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：实验二： 线性方程组的直接求解 | |
| 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： 2024/7/2 | 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； 学会服务器上的一些linux基本命令；熟悉求解线性方程组的顺序Gauss消去法、列主元Gauss消去法和LU分解法；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：（顺序Gauss消去法求解线性方程组1）  针对方程组    采用顺序Gauss消去法，以浮点计算方式进行求解。  实验1.2：（顺序Gauss消去法求解线性方程组2）  针对方程组    采用顺序Gauss消去法，以浮点计算方式进行求解。  实验1.3：（顺序Gauss消去法求解线性方程组3）  针对方程组    采用顺序Gauss消去法，以浮点计算方式进行求解。  实验2.1：（列主元顺序Gauss消去法求解线性方程组1）  针对方程组    采用列主元Gauss消去法，以浮点计算方式进行求解。  实验2.2：（列主元顺序Gauss消去法求解线性方程组2）  针对方程组    采用列主元Gauss消去法，以浮点计算方式进行求解。  实验2.3：（列主元顺序Gauss消去法求解线性方程组3）  针对方程组    采用列主元Gauss消去法，以浮点计算方式进行求解。  实验3.1：（Doolittle三角分解法（LU分解）求解线性方程组1）  针对方程组    采用Doolittle三角分解法，以浮点计算方式进行求解。  实验3.2：（Doolittle三角分解法（LU分解）求解线性方程组2）  针对方程组    采用Doolittle三角分解法，以浮点计算方式进行求解。  实验3.3：（Doolittle三角分解法（LU分解）求解线性方程组3）  针对方程组    采用Doolittle三角分解法，以浮点计算方式进行求解。  实验4.1：两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，。 | | | |
| 实验过程与结果：  实验1.1：（顺序Gauss消去法求解线性方程组1）  #include <stdio.h>  #include <stdlib.h>  int main(void) {  float \*\*A = NULL;  int n = 2;  A = (float \*\*)malloc(n \* sizeof(float \*));  if(A == NULL) {  printf("Memory allocation failed.\n");  return 1;  }  for (int i = 0; i < n; i++) {  A[i] = (float \*)malloc(n \* sizeof(float));  }  A[0][0] = 0.000000010;A[0][1] = 1.00;  A[1][0] = 1.00;A[1][1] = 1.00;  float\* b = (float\*)malloc(n \* sizeof(float));  b[0] = 1.00;  b[1] = 2.00;  /\* printf("Enter the size of the matrix: ");  scanf("%d", &n);    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");  }      printf("Enter the number of iterations:\n");  double\* b = (double\*)malloc(n \* sizeof(double));  for (int i = 0; i < n; i++) {  scanf("%lf", &b[i]);  } \*/  // 顺序gauss消去法  for (int i = 0; i < n; i++) {  for (int j = i + 1; j < n; j++) {  float 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  printf("The transformed matrix is:\n");  for (int i = 0; i < n; i++) {  for (int j = 0; j < n; j++) {  printf("%f ", A[i][j]);  }  printf("\n");  }  printf("The transformed vector is:\n");  printf("\n");  for(int i = 0; i < n; i++) {  printf("%f ", b[i]);  }  printf("\n");  float\* x = (float\*)malloc(n \* sizeof(float));  // 解线性方程组  for (int i = n - 1; i >= 0; i--) {  float 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("%f\n", x[i]);  }  printf("\n");  for (int i = 0; i < n; i++) {  free(A[i]);  }  free(A);  free(b);  return 0;  }  为了达到实验目的，展示顺序gauss消去法所带来的误差问题，将原题目中的0.0000010改为0.000000010    实验1.2：（顺序Gauss消去法求解线性方程组2）  #include <stdio.h>  #include <stdlib.h>  int main(void) {  float \*\*A = NULL;  int n = 3;  A = (float \*\*)malloc(n \* sizeof(float \*));  if(A == NULL) {  printf("Memory allocation failed.