****

**实验报告**

**课程名称： 数值分析第一次上机报告**

**实验项目： 解非线性方程组**

**专业班级： 软件工程1808班**

**姓 名： 赵祎泽 学 号： 181203817**

**实验室号： 图书馆715-1 实验组号：**

**实验时间： 2020.10 批阅时间：**

**指导教师： 张琪 成 绩：**

**沈阳工业大学实验报告**

（适用计算机程序设计类）

专业班级： 学号： 姓名：

实验名称：

|  |
| --- |
| 1.实验目的：  通过编程算法解决解非线性方程组以及解方程组  2.实验内容：  分别用牛顿法、简化牛顿法、牛顿下山法和割线法求解方程  f(x) = xex - 1  分别用高斯消去法，列主元高斯消去法和LU分解法求解下列方程组  2x1+4x2+2x3+6x4=9  4x1+9x2+6x3+15x4=23  2x1+6x2+9x3+18x4=22  6x1+15x2+18x3+40x4=47  3. 实验方案（程序设计说明）  合理运用公式，牛顿法需要公式 f(xk+1) = xk –  简化牛顿法需要公式 f(xk+1) = xk – C 其中C =  牛顿法下山法需要公式 f(xk+1) = xk –  割线法需要公式 xk+1 = xk –  4. 实验步骤或程序（经调试后正确的源程序）  （见附件A）  5．程序运行结果  （见附件A）  6．出现的问题及解决方法 |

**附件A 沈阳工业大学实验报告**

（适用计算机程序设计类）

专业班级： 学号： 姓名：

**实验步骤或程序**：

1. 非线性方程组

牛顿法

核心代码：

double Newton() {

float x0, x1 = 0.0, precision;

int count = 0;

printf("输入初始迭代值x0: ");

scanf("%f", &x0);

printf("迭代要求的精度：");

scanf("%f", &precision);

x1 = x0 - func(x0) / func1(x0);

while (fabs(x1 - x0) >= precision) {

count ++;

x0 = x1;

x1 = x0 - func(x0) / func1(x0);

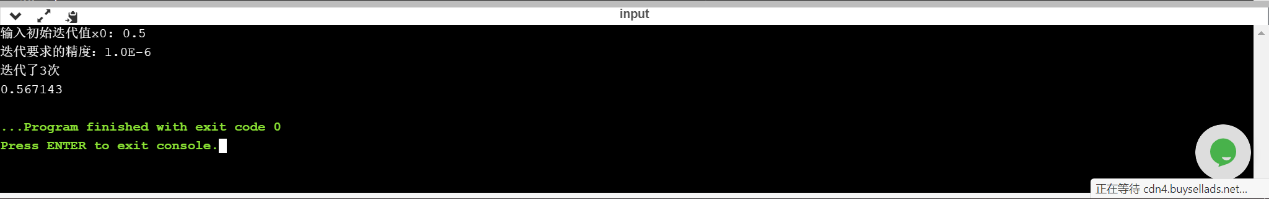
}

printf("迭代了%d次\n", count);

return x1;

}

结果：



简化牛顿法

核心代码：

double Newton() {

double x0, x1 = 0.0, precision;

int count = 0;

printf("输入初始迭代值x0:");

cin >> x0;

printf("迭代要求的精度：");

cin >> precision;

double staticNum = func1(x0);

x1 = x0 - func(x0) / func1(x0);

while (fabs(x1 - x0) >= precision) {

count++;

x0 = x1;

x1 = x0 - func(x0) / staticNum;

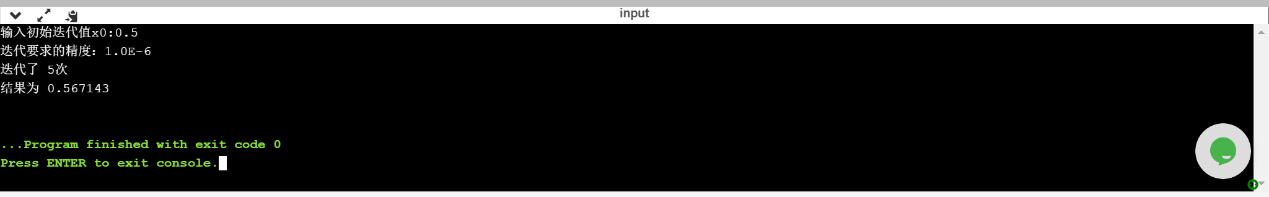
}

cout << "迭代了 " << count<<"次" <<endl;

return x1;

