# 矩阵求逆实现原理

**——高斯消元法求逆**

在初值时，我们首次接触求解二元一次方程组，当时我只是知道求解的过程那样来写是理所当然的，就想知道这个方法是谁想出来的。后来随着学习的深入，知道这种方法叫做高斯消元法。此次使用原就是高斯消元法。矩阵求逆的方法还有公式法（伴随矩阵法），那我为什么选择使用高斯消元法呢？这是因为在当今计算机内存很大的背景小，我们这是希望程序运行的时间越小越好，即算法的时间复杂度越小越好。而公式法求逆的时间复杂度是(n+1)!级，而高斯消元法只有n2级，远小于公式法，所以选择了高斯消元法。

高斯消元法最初用来求解线性方程注的，当将方程组用矩阵来表示时，消元的过程就变成了矩阵的初等行变换的过程。那求解线性方程组与矩阵的求逆有什么关系呢？

方程Ax = b是我们常见的非齐次线性方程组，若现在假设矩阵A可逆，那方程组的结就是：

x = A-1b (1)

我们又知当矩阵可逆时有AA-1 = A-1A = I。所以当b = I时，(1)式就变成了：

x = A-1 (2)

即意味着此时线性方程组Ax = b的解即就是矩阵A的逆矩阵。

明白了上述的原理之后，下面就给出求逆的过程（原理）：

设待求逆的矩阵为A，且可逆。

，只要将前面的增广矩阵经过初等行变换，转换成如后部分增广矩阵的形式，那么B = A-1。

初等行变换包括如下三种：

1. Eij——第i行与第j行进行交换
2. Ei(k)——第i行乘以非0倍数k
3. Eij(k)——第i行乘以非0倍数k加到第j行

程序实现过程：

1. 输入方阵A，及单位矩阵B（用于记录对矩阵A进行的所有变换，在下面对矩阵A进行的所有初等行变换都同样作用到矩阵B上，从而最终的B矩阵就是矩阵A的逆）。
2. 首先是进行列选主元运算。对mXn阶矩阵A,在确定第k个主元()时，先从该列自主元位置()至列尾的所有元素中选择绝对值最大的元素，与交换。列选主元可以保证在向下消去时的乘子不超过1，这也抑制了数据误差的放大和传播[1]。在列选主元的时候判断矩阵是否可逆——当发现该列对角元及至列尾的元素绝对值的最大值为0时，矩阵不可逆。

