("----------\n");  printf("index\tn\terror\_1\t\terror\_2\t\ttime\_used\n");  printf("--------------------------\n");    printf("n1\t10\t%lf\t%lf\t%lf\n", err\_1\_1, err\_2\_1, time\_used\_1);  printf("n2\t20\t%lf\t%lf\t%lf\n", err\_1\_2, err\_2\_2, time\_used\_2);  printf("n3\t40\t%lf\t%lf\t%lf\n", err\_1\_3, err\_2\_3, time\_used\_3);  printf("n4\t80\t%lf\t%lf\t%lf\n", err\_1\_4, err\_2\_4, time\_used\_4);  printf("n5\t160\t%lf\t%lf\t%lf\n", err\_1\_5, err\_2\_5, time\_used\_5);  printf("--------------------------\n");  printf("\n");    jisuan\_diedai(n1, &err\_1\_1, &err\_2\_1, &time\_used\_1);  jisuan\_diedai(n2, &err\_1\_2, &err\_2\_2, &time\_used\_2);  jisuan\_diedai(n3, &err\_1\_3, &err\_2\_3, &time\_used\_3);  jisuan\_diedai(n4, &err\_1\_4, &err\_2\_4, &time\_used\_4);  jisuan\_diedai(n5, &err\_1\_5, &err\_2\_5, &time\_used\_5);  printf("Iterative solution method\n");  printf("----------\n");  printf("index\tn\terror\_1\t\terror\_2\t\ttime\_used\n");  printf("--------------------------\n");    printf("n1\t10\t%lf\t%lf\t%lf\n", err\_1\_1, err\_2\_1, time\_used\_1);  printf("n2\t20\t%lf\t%lf\t%lf\n", err\_1\_2, err\_2\_2, time\_used\_2);  printf("n3\t40\t%lf\t%lf\t%lf\n", err\_1\_3, err\_2\_3, time\_used\_3);  printf("n4\t80\t%lf\t%lf\t%lf\n", err\_1\_4, err\_2\_4, time\_used\_4);  printf("n5\t160\t%lf\t%lf\t%lf\n", err\_1\_5, err\_2\_5, time\_used\_5);  printf("----------\n");  return 0;  } | | | |
| 实验分析与总结：  经过本次实验，了解到了误差产生的原因以及为什么要避免误差，如何避免误差。  强化了编程能力，学会了如何使用远程服务器辅助完成代码的运行。  通过实验1.1与1.2，我了解到了Jacobi迭代法的原理以及适用范围：严格对角占优时才可使用  通过实验2.1与2.2，我了解到了Gauss-Seidel迭代法，他是对Jacobi迭代法的一种优化  通过实验3.1与3.2，我了解到了SOR松弛迭代法，他的求解效率很大程度上由 值的设定来决定  通过实验4.1，我比较了直接求解法与迭代求解法的求解速度，发现在两点边值问题下直接求解法速度要高于迭代求解法 | | | |

注：若填写内容较多，可在背面继续填写。