\n");  return 1;  }  for (int i = 0; i < n; i++) {  A[i] = (float \*)malloc(n \* sizeof(float));  }  A[0][0] = 2;A[0][1] = 4;A[0][2] = -2;  A[1][0] = 1;A[1][1] = -3;A[1][2] = -3;  A[2][0] = 4;A[2][1] = 2;A[2][2] = 2;  float\* b = (float\*)malloc(n \* sizeof(float));  b[0] = 2;  b[1] = -1;  b[2] = 3;  /\* printf("Enter the size of the matrix: ");  scanf("%d", &n);    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");  }      printf("Enter the number of iterations:\n");  double\* b = (double\*)malloc(n \* sizeof(double));  for (int i = 0; i < n; i++) {  scanf("%lf", &b[i]);  } \*/  // 顺序gauss消去法  for (int i = 0; i < n; i++) {  for (int j = i + 1; j < n; j++) {  float 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  printf("The transformed matrix is:\n");  for (int i = 0; i < n; i++) {  for (int j = 0; j < n; j++) {  printf("%f ", A[i][j]);  }  printf("\n");  }  printf("The transformed vector is:\n");  printf("\n");  for(int i = 0; i < n; i++) {  printf("%f ", b[i]);  }  printf("\n");  float\* x = (float\*)malloc(n \* sizeof(float));  // 解线性方程组  for (int i = n - 1; i >= 0; i--) {  float 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("%f\n", x[i]);  }  printf("\n");  for (int i = 0; i < n; i++) {  free(A[i]);  }  free(A);  free(b);  return 0;  }    实验1.3：（顺序Gauss消去法求解线性方程组3）  #include <stdio.h>  #include <stdlib.h>  int main(void) {  float \*\*A = NULL;  int n = 3;  A = (float \*\*)malloc(n \* sizeof(float \*));  if(A == NULL) {  printf("Memory allocation failed.\n");  return 1;  }  for (int i = 0; i < n; i++) {  A[i] = (float \*)malloc(n \* sizeof(float));  }  A[0][0] = 0.012;A[0][1] = 0.01;A[0][2] = 0.167;  A[1][0] = 1;A[1][1] = 0.8334;A[1][2] = 5.91;  A[2][0] = 3200;A[2][1] = 1200;A[2][2] = 4.2;  float\* b = (float\*)malloc(n \* sizeof(float));  b[0] = 0.6781;  b[1] = 12.1;  b[2] = 981;  /\* printf("Enter the size of the matrix: ");  scanf("%d", &n);    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");  }      printf("Enter the number of iterations:\n");  double\* b = (double\*)malloc(n \* sizeof(double));  for (int i = 0; i < n; i++) {  scanf("%lf", &b[i]);  } \*/  // 顺序gauss消去法  for (int i = 0; i < n; i++) {  for (int j = i + 1; j < n; j++) {  float 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  printf("The transformed matrix is:\n");  for (int i = 0; i < n; i++) {  for (int j = 0; j < n; j++) {  printf("%f ", A[i][j]);  }  printf("\n");  }  printf("The transformed vector is:\n");  printf("\n");  for(int i = 0; i < n; i++) {  printf("%f ", b[i]);  }  printf("\n");  float\* x = (float\*)malloc(n \* sizeof(float));  // 解线性方程组  for (int i = n - 1; i >= 0; i--) {  float 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("%f\n", x[i]);  }  printf("\n");  for (int i = 0; i < n; i++) {  free(A[i]);  }  free(A);  free(b);  return 0;  }    实验2.1：（列主元顺序Gauss消去法求解线性方程组1）  #include <stdio.h>  #include <stdlib.h>  #include <math.h>  int main(void) {  float \*\*A = NULL;  int n = 2;  A = (float \*\*)malloc(n \* sizeof(float \*));  if(A == NULL) {  printf("Memory allocation failed.