}

结果：

****

牛顿下山法

核心代码：

double Newton() {

double x, x1, precision;

printf("输入初始迭代值x0:");

cin >> x;

printf("输入精度x0:");

cin >> precision;

double y = func(x);

double r = 1.0; //下山因子

int time = 1, count = 0;

while (abs(y) > precision) {

r = 1;

x1 = x - r \* y / (func1(x));

while (abs(func(x1)) > abs(func(x))) {

r = r / 2;

x1 = x - r \* y / func1(x);

}

x = x1;

y = func(x);

time += 1;

count++;

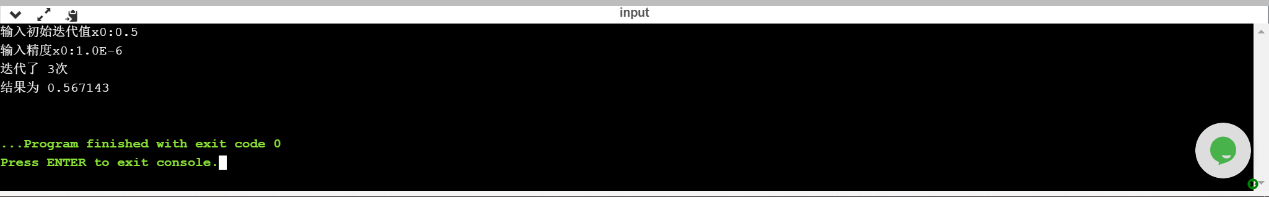
}

cout << "迭代了 " << count << "次" << endl;

return x;

}

结果：



割线法

核心代码：

while (fabs(x - x1) >= x2) {

double tmp = x1;

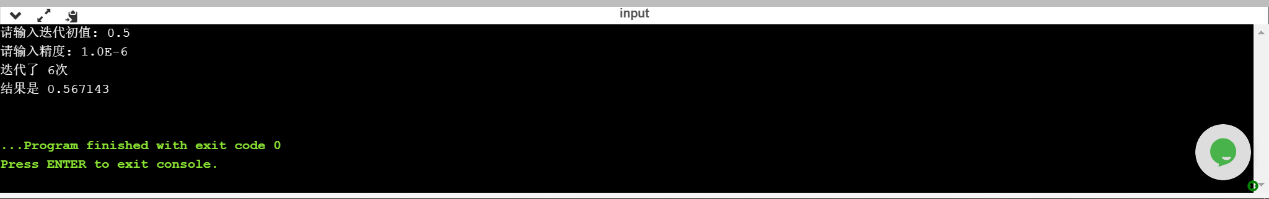
x1 = x1 - f(x1) \* (x1 - x) / (f(x1) - f(x));

x = tmp;

count++;

}

结果：

****

1. 方程组

高斯消去法

核心代码：

#define MAX 10

double A[MAX][MAX]; //系数矩阵

double b[MAX]; //右端项

double X[MAX]; //迭代向量

int NUM; //A的阶数

int size; //最大迭代次数

int main(void)

{

int i, j, k; //计数器

float Aik; //正消过程用到的变量名

float S; //回代过程用到的变量名

//以下代码输入系数矩阵A,右端项b

printf("请输入系数矩阵A的阶数：");

scanf("%d", &NUM);

size = NUM;

for (i = 1; i <= size; i++) {

printf("请输入A的第%d行元素,各元素间以空格间隔:\n", i);

for (j = 1; j <= size; j++)

scanf("%lf", &A[i - 1][j - 1]);

}

printf("输入右端项b,各元素间以空格间隔:\n");

for (i = 1; i <= size; i++) {

scanf("%lf", &b[i - 1]);

}

//在屏幕中输出用户输入的系数矩阵A和矩阵B

printf("您输入的维度是%d!\n您输入的矩阵A[][]:\n", NUM); //在屏幕中输出用户输入的矩阵A

for (i = 0; i < size; i++) {

for (j = 0; j < size; j++)

printf("\t%f\t", A[i][j]);

printf("\n");

}

printf("您输入的矩阵b[]：\n");

for (i = 0; i < size; i++)

printf("\t%f\t\n", b[i]);

printf("\n");//打印矩阵b[]

//以下代码是高斯消去法的主要步骤

for (k = 0; k < size - 1; k++) {

if (!A[k][k])

return -1;

for (i = k + 1; i < size; i++) {

Aik = A[i][k] / A[k][k];

for (j = k; j < size; j++) {

A[i][j] = A[i][j] - Aik \* A[k][j];

}

b[i] = b[i] - Aik \* b[k];