如：

主元在第3行，所以第一行要与第三行进行交换。

1. 列选主元后，将该行除以主元。将对角元元素化为1，方便后续的消元计算。

在此用\*表示计算结果。

1. 向下消元。通过初等行变换将第一列对角元以下的元素变为0。

1. 对后续列循环该过程，最后将矩阵A化为单位上三角矩阵U（矩阵可逆的情况下，如果对角元元素为0，则判定矩阵不可逆）

（当矩阵不可逆时会出现类似这样的矩阵）

1. 对U矩阵进行初等行变换向上消元，直至将矩阵U化为单位阵I

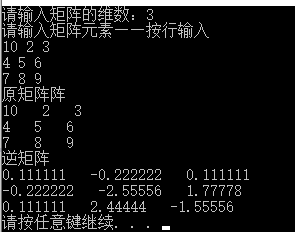
1. 完成，显示逆矩阵

参考文献

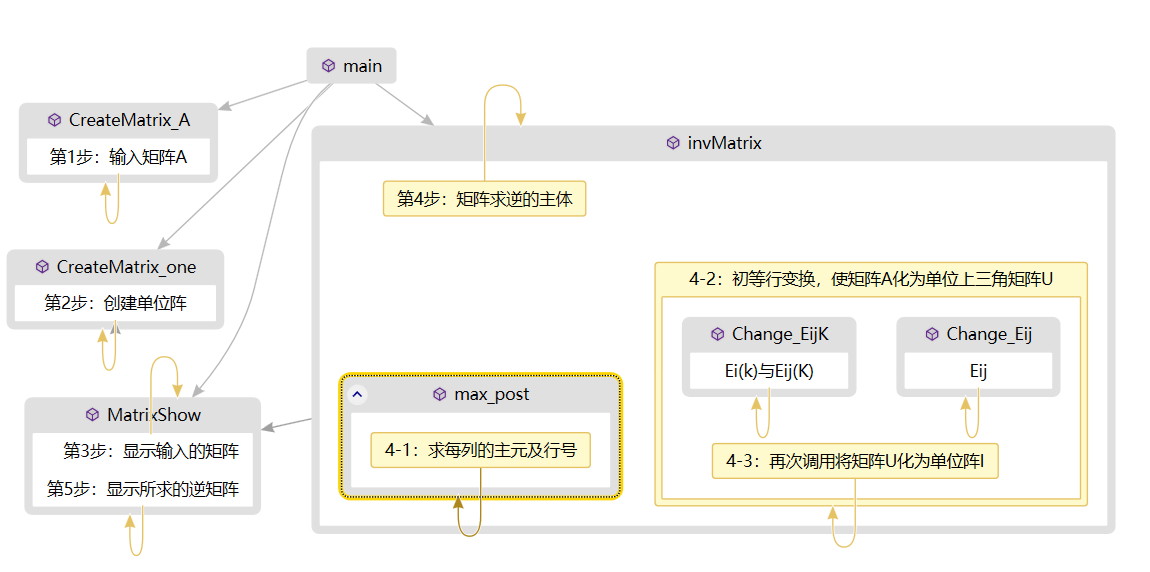
1. 李继根,张新发.矩阵分析与计算[M].湖北武昌:武汉大学出版社,2013年:18页.

附录

1. 输入及显示界面



1. 程序执行图，及调用关系图



1. 原程序

#include <iostream>

#include <cmath>

#include <ctime>

//定义最大维度

#define MaxN 300

using namespace std;

//创建矩阵

double \*CreateMatrix\_A(int N)

{

double \*pMatrix\_ = new double[N\*N + 1]; //一维向量从二维矩阵,第一个元素数矩阵维度信息N

cout << "请输入矩阵元素——按行输入" << endl;

for (int i = 1; i <= N\*N; i++)

{

cin >> pMatrix\_[i];

}

//cout << "输入完" << endl << "pMatrix\_A" << endl;

pMatrix\_[0] = N;

return pMatrix\_;

}

//创建单位阵

double \*CreateMatrix\_one(int N)

{

double \*pMatrix\_one = new double[N \* N + 1]; //一维向量从二维矩阵,第一个元素数矩阵维度信息N

for (int j=0, i = 1; i <= N \* N; i++)

{

if (i == 1 + j \* (N + 1))

{

pMatrix\_one[i] = 1;

j += 1;

}

else pMatrix\_one[i] = 0;

}

pMatrix\_one[0] = N;

return pMatrix\_one;

}

//初等变换函数 Eij——交换i、j两行

int Change\_Eij(double Mat[], int ri, int rj) //ri,rj都是从1来开始表示行数

{

double temp = 0;

int N = Mat[0];

if (ri == rj || ri < 1 || rj < 1 || ri > N || rj > N)

return 0;

else

for (int i = 1; i <= N; i++)

{

temp = Mat[N \* (ri - 1) + i];

Mat[N \* (ri - 1) + i] = Mat[N \* (rj - 1) + i];

Mat[N \* (rj - 1) + i] = temp;

}

return 0;

}

//初等变换函数 Eij(k) Ei(k)

int Change\_EijK(double Mat[], int ri, double k = 1, int rj = 0)//ri,rj都是从1来开始表示行数

{

int N = Mat[0];

if (rj != 0) //Eij(k)

{

if (ri < 1 || rj < 1 || ri > N || rj > N || k ==0)

return 0;

else

for (int i = 1; i <= N; i++)

Mat[N \* (rj - 1) + i] = Mat[N \* (rj - 1) + i] + k \* Mat[N \* (ri - 1) + i];

