**Name: Toshan Luktuke Date: 25-01-22**

**SRA ID: 007**

**Output:**

**Text

Description automatically generated**

**Code:**

#include <bits/stdc++.h>

using namespace std;

void display(vector<vector<double>> &v);

void display(vector<double> &v);

void display(vector<int> &v);

vector<vector<double>> rref(vector<vector<double>> v);

vector<vector<double>> rref(vector<vector<double>> mat, vector<vector<double>> b);

vector<double> xparticular(vector<vector<double>> &mat);

vector<vector<double>> xnull(vector<vector<double>> &mat);

bool zero\_row(vector<double> &v);

bool check\_form(vector<vector<double>> &mat);

void exchange\_row(vector<vector<double>> &mat, int r1, int r2);

vector<double> leading\_zeros(vector<vector<double>> &mat);

void arrange\_in\_descending(vector<double> &z, vector<vector<double>> &mat);

void arrange\_in\_descending(vector<double> &z, vector<vector<double>> &mat, vector<vector<double>> &b);

void divide\_row(vector<vector<double>> &mat, int val, int row);

void subtract\_rows(vector<vector<double>> &mat, int row1, double factor, int row2);

void reduce\_col(vector<vector<double>> &mat, int val, int row, int col);

void reduce\_col(vector<vector<double>> &mat, vector<vector<double>> &b, int val, int row, int col);

double roundoff(double value, unsigned char prec);

//global vars for rows, cols

int m, n;

int main()

{

    cout << "For a given equation Ax = b" << endl;

    cout << "Input the dimensions of the matrix A (mxn)" << endl;

    cin >> m >> n;

    vector<vector<double>> mat(m, vector<double>(n));

    cout << "Enter the values of the matrix: ";

    for (int i = 0; i < m; i++)

    {

        for (int j = 0; j < n; j++)

        {

            cin >> mat[i][j]; //{1,2,3} ->

                              //{4,5,6} :down:

        }

    }

    display(mat);

    cout << "Input the matrix b of Ax = b" << endl;

    vector<vector<double>> b(m, vector<double>(1));

    for (int i = 0; i < m; i++)

    {

        cin >> b[i][0]; //{1,2,3} ->

                        //{4,5,6} :down:

    }

    //get Row-Reduced Echelon Form

    vector<vector<double>> rref\_mat = rref(mat);

    vector<vector<double>> aug\_mat = rref(mat, b);

    // Get particular solution

    vector<double> particular = xparticular(aug\_mat);

    // Get Nullspace

    vector<vector<double>> nullspace = xnull(rref\_mat);

    cout << "Row-Reduced Echelon Form: -" << endl;

    display(rref\_mat);

    cout << "Particular Solution: -" << endl;

    display(particular);

    cout << "Nullspace: -" << endl;

    display(nullspace);

    cout << "Complete Solution is Particular(Xp) + Nullspace(Xn): -" << endl;

    return 0;

}

void display(vector<double> &v)

{

    cout << "[ ";

    for (int i = 0; i < v.size(); i++)

    {

        cout << v[i] << " ";

    }

    cout << "]" << endl;

}

void display(vector<int> &v)

{

    cout << "[ ";

    for (int i = 0; i < v.size(); i++)

    {

        cout << v[i] << " ";

    }

    cout << "]" << endl;

}

void display(vector<vector<double>> &v)

{

    cout << "Matrix is being displayed:-" << endl;

    cout << "----------" << endl;

    for (int i = 0; i < v.size(); i++)

    {

        for (int j = 0; j < v[i].size(); j++)

        {

            cout << v[i][j] << " ";

        }

        cout << endl;

    }

    cout << "----------" << endl;

}

vector<double> xparticular(vector<vector<double>> &mat)

{

    //Find all the pivots and their positions

    vector<int> piv\_pos;

    vector<double> piv\_vals(n, 0);

    for (int i = 0; i < mat.size(); i++)

    {

        for (int j = 0; j < mat[i].size(); j++)

        {

            if (mat[i][j] == 1)

            {

                piv\_pos.push\_back(j);

                break;

            }

        }

    }

    //Go from bottom to top assigning values

    for (int i = mat.size() - 1; i >= 0; i--)

    {

        // Get each row in reverse-order

        vector<double> co\_eff\_row = mat[i];

        //Check if this is a zero row

        if (zero\_row(co\_eff\_row))

        {

            continue;

        }

        int preserve = piv\_pos.back();

        double w\_sum = 0;

        for (int j = 0; j < co\_eff\_row.size() - 1; j++)

        {

            if (j == preserve)

            {

                continue;