}

}//首先通过正消过程

printf("A[]=\n");

for (i = 0; i < size; i++) {

for (j = 0; j < size; j++)

printf("\t%f\t", A[i][j]);

printf("\n");

}

printf("b[]=\n");

for (i = 0; i < size; i++)

printf("\t%f\t\n", b[i]);

printf("\n"); //此处返回正消过程的结果

X[size - 1] = b[size - 1] / A[size - 1][size - 1];

for (k = size - 2; k >= 0; k--) {

S = b[k];

for (j = k + 1; j < size; j++) {

S = S - A[k][j] \* X[j];

}

X[k] = S / A[k][k];

} //回代

//此处打印出向量X

printf("x[]=\n");

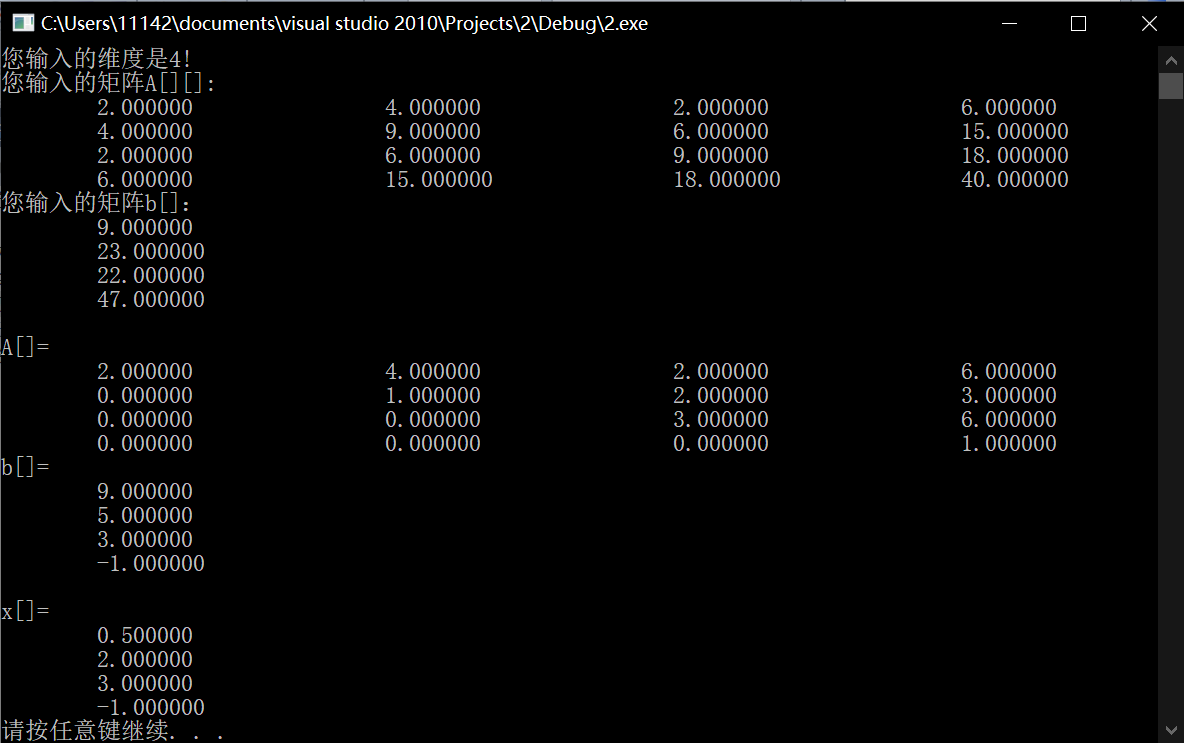
for (i = 0; i < size; i++)

printf("\t%f\t\n", X[i]);

return 0;

}

结果：



列主元高斯消去法

核心代码：

#define MAX 10

double A[MAX][MAX];//系数矩阵

double b[MAX];//右端项

double X[MAX];//求得的方程组结果

int n;

int x;

void setMatrix() {

int i, j;

for (i = 0; i < n; i++) {

printf("请输入系数矩阵A第%d行元素，并以空格作为间隔:\n", i + 1);

for (j = 0; j < n; j++) {

scanf("%lf", &A[i][j]);

}

}

printf("请输入右端项b的元素，并以空格作为间隔:\n");

for (i = 0; i < n; i++) {

scanf("%lf", &b[i]);

}

printf("输入的系数矩阵A[][]为:\n");

for (i = 0; i < n; i++) {

for (j = 0; j < n; j++) {

printf("\t%f\t", A[i][j]);

}

printf("\n\n");