\n");  return 1;  }  for (int i = 0; i < n; i++) {  A[i] = (float \*)malloc(n \* sizeof(float));  }  A[0][0] = 0.000000010;A[0][1] = 1.00;  A[1][0] = 1.00;A[1][1] = 1.00;  float\* b = (float\*)malloc(n \* sizeof(float));  b[0] = 1.00;  b[1] = 2.00;  /\* printf("Enter the size of the matrix: ");  scanf("%d", &n);    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");  }      printf("Enter the number of iterations:\n");  double\* b = (double\*)malloc(n \* sizeof(double));  for (int i = 0; i < n; i++) {  scanf("%lf", &b[i]);  } \*/  /\* // 顺序gauss消去法  for (int i = 0; i < n; i++) {  for (int j = i + 1; j < n; j++) {  float 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];  }  } \*/  //列主元高斯消去法  //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");  return 1;  }  if (max\_index != i) {//交换两行  for(int j = i; j < n; j++) {  float temp = A[i][j];  A[i][j] = A[max\_index][j];  A[max\_index][j] = temp;  }  float 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  float 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  printf("The transformed matrix is:\n");  for (int i = 0; i < n; i++) {  for (int j = 0; j < n; j++) {  printf("%f ", A[i][j]);  }  printf("\n");  }  printf("The transformed vector is:\n");  printf("\n");  for(int i = 0; i < n; i++) {  printf("%f ", b[i]);  }  printf("\n");  float\* x = (float\*)malloc(n \* sizeof(float));  // 解线性方程组  for (int i = n - 1; i >= 0; i--) {  float 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("%f\n", x[i]);  }  printf("\n");  for (int i = 0; i < n; i++) {  free(A[i]);  }  free(A);  free(b);  return 0;  }    实验2.2：（列主元顺序Gauss消去法求解线性方程组2）  #include <stdio.h>  #include <stdlib.h>  #include <math.h>  int main(void) {  float \*\*A = NULL;  int n = 3;  A = (float \*\*)malloc(n \* sizeof(float \*));  if(A == NULL) {  printf("Memory allocation failed.\n");  return 1;  }  for (int i = 0; i < n; i++) {  A[i] = (float \*)malloc(n \* sizeof(float));  }  A[0][0] = 2;A[0][1] = 4;A[0][2] = -2;  A[1][0] = 1;A[1][1] = -3;A[1][2] = -3;  A[2][0] = 4;A[2][1] = 2;A[2][2] = 2;  float\* b = (float\*)malloc(n \* sizeof(float));  b[0] = 2;  b[1] = -1;  b[2] = 3;  /\* printf("Enter the size of the matrix: ");  scanf("%d", &n);    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");  }      printf("Enter the number of iterations:\n");  double\* b = (double\*)malloc(n \* sizeof(double));  for (int i = 0; i < n; i++) {  scanf("%lf", &b[i]);  } \*/  /\* // 顺序gauss消去法  for (int i = 0; i < n; i++) {  for (int j = i + 1; j < n; j++) {  float 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];  }  } \*/  //列主元高斯消去法  //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");  return 1;  }  if (max\_index != i) {//交换两行  for(int j = i; j < n; j++) {  float temp = A[i][j];  A[i][j] = A[max\_index][j];  A[max\_index][j] = temp;  }  float 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  float 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  printf("The transformed matrix is:\n");  for (int i = 0; i < n; i++) {  for (int j = 0; j < n; j++) {  printf("%f ", A[i][j]);  }  printf("\n");  }  printf("The transformed vector is:\n");  printf("\n");  for(int i = 0; i < n; i++) {  printf("%f ", b[i]);  }  printf("\n");  float\* x = (float\*)malloc(n \* sizeof(float));  // 解线性方程组  for (int i = n - 1; i >= 0; i--) {  float 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("%f\n", x[i]);  }  printf("\n");  for (int i = 0; i < n; i++) {  free(A[i]);  }  free(A);  free(b);  return 0;  }    实验2.3：（列主元顺序Gauss消去法求解线性方程组3）  #include <stdio.h>  #include <stdlib.h>  #include <math.h>  int main(void) {  float \*\*A = NULL;  int n = 3;  A = (float \*\*)malloc(n \* sizeof(float \*));  if(A == NULL) {  printf("Memory allocation failed.