Sanjai Syamaprasad 12/22/14

Linear Algebra EMPLID:10882651

// Sanjai Syamaprasad 12/21/14

// Project: Inverse of 3x3 matrix using row reduction

// using array as container, but could have also done this using vector of type vector for variable sized square matrices

// could have used classes for setters, getters, private and public functions, but for purpose of our program

// , it is not necessary

// also could have used a class with private members numerator and denominator to keep fractions

// could have done this by creating 1 matrix from original matrix and augmented matrix

// another method can be used to find inverse which is coded in second program

#include <iostream>

#include <iomanip>

using namespace std;

double det(double[][3]);

double cofactor(double [][3],int,int);

void rowReduce(double [][3]);

void swapRows(double [][3],int,int);

void addRows(double [][3],int,int,double);

void divRowbyPivot(double [][3],int,int);

void multRowbyConstant(double [][3],int, int);

void writesqMatrix(double [][3]);

int main()

{

double a[3][3],b[3][3],temp1,temp2, temp3;

double c;

int i,j,k,p;

cout<<"We will find inverse of a 3x3 matrix!"<<endl<<endl;

for(i=0;i<3;i++)

{

for(j=0;j<3;j++)

{

cout<<"Enter element ("<<i<<","<<j<<"): ";

cin>>a[i][j];

if(i==j)

b[i][j]=1;

else

b[i][j]=0;

}

}

cout<<endl<<"Matrix A: "<<endl<<endl;

writesqMatrix(a);

if(det(a)==0)

{

cout<< "Matrix A does not have an inverse as the determinant is zero!"<<endl<<endl;

return 0;

}

for(i=0;i<3;i++)

{

//ignoring rows in echelon form, selecting row w/ largest absolute value (magnitude)

// in current leading column

temp1=a[i][i];

if(temp1<0)

temp1\*=-1;

p=i;

for(j=i+1;j<3;j++)

{

if(a[j][i]<0)

temp2=a[j][i]\*(-1);

else

temp2=a[j][i];

if(temp1<0)

temp1\*=-1;

if(temp2>temp1)

{

p=j;

temp1=a[j][i];

}

}

//swapping the row with largest magnitude in current leading column with the leading row for that column

swapRows(a, i, p);

swapRows(b, i, p);

//dividing leading row by a[i][i] to get lead as 1

temp3=a[i][i];

for(j=0;j<3;j++)

{

a[i][j]=a[i][j]/temp3;

b[i][j]=b[i][j]/temp3;

}

//making other elements in column zero by

// replacing each non leading row by subtracting itself from the leading row multiplied by the negative value

// of the column element of the non leading row

for(k=0;k<3;k++)

{

if(k==i)

continue;

c=a[k][i]\*-1.0;

addRows(a, i, k,c);

addRows(b, i, k,c);

}

}

//

cout<<"Matrix A^-1: "<<endl<<endl;

writesqMatrix(b);

return 0;

}

//determinant can be found using recursive calls if variable sized square matrix

// , but for purpose of 3x3 matrix, this is not necessary

double det(double data[][3])

{

double determinant;

double x,y,z;

x=data[0][0]\*cofactor(data,0,0);

y=data[0][1]\*cofactor(data,0,1);

z=data[0][2]\*cofactor(data,0,2);

determinant=x+y+z;

return determinant;

}

double cofactor(double data[][3],int x,int y)

{

double cofactor\_v;

cofactor\_v = data[(x + 1) % 3][(y + 1) % 3]

\* data[(x + 2) % 3][(y + 2) % 3]

- data[(x + 1) % 3][(y + 2) % 3]

\* data[(x + 2) % 3][(y + 1) % 3];

return cofactor\_v;

}

// swaps 2 rows of a 3x3 matrix

void swapRows(double matrix[][3], int row1, int row2)

{

double temp;

for(int i=0; i<3; i++)

{

temp=matrix[row1][i];

matrix[row1][i]= matrix[row2][i];

matrix[row2][i]=temp;

}

}

// adds row 1 multiplied by a constant to row 2 for a 3x3 matrix

void addRows(double matrix [][3], int row1, int row2, double c)

{

for(int i=0; i<3; i++)

matrix[row2][i]+=c\*matrix[row1][i];

}

//output the 3x3 matrix

void writesqMatrix(double matrix[][3])

{

for(int i=0; i<3; i++)