}

printf("\n输入的右端项b[]为:\n");

for (i = 0; i < n; i++) {

printf("\t%f\n\n", b[i]);

}

}

int main(void) {

int i, j, k;

double max;

double temp;

double temp1;

double Aik;

double S;

printf("请输入方程组的阶数:");

scanf("%d", &n);

setMatrix(); //初始化，输入系数矩阵

for (k = 0; k < n - 1; k++) //对整个系数矩阵进行列主元消元

{

max = A[k][k];

for (i = k + 1; i < n; i++) //选主元

{

if (fabs(max) < fabs(A[i][k])) {

max = A[i][k];

x = i; //记录选择交换的行数

}

}

for (j = k; j < n; j++) //选出列主元后，交换对应的两行

{

temp = A[k][j];

A[k][j] = A[x][j];

A[x][j] = temp;

}

temp1 = b[k];

b[k] = b[x];

b[x] = temp1;

//消去过程

for (i = k + 1; i < n; i++) {

Aik = A[i][k] / A[k][k];

for (j = k; j < n; j++) {

A[i][j] = A[i][j] - Aik \* A[k][j];

}

b[i] = b[i] - Aik \* b[k];

}

}

printf("列主元消元后的系数矩阵A[][]:\n\n");

for (i = 0; i < n; i++) {

for (j = 0; j < n; j++) {

printf("\t%f\t", A[i][j]);

}

printf("\n\n");

}

printf("列主元消元后的右端项b[]:\n\n");

for (i = 0; i < n; i++) {

printf("\t%f\t\n\n", b[i]);

}

//回代过程

X[n - 1] = b[n - 1] / A[n - 1][n - 1];

for (k = n - 2; k >= 0; k--) {

S = b[k];

for (j = k + 1; j < n; j++) {

S = S - A[k][j] \* X[j];

}

X[k] = S / A[k][k];

}

printf("输出求得的方程组结果X[]:\n");