            }

            w\_sum = co\_eff\_row[j] \* piv\_vals[j];

        }

        piv\_vals[piv\_pos.back()] = co\_eff\_row.back() - w\_sum;

        piv\_pos.pop\_back();

    }

    //display(piv\_vals);

    return piv\_vals;

}

vector<vector<double>> xnull(vector<vector<double>> &mat)

{

    // Identify the pivots and their positions

    // Identify the free variables and their positions

    vector<int> piv\_pos;

    vector<int> free\_vars(n, 1);

    vector<double> piv\_vals(n, 0);

    for (int i = 0; i < mat.size(); i++)

    {

        for (int j = 0; j < mat[i].size(); j++)

        {

            if (mat[i][j] == 1)

            {

                piv\_pos.push\_back(j);

                break;

            }

        }

    }

    int no\_free\_vars = n - piv\_pos.size();

    //Get the positions of free vars

    for (int i = 0; i < piv\_pos.size(); i++)

    {

        free\_vars[piv\_pos[i]] = 0;

    }

    display(piv\_pos);

    display(free\_vars);

    //Create a DD vector with dimensions m x no\_free\_vars

    vector<vector<double>> nullspace(no\_free\_vars, vector<double>(n));

    int p = 0;

    //Going by each free variable

    //Fill 1's at the nullspace at the index of each corresponding variable

    for (int i = 0; i < free\_vars.size(); i++)

    {

        if (free\_vars[i] == 0)

        {

            continue;

        }

        // at index of free var it will be 1 and of other free vars it will be 0

        nullspace[p][i] = 1;

        for (int j = 0; j < free\_vars.size(); j++)

        {

            if (j == i)

            {

                continue;

            }

            if (free\_vars[j] == 1)

            {

                nullspace[p][j] = 0; //Corresponding nullspace is 0

            }

        }

        // Iterate through pivot rows of the main matrice determine the other co-effs

        // Iterate columnwise

        for (int j = 0; j < piv\_pos.size(); j++)

        {

            //Kaafi time ke baad ye result aaya

            nullspace[p][piv\_pos[j]] = mat[j][i] == 0 ? mat[j][i] : -mat[j][i];

            //The ternary operator is to prevent a -ve 0

        }

        p++;

    }

    // display(nullspace);

    return nullspace;

}

bool zero\_row(vector<double> &v)

{

    bool ans = true;

    for (int i = 0; i < v.size(); i++)

    {

        if (v[i] != 0)

        {

            ans = false;

            break;

        }

    }

    return ans;

}

vector<vector<double>> rref(vector<vector<double>> mat)

{

    // 1. Format the matrix correctly, all columns with leading zeroes should be at the bottom

    vector<double> no\_of\_leading\_zeros = leading\_zeros(mat);

    arrange\_in\_descending(no\_of\_leading\_zeros, mat);

    // 2. Get the first non-zero element of each row -> convert to 1 -> Subtract from rows below

    for (int i = 0; i < m; i++)

    {

        for (int j = 0; j < n; j++)

        {

            if (mat[i][j] != 0.0 || mat[i][j] != -0)

            {

                //cout << mat[i][j] << endl;

                // Get the leading value

                double val = mat[i][j];

                // Divide whole row by this value thereby converting leader into 1

                divide\_row(mat, val, i);

                // Reduce all the rows above and below this element

                // Current element's column become 0

                reduce\_col(mat, val, i, j);

                break;

            }

        }

        // Re-format the matrix

        no\_of\_leading\_zeros = leading\_zeros(mat);

        arrange\_in\_descending(no\_of\_leading\_zeros, mat);

    }

    return mat;

}

vector<vector<double>> rref(vector<vector<double>> mat, vector<vector<double>> b)

{

    // 1. Format the matrix correctly, all columns with leading zeroes should be at the bottom

    vector<double> no\_of\_leading\_zeros = leading\_zeros(mat);

    arrange\_in\_descending(no\_of\_leading\_zeros, mat, b);

    // 2. Get the first non-zero element of each row -> convert to 1 -> Subtract from rows below

    for (int i = 0; i < m; i++)

    {

        for (int j = 0; j < n; j++)

        {

            if (mat[i][j] != 0.0 || mat[i][j] != -0)

            {

                //cout << mat[i][j] << endl;

                // Get the leading value

                double val = mat[i][j];

                // Divide whole row by this value thereby converting leader into 1

                divide\_row(mat, val, i);

                divide\_row(b, val, i);

                // Reduce all the rows above and below this element

                // Current element's column become 0

                reduce\_col(mat, b, val, i, j);

                break;