}

else if (rj == 0) //Ei(k)

if (ri < 1 || ri > N || k == 0)

return 0;

else

for (int i = 1; i <= N; i++)

Mat[N \* (ri - 1) + i] = k \* Mat[N \* (ri - 1) + i];

return 0;

}

//返回绝对值最大值所在的行

bool max\_post(double mat[], int start, int length)

{

double max = -1;

int post = 0;

for (int i = start; i < length; i++)

{

if (abs(mat[i]) > max)

{

max = abs(mat[i]);

post = i;

}

}

mat[0] = post +1;// 主元所在的行

if (max == 0)

return false; //矩阵不可逆

return true; //矩阵可逆

}

//矩阵求逆——初等行变换法[Mat1 Mat2]

bool invMatrix(double Mat1[], double Mat2[])

{

bool inv\_Yes;//是否可逆标志

int N = Mat1[0];//矩阵位数

int main\_row = 0;//主元所在行

double PriElementValue = 0;//主元值

double Nextvalue = 0;//用于消元时使用

double \*tempCol\_Mat1 = new double[N];

for (int s = 0; s < N; s++) //向下消元——最后Mat1变为单位上三角矩阵

{

for (int i = 0; i < N; i++)//取Mat1的列向量

tempCol\_Mat1[i] = Mat1[1 + i \* N + s];

inv\_Yes = max\_post(tempCol\_Mat1, s, N);

if (inv\_Yes == false)//矩阵不可逆

return false;

main\_row = tempCol\_Mat1[0];

//可逆时交换主元行

Change\_Eij(Mat1, s + 1, main\_row);//A

Change\_Eij(Mat2, s + 1, main\_row);//扩展阵

PriElementValue = Mat1[1 + s \* (N + 1)];//主元值

Change\_EijK(Mat1, s + 1, 1.0/PriElementValue); //将改行除以主元值，将对角元化为1

Change\_EijK(Mat2, s + 1, 1.0/PriElementValue);//扩展阵

for (int i = s + 1; i < N; i++)//消元

{

Nextvalue = Mat1[i \* N + 1 + s];

Change\_EijK(Mat1, s + 1, -Nextvalue, i + 1);

Change\_EijK(Mat2, s + 1, -Nextvalue, i + 1);

}

}

for (int s = 0; s < N; s++)

if (Mat1[1 + s \* (N + 1)] == 0)//主元值

return false;//不可逆

for (int j = 0, s = N; s > 1; --s)//向上消元——最后Mat1变为单位矩阵

{

for (int i =1 ; i <s ; i++)

{

Nextvalue = Mat1[N \* N - j - i \* N \* (j + 1)];

Change\_EijK(Mat1, s, -Nextvalue, s - i);

Change\_EijK(Mat2, s, -Nextvalue, s - i);

}

j++;

}

delete[] tempCol\_Mat1;

return true;

}

//显示矩阵

void MatrixShow(double Mat[])

{

for (int j = 1, i = 1; i <= pow(Mat[0], 2); i++)

{

if (i == j \* Mat[0] +1)

{

j += 1;

cout << endl;

}

cout << Mat[i] << " ";

}

cout << endl;

}

int main()

{

int N;//矩阵维度

bool cri\_Y;//可逆

cout << "请输入矩阵的维数：";

cin >> N; //输入矩阵维度

double \*pMatrix\_A = CreateMatrix\_A(N);//创建矩阵A

double \*pMatrix\_Aug = CreateMatrix\_one(N);//创建单位阵

cout << "原矩阵阵"<<endl;

MatrixShow(pMatrix\_A);//显示矩阵A

//cout << "原单位阵" << endl;

//MatrixShow(pMatrix\_Aug);//显示矩阵I

cri\_Y = invMatrix(pMatrix\_A, pMatrix\_Aug);

if (cri\_Y == false)

cout << "矩阵不可逆" << endl;

else

{

cout << "逆矩阵" << endl;

MatrixShow(pMatrix\_Aug);//显示矩阵A

}

system("pause");

delete[] pMatrix\_A;

delete[] pMatrix\_Aug;

return 0;

}