\n");  return 1;  }  for (int i = 0; i < n; i++) {  A[i] = (float \*)malloc(n \* sizeof(float));  }  A[0][0] = 0.012;A[0][1] = 0.01;A[0][2] = 0.167;  A[1][0] = 1;A[1][1] = 0.8334;A[1][2] = 5.91;  A[2][0] = 3200;A[2][1] = 1200;A[2][2] = 4.2;  float\* b = (float\*)malloc(n \* sizeof(float));  b[0] = 0.6781;  b[1] = 12.1;  b[2] = 981;  /\* printf("Enter the size of the matrix: ");  scanf("%d", &n);    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");  }      printf("Enter the number of iterations:\n");  double\* b = (double\*)malloc(n \* sizeof(double));  for (int i = 0; i < n; i++) {  scanf("%lf", &b[i]);  } \*/  /\* // 顺序gauss消去法  for (int i = 0; i < n; i++) {  for (int j = i + 1; j < n; j++) {  float 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];  }  } \*/  //列主元高斯消去法  //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");  return 1;  }  if (max\_index != i) {//交换两行  for(int j = i; j < n; j++) {  float temp = A[i][j];  A[i][j] = A[max\_index][j];  A[max\_index][j] = temp;  }  float 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  float 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  printf("The transformed matrix is:\n");  for (int i = 0; i < n; i++) {  for (int j = 0; j < n; j++) {  printf("%f ", A[i][j]);  }  printf("\n");  }  printf("The transformed vector is:\n");  printf("\n");  for(int i = 0; i < n; i++) {  printf("%f ", b[i]);  }  printf("\n");  float\* x = (float\*)malloc(n \* sizeof(float));  // 解线性方程组  for (int i = n - 1; i >= 0; i--) {  float 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("%f\n", x[i]);  }  printf("\n");  for (int i = 0; i < n; i++) {  free(A[i]);  }  free(A);  free(b);  return 0;  }    实验3.1：（Doolittle三角分解法（LU分解）求解线性方程组1）  #include <stdio.h>  #include <stdlib.h>  int main(void) {  // 矩阵大小  int n=2;  /\* // 输入矩阵与右侧向量  printf("Enter the size of the matrix: ");  scanf("%d", &n); \*/  float \*\*A = (float \*\*)malloc(n \* sizeof(float \*));  for (int i = 0; i < n; i++) {  A[i] = (float \*)malloc(n \* sizeof(float));  }  /\* 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]);  }  } \*/  //float \*b = (float \*)malloc(n \* sizeof(float));  /\* printf("Enter the elements of the righr-hand side:\n");  for (int i = 0; i < n; i++) {  scanf("%lf", &b[i]);  } \*/    A[0][0] = 0.000000010;A[0][1] = 1.00;  A[1][0] = 1.00;A[1][1] = 1.00;  float\* b = (float\*)malloc(n \* sizeof(float));  b[0] = 1.00;  b[1] = 2.00;    /\* // 输出矩阵    printf("The matrix is:\n");  for (int i = 0; i < n; i++) {  for (int j = 0; j < n; j++) {  printf("%lf ", A[i][j]);  }  printf("\n");  }  printf("The righr-hand side is:\n");  for (int i = 0; i < n; i++) {  printf("%lf ", b[i]);  }  printf("\n"); \*/  // 进行LU分解  float \*\*L = (float \*\*)malloc(n \* sizeof(float \*));  for (int i = 0; i < n; i++) {  L[i] = (float \*)malloc(n \* sizeof(float));  }  float \*\*U = (float \*\*)malloc(n \* sizeof(float \*));  for (int i = 0; i < n; i++) {  U[i] = (float \*)malloc(n \* sizeof(float));  }  //给L和U初始化为0  for (int i = 0; i < n; i++) {  for (int j = 0; j < n; j++) {  L[i][j] = 0;  U[i][j] = 0;  }  }    //进行LU分解    //对第一行U赋值  for (int j = 0; j < n; j++) {  U[0][j] = A[0][j];  }  //对第一列L赋值  L[0][0] = 1;  for (int i = 1; i < n; i++) {  L[i][0] = A[i][0] / U[0][0];  }      //对剩余元素进行LU分解    for (int r = 1; r < n; r++) {    //对第r行的U赋值  for (int i = r; i < n; i++) {  float sum = 0;  for (int k = 0; k < r; k++) {  sum += L[r][k] \* U[k][i];  }  U[r][i] = A[r][i] - sum;  }  //对第r列的L赋值  for (int i = r; i < n; i++){  float sum = 0;  for (int k = 0; k < r; k++){  sum += L[i][k] \* U[k][r];  }  if (r != n-1) {  L[i][r] = (A[i][r] - sum) / U[r][r];  }  else if(r == n-1 && i == n-1){  L[i][r] = 1;  }  else{  L[i][r] = 0;  }  }  }  //输出L和U  printf("The L matrix is:\n");  for (int i = 0; i < n; i++) {  for (int j = 0; j < n; j++) {  printf("%lf ", L[i][j]);  }  printf("\n");  }  printf("The U matrix is:\n");  for (int i = 0; i < n; i++) {  for (int j = 0; j < n; j++) {  printf("%lf ", U[i][j]);  }  printf("\n");  }  // 求解y  float \*y = (float \*)malloc(n \* sizeof(float));  y[0] = b[0];  for (int i = 1; i < n; i++) {  float sum = 0;  for (int k = 0; k < i; k++) {  sum += L[i][k] \* y[k];  }  y[i] = b[i] - sum;  }  // 求解x  float \*x = (float \*)malloc(n \* sizeof(float));  x[n-1] = y[n-1] / U[n-1][n-1];  for (int i = n-2; i >= 0; i--) {  float sum = 0;  for (int k = i+1; k < n; k++) {  sum += U[i][k] \* x[k];  }  x[i] = (y[i] - sum) / U[i][i];  }      // 输出结果  printf("The solution is:\n");  for (int i = 0; i < n; i++) {  printf("%f\n", x[i]);  }  printf("\n");  // 释放内存  for (int i = 0; i < n; i++) {  free(A[i]);  free(L[i]);  free(U[i]);  }  free(A);  free(L);  free(U);  free(b);  free(y);  free(x);  return 0;  }    实验3.2：（Doolittle三角分解法（LU分解）求解线性方程组2）  #include <stdio.h>  #include <stdlib.h>  int main(void) {  // 矩阵大小  int n=3;  /\* // 输入矩阵与右侧向量  printf("Enter the size of the matrix: ");  scanf("%d", &n); \*/  float \*\*A = (float \*\*)malloc(n \* sizeof(float \*));  for (int i = 0; i < n; i++) {  A[i] = (float \*)malloc(n \* sizeof(float));  }  /\* 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]);  }  } \*/  //float \*b = (float \*)malloc(n \* sizeof(float));  /\* printf("Enter the elements of the righr-hand side:\n");  for (int i = 0; i < n; i++) {  scanf("%lf", &b[i]);  } \*/    A[0][0] = 2;A[0][1] = 4;A[0][2] = -2;  A[1][0] = 1;A[1][1] = -3;A[1][2] = -3;  A[2][0] = 4;A[2][1] = 2;A[2][2] = 2;  float\* b = (float\*)malloc(n \* sizeof(float));  b[0] = 2;  b[1] = -1;  b[2] = 3;    /\* // 输出矩阵    printf("The matrix is:\n");  for (int i = 0; i < n; i++) {  for (int j = 0; j < n; j++) {  printf("%lf ", A[i][j]);  }  printf("\n");  }  printf("The righr-hand side is:\n");  for (int i = 0; i < n; i++) {  printf("%lf ", b[i]);  }  printf("\n"); \*/  // 进行LU分解  float \*\*L = (float \*\*)malloc(n \* sizeof(float \*));  for (int i = 0; i < n; i++) {  L[i] = (float \*)malloc(n \* sizeof(float));  }  float \*\*U = (float \*\*)malloc(n \* sizeof(float \*));  for (int i = 0; i < n; i++) {  U[i] = (float \*)malloc(n \* sizeof(float));  }  //给L和U初始化为0  for (int i = 0; i < n; i++) {  for (int j = 0; j < n; j++) {  L[i][j] = 0;  U[i][j] = 0;  }  }    //进行LU分解    //对第一行U赋值  for (int j = 0; j < n; j++) {  U[0][j] = A[0][j];  }  //对第一列L赋值  L[0][0] = 1;  for (int i = 1; i < n; i++) {  L[i][0] = A[i][0] / U[0][0];  }      //对剩余元素进行LU分解    for (int r = 1; r < n; r++) {    //对第r行的U赋值  for (int i = r; i < n; i++) {  float sum = 0;  for (int k = 0; k < r; k++) {  sum += L[r][k] \* U[k][i];  }  U[r][i] = A[r][i] - sum;  }  //对第r列的L赋值  for (int i = r; i < n; i++){  float sum = 0;  for (int k = 0; k < r; k++){  sum += L[i][k] \* U[k][r];  }  if (r != n-1) {  L[i][r] = (A[i][r] - sum) / U[r][r];  }  else if(r == n-1 && i == n-1){  L[i][r] = 1;  }  else{  L[i][r] = 0;  }  }  }  //输出L和U  printf("The L matrix is:\n");  for (int i = 0; i < n; i++) {  for (int j = 0; j < n; j++) {  printf("%lf ", L[i][j]);  }  printf("\n");  }  printf("The U matrix is:\n");  for (int i = 0; i < n; i++) {  for (int j = 0; j < n; j++) {  printf("%lf ", U[i][j]);  }  printf("\n");  }  // 求解y  float \*y = (float \*)malloc(n \* sizeof(float));  y[0] = b[0];  for (int i = 1; i < n; i++) {  float sum = 0;  for (int k = 0; k < i; k++) {  sum += L[i][k] \* y[k];  }  y[i] = b[i] - sum;  }  // 求解x  float \*x = (float \*)malloc(n \* sizeof(float));  x[n-1] = y[n-1] / U[n-1][n-1];  for (int i = n-2; i >= 0; i--) {  float sum = 0;  for (int k = i+1; k < n; k++) {  sum += U[i][k] \* x[k];  }  x[i] = (y[i] - sum) / U[i][i];  }      // 输出结果  printf("The solution is:\n");  for (int i = 0; i < n; i++) {  printf("%f\n", x[i]);  }  printf("\n");  // 释放内存  for (int i = 0; i < n; i++) {  free(A[i]);  free(L[i]);  free(U[i]);  }  free(A);  free(L);  free(U);  free(b);  free(y);  free(x);  return 0;  }    实验3.3：（Doolittle三角分解法（LU分解）求解线性方程组3）  #include <stdio.h>  #include <stdlib.h>  int main(void) {  // 矩阵大小  int n=3;  /\* // 输入矩阵与右侧向量  printf("Enter the size of the matrix: ");  scanf("%d", &n); \*/  float \*\*A = (float \*\*)malloc(n \* sizeof(float \*));  for (int i = 0; i < n; i++) {  A[i] = (float \*)malloc(n \* sizeof(float));  }  /\* 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]);  }  } \*/  //float \*b = (float \*)malloc(n \* sizeof(float));  /\* printf("Enter the elements of the righr-hand side:\n");  for (int i = 0; i < n; i++) {  scanf("%lf", &b[i]);  } \*/    A[0][0] = 0.012;A[0][1] = 0.01;A[0][2] = 0.167;  A[1][0] = 1;A[1][1] = 0.8334;A[1][2] = 5.91;  A[2][0] = 3200;A[2][1] = 1200;A[2][2] = 4.2;  float\* b = (float\*)malloc(n \* sizeof(float));  b[0] = 0.6781;  b[1] = 12.1;  b[2] = 981;    /\* // 输出矩阵    printf("The matrix is:\n");  for (int i = 0; i < n; i++) {  for (int j = 0; j < n; j++) {  printf("%lf ", A[i][j]);  }  printf("\n");  }  printf("The righr-hand side is:\n");  for (int i = 0; i < n; i++) {  printf("%lf ", b[i]);  }  printf("\n"); \*/  // 进行LU分解  float \*\*L = (float \*\*)malloc(n \* sizeof(float \*));  for (int i = 0; i < n; i++) {  L[i] = (float \*)malloc(n \* sizeof(float));  }  float \*\*U = (float \*\*)malloc(n \* sizeof(float \*));  for (int i = 0; i < n; i++) {  U[i] = (float \*)malloc(n \* sizeof(float));  }  //给L和U初始化为0  for (int