{

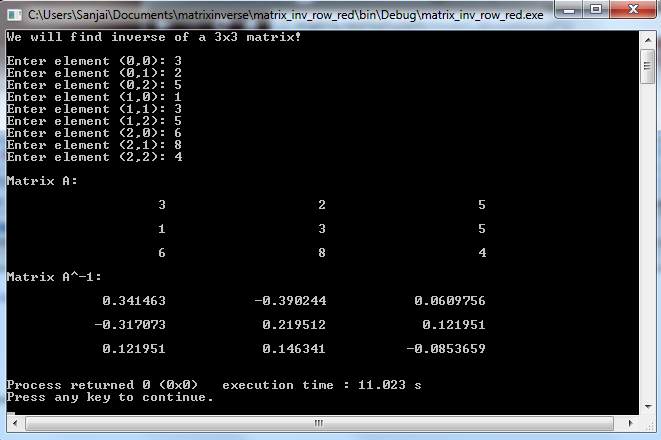
for(int j=0; j<3; j++)

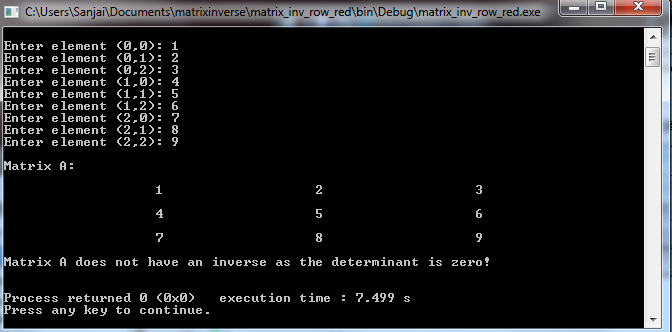
cout<<setw(20)<<matrix[i][j];

cout<<endl<<endl;

}

}





//This second method is a lot easier for small matrices. I was able to check inverse using this method on first exam as I made arithmetic mistake with reducing to row echelon form.

#include <iostream>

#include <iomanip>

// This method provides easy solution for inverse.

// Inverse matrix is the transpose of the adjoint matrix of A divided by determinant if

// determinant is not equal to zero.

// A^-1=adjoint(A)/determinant(A)

//adjoint(A)= (cofactor matrix(A))^T

using namespace std;

int det(int [][3]);

int cofactor(int [][3],int ,int);

int GCD(int, int);

void writesqMatrix(int [][3]);

int main()

{

int numerator, denominator, gcd, determ,i, j;

int matrix[3][3];

cout<<"We will find inverse of a 3x3 matrix!"<<endl<<endl;

for(i=0;i<3;i++)

{

for(j=0;j<3;j++)

{

cout<<"Enter element ("<<i<<","<<j<<"): ";

cin>>matrix[i][j];

}

}

cout<<endl<<"Matrix A: "<<endl<<endl;

writesqMatrix(matrix);

determ=det(matrix);

if(determ==0){

cout<< "Matrix does not have inverse as determinant is zero."<<endl<<endl;

return 0;

}

cout<<"Matrix A^-1: "<<endl<<endl;

for (i=0; i<3; i++)

{

for (j=0; j<3; j++)

{

numerator=cofactor(matrix, j, i);

denominator=determ;

//reducing fraction by dividing by greatest common factor

gcd=GCD(numerator, denominator);

numerator/=gcd;

denominator/=gcd;

//making negative sign appear before fraction instead of accompanying denominator

if(denominator<0)

{

numerator\*=-1;

denominator\*=-1;

}

cout<<setw(20)<<numerator<<"/"<<denominator;

}

cout<<endl;

}

return 0;

}

//returning cofactor

int cofactor(int data[][3],int x,int y)

{

int cofactor\_v;

cofactor\_v = data[(x + 1) % 3][(y + 1) % 3]

\* data[(x + 2) % 3][(y + 2) % 3]

- data[(x + 1) % 3][(y + 2) % 3]

\* data[(x + 2) % 3][(y + 1) % 3];

return cofactor\_v;

}

//returns determinant for a 3x3 matrix

int det(int data[][3])

{

int determinant;

int x,y,z;

x=data[0][0]\*cofactor(data,0,0);

y=data[0][1]\*cofactor(data,0,1);

z=data[0][2]\*cofactor(data,0,2);

determinant=x+y+z;

return determinant;

}

// returns greatest common denominator

int GCD(int a, int b)

{

while(true)

{

a = a % b;

if( a == 0 )

return b;

b = b % a;

if( b == 0 )

return a;

}

}

//outputs 3x3 square matrix

void writesqMatrix(int matrix[][3])

{

for(int i=0; i<3; i++)

{

for(int j=0; j<3; j++)

cout<<setw(20)<<matrix[i][j];

cout<<endl<<endl;

}

}