            }

        }

        // Re-format the matrix

        no\_of\_leading\_zeros = leading\_zeros(mat);

        arrange\_in\_descending(no\_of\_leading\_zeros, mat, b);

    }

    //return the augmented matrix

    for (int i = 0; i < mat.size(); i++)

    {

        mat[i].push\_back(b[i][0]);

    }

    return mat;

}

bool check\_form(vector<vector<double>> &mat)

{

    //For every row, the leading element has to be 1

    //Everything above and below the leading element has to be 1

    bool ans = true;

    for (int i = 0; i < m; i++)

    {

        for (int j = 0; j < n; j++)

        {

            if (mat[i][j] != 0 && mat[i][j] == 1)

            {

                //Check above and below if all are 1

                for (int k = 0; k < n; k++)

                {

                    if (mat[i][k] != 0)

                    {

                        ans = false;

                        break;

                    }

                }

                break;

            }

            else if (mat[i][j] != 0 && mat[i][j] != 1)

            {

                ans = false;

                break;

            }

        }

    }

    return ans;

}

void reduce\_col(vector<vector<double>> &mat, int val, int row, int col)

{

    for (int i = 0; i < m; i++)

    {

        if (i != row)

        {

            //Subtract all columns by the required values

            double factor = mat[i][col];

            //cout << "Factor: " << factor << endl;

            subtract\_rows(mat, i, factor, row);

        }

    }

}

void reduce\_col(vector<vector<double>> &mat, vector<vector<double>> &b, int val, int row, int col)

{

    for (int i = 0; i < mat.size(); i++)

    {

        if (i != row)

        {

            //Subtract all columns by the required values

            double factor = mat[i][col];

            //cout << "Factor: " << factor << endl;

            subtract\_rows(mat, i, factor, row);

            subtract\_rows(b, i, factor, row);

        }

    }

}

void subtract\_rows(vector<vector<double>> &mat, int row1, double factor, int row2)

{

    for (int j = 0; j < mat[row1].size(); j++)

    {

        mat[row1][j] -= (factor \* mat[row2][j]);

        //cout << "Subtrahend: " << (factor \* mat[row2][j]) << "," << mat[row2][j] << " Factor: " << factor << endl;

    }

}

void exchange\_row(vector<vector<double>> &mat, int r1, int r2)

{

    vector<double> temp = mat[r2];

    mat[r2] = mat[r1];

    mat[r1] = temp;

}

vector<double> leading\_zeros(vector<vector<double>> &mat)

{

    vector<double> no\_of\_leading\_zeros(m, 0);

    //get the no of leading zeros in every row

    for (int i = 0; i < m; i++)

    {

        for (int j = 0; j < n; j++)

        {

            if (mat[i][j] == 0)

            {

                no\_of\_leading\_zeros[i]++;

            }

            else

            {

                break;

            }

        }

    }

    return no\_of\_leading\_zeros;

}

void arrange\_in\_descending(vector<double> &z, vector<vector<double>> &mat)

{

    //Selection sort

    int min = z[0];

    for (int i = 0; i < z.size(); i++)

    {

        min = i;

        for (int j = i + 1; j < z.size(); j++)

        {

            if (z[j] < z[min])

            {

                min = j;

            }

        }

        double temp = z[i];

        z[i] = z[min];

        z[min] = temp;

        exchange\_row(mat, i, min);

    }

    // for (int i = 0; i < m; i++)

    // {

    //     cout << v[i] << " " << endl;

    // }

}

void arrange\_in\_descending(vector<double> &z, vector<vector<double>> &mat, vector<vector<double>> &b)

{

    //Selection sort

    int min = z[0];

    for (int i = 0; i < z.size(); i++)

    {

        min = i;

        for (int j = i + 1; j < z.size(); j++)

        {

            if (z[j] < z[min])

            {

                min = j;

            }

        }

        double temp = z[i];

        z[i] = z[min];

        z[min] = temp;

        exchange\_row(mat, i, min);

        exchange\_row(b, i, min);

    }

    // for (int i = 0; i < m; i++)

    // {

    //     cout << v[i] << " " << endl;

    // }

}

void divide\_row(vector<vector<double>> &mat, int val, int row)

{

    for (int j = 0; j < mat[row].size(); j++)

    {

        mat[row][j] = mat[row][j] / val;

        if (mat[row][j] < 0.001 && mat[row][j] > -0.001)

        {

            mat[row][j] = 0.0;

        }

    }

}

double roundoff(double value, unsigned char prec)

{

    double pow\_10 = pow(10.0f, (double)prec);

    return round(value \* pow\_10) / pow\_10;

}