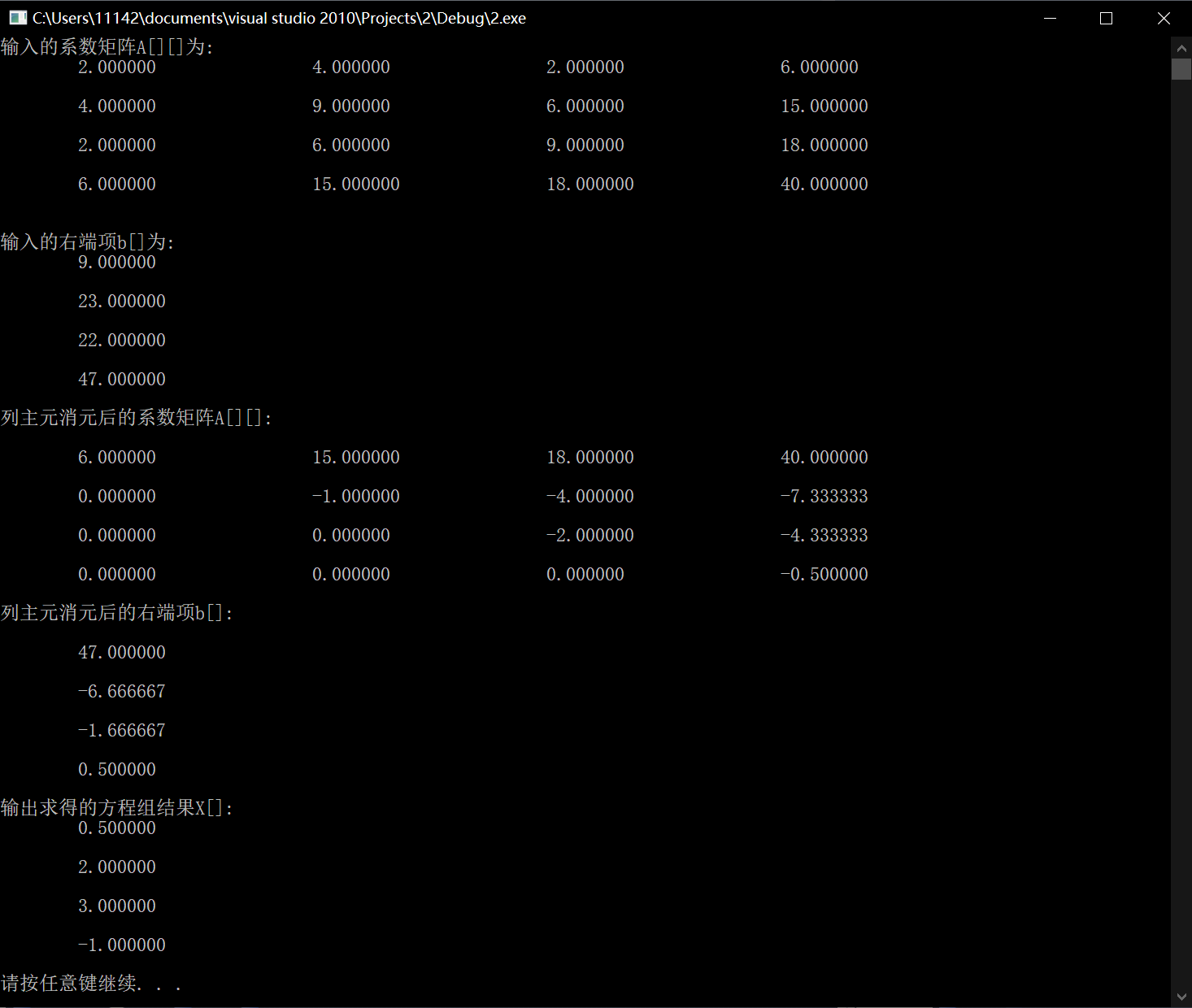
for (i = 0; i < n; i++) {

printf("\t%f\n\n", X[i]);

}

}

结果：



LU分解法

核心代码：

double sumU(double L[4][4] ,double U[4][4], int i, int j ) {

double sU = 0.0;

for (int k = 1; k <= i - 1; k++) {

sU += L[i - 1][k - 1] \* U[k - 1][j - 1];

}

return sU;

} //计算求和1

double sumL(double L[4][4] ,double U[4][4], int i, int j ) {

double sL = 0.0;

for (int k = 0; k <= j - 1; k++) {

sL += L[i - 1][k - 1] \* U[k - 1][j - 1];

}

return sL;

} //计算求和2

double sumY(double L[4][4] ,double y[4],int i) {

double sY = 0.0;

for (int k = 1; k <= i - 1; k++) {

sY += L[i - 1][k - 1] \* y[k - 1];

}

return sY;

} //计算求和3

double sumX(double U[4][4] ,double x[4],int i ,int m) {

double sX = 0.0;

for (int k = i + 1; k <= m; k++) {

sX += U[i - 1][k - 1] \* x[k - 1];

}

return sX;

} //计算求和4

int main(void) {

double a[4][4] = {{2, 4, 2, 6,},

{4, 9, 6, 15,},

{2, 6, 9, 18,},

{6, 15, 18, 40}}; //将系数存入二维数组

double L[4][4] = {0};

double U[4][4] = {0}; //初始化部分

double b[4] = {9, 23, 22, 47};

int n = 4;//n阶

//输出[Ab]

printf("[A]:\n");

for (int i = 1; i <= n; i++) {

for (int j = 1; j <= n; j++) {

printf("%f\t", a[i - 1][j - 1]);

}

printf("\n");

}

//计算L,U

for (int i = 1; i <= n; i++) {

L[i - 1][i - 1] = 1;//对角线元素为1

for (int j = i; j <= n; j++) {

//由于数组下标从0开始 所以i-1,j-1

U[i - 1][j - 1] = a[i - 1][j - 1] - sumU(L, U, i, j);

if (j + 1 <= n) L[j][i - 1] = (a[j][i - 1] - sumL(L, U, j + 1, i)) / U[i - 1][i - 1];//i变j+1，j变i

}

}

//输出U

printf("U:\n");

for (int i = 1; i <= n; i++) {

for (int j = 1; j <= n; j++) {

printf("%f\t", U[i - 1][j - 1]);

}

printf("\n");

}

//输出L

printf("L:\n");

for (int i = 1; i <= n; i++) {

for (int j = 1; j <= n; j++) {

printf("%f\t", L[i - 1][j - 1]);

}

printf("\n");

}

//由Ly=b 求y

double y[4] = {0.0};

y[0] = b[0];//y(1) = b(1);

for (int i = 2; i <= n; i++) {

y[i - 1] = b[i - 1] - sumY(L, y, i);

}

//由 Ux=y 求x

double x[4] = {0.0};

for (int i = n; i >= 1; i--) {

x[i - 1] = (y[i - 1] - sumX(U, x, i, n)) / U[i - 1][i - 1];

}

//输出y

printf("y:\n");

for (int i = 0; i < n; i++) {

printf("%f\n", y[i]);

}

//输出x

printf("x:\n");

for (int i = 0; i < n; i++) {

printf("%f\n", x[i]);

}

printf("\n");

结果：