i = 0; i < n; i++) {  for (int j = 0; j < n; j++) {  L[i][j] = 0;  U[i][j] = 0;  }  }    //进行LU分解    //对第一行U赋值  for (int j = 0; j < n; j++) {  U[0][j] = A[0][j];  }  //对第一列L赋值  L[0][0] = 1;  for (int i = 1; i < n; i++) {  L[i][0] = A[i][0] / U[0][0];  }      //对剩余元素进行LU分解    for (int r = 1; r < n; r++) {    //对第r行的U赋值  for (int i = r; i < n; i++) {  float sum = 0;  for (int k = 0; k < r; k++) {  sum += L[r][k] \* U[k][i];  }  U[r][i] = A[r][i] - sum;  }  //对第r列的L赋值  for (int i = r; i < n; i++){  float sum = 0;  for (int k = 0; k < r; k++){  sum += L[i][k] \* U[k][r];  }  if (r != n-1) {  L[i][r] = (A[i][r] - sum) / U[r][r];  }  else if(r == n-1 && i == n-1){  L[i][r] = 1;  }  else{  L[i][r] = 0;  }  }  }  //输出L和U  printf("The L matrix is:\n");  for (int i = 0; i < n; i++) {  for (int j = 0; j < n; j++) {  printf("%lf ", L[i][j]);  }  printf("\n");  }  printf("The U matrix is:\n");  for (int i = 0; i < n; i++) {  for (int j = 0; j < n; j++) {  printf("%lf ", U[i][j]);  }  printf("\n");  }  // 求解y  float \*y = (float \*)malloc(n \* sizeof(float));  y[0] = b[0];  for (int i = 1; i < n; i++) {  float sum = 0;  for (int k = 0; k < i; k++) {  sum += L[i][k] \* y[k];  }  y[i] = b[i] - sum;  }  // 求解x  float \*x = (float \*)malloc(n \* sizeof(float));  x[n-1] = y[n-1] / U[n-1][n-1];  for (int i = n-2; i >= 0; i--) {  float sum = 0;  for (int k = i+1; k < n; k++) {  sum += U[i][k] \* x[k];  }  x[i] = (y[i] - sum) / U[i][i];  }      // 输出结果  printf("The solution is:\n");  for (int i = 0; i < n; i++) {  printf("%f\n", x[i]);  }  printf("\n");  // 释放内存  for (int i = 0; i < n; i++) {  free(A[i]);  free(L[i]);  free(U[i]);  }  free(A);  free(L);  free(U);  free(b);  free(y);  free(x);  return 0;  }    实验4.1：两点边值问题的有限差分求解  #include <stdio.h>  #include <stdlib.h>  #include <math.h>  void jisuan(int n, double \* er\_1, double \* er\_2) {  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;    n=n+1;  // 释放内存  free(x);  free(u);  free(b);  for (int i = 0; i < n; i++) {  free(A[i]);  }  free(A);      }  int main(void) {  int n1 = 10;  int n2 = 20;  int n3 = 40;  int n4 = 80;  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;    jisuan(n1, &err\_1\_1, &err\_2\_1);  jisuan(n2, &err\_1\_2, &err\_2\_2);  jisuan(n3, &err\_1\_3, &err\_2\_3);  jisuan(n4, &err\_1\_4, &err\_2\_4);    printf("index\tn\terror\_1\t\terror\_2\n");  printf("--------------------------\n");    printf("n1\t10\t%lf\t%lf\n", err\_1\_1, err\_2\_1);  printf("n2\t20\t%lf\t%lf\n", err\_1\_2, err\_2\_2);  printf("n3\t40\t%lf\t%lf\n", err\_1\_3, err\_2\_3);  printf("n4\t80\t%lf\t%lf\n", err\_1\_4, err\_2\_4);  printf("--------------------------\n");  return 0;  }    以下为老师ppt上的答案 | | | |
| 实验分析与总结：  经过本次实验，熟悉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，复习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,我了解到了顺序gauss消去法可能会由于小数被除而造成极大的误差；只有严格对角占优时这种误差才会小；  通过实验2\_1,我知道了列主元gauss消去法消除上述误差的原理；  通过实验1\_1,2\_1与3\_1之间的对比，了解到了LU分解法与顺序gauss分解法的等价性，求解结果相同，但多次求解系数矩阵相同的问题时，LU分解法可以有效提速；  通过实验4\_1,我尝试了使用列主元gauss消去法进行简单的应用，更加深刻理解了算法的逻辑，也认识到了基础算法的重要性，体会到了，当用离散的方法近似求解微分问题时，随划分的加细，结果将更加接近真实值。 | | | |

注：若填写内容较多，可在背面继续填写。