선형대수 006분반 프로젝트 1 보고서

- 조 번호 : 3팀
- 조원 : 김다영(12201856) / 김민겸(12201863) / 박용민(12171786) / 왕인성(12171806)

C++을 이용한 Gaussian Elimination 프로그램 구현

C++ 프로그래밍 언어를 이용해 행렬에서 가우스 소거법을 구현하는 프로그램을 만들었습니다. 프로그램 구현 과정은 다음 네 단계로 분류해 설명할 수 있습니다.

① vector 헤더파일을 선언, vector를 이용해 이차원 배열(행렬)을 만들고 Augmented 행렬의 각 행 내용을 입력 받아 값을 할당한다.

② 가우스 소거법을 프로그램으로 구현하여 입력한 Augumented 행렬을 Row Echelon Form으로 만든다.

③ 만들어진 Row Echelon Form에 가우스 조던 소거법을 적용해 Reduced Row Echelon Form으로 만든다.

④ 만들어진 Reduced Row Echelon Form으로, 세 가지 케이스에 대한 해답을 출력하는 프로그램을 구현한다.

프로그램 소스 코드

```
#include <iostream>
#include <iomanip>
#include <vector>

using namespace std;

#define mat vector<vector<double>>
#define row vector<double>

int zero_row(double arr[], int size); // 행이 0 으로 이루어진 행인지 확인해주는 함수
void swap_row(double arr1[], double arr2[], int size); // arr1 행과 arr2 행을 바꿔주는 함수
void sort_array(double** arr, int size1, int size2, int count[]); // 2 차원 배열을 leading-1 의 위치마다 정렬해주는 함수

int zero_row(row& r); // 영행렬 확인 함수
int zero_mat(mat& m); // 행렬 안에 있는 영행렬 갯수 반환
int index_lead(row& r); // leading-1 이 있는 위치 반환
int inconsistent(mat& m); // 해가 존재하지 않는지 확인
```

```
int consistent(mat& m); // 해가 존재하는지 확인
int result(mat& m); // 3 가지 케이스(유일 해 = 1, 해가 무수히 많다 = 2, 해x = 3) 분류
void print(mat& m); // 행렬 반환함수
void sol(mat& m); // 해의 값을 반환
void unique(mat& m); // 유일해인경우, 해 반환
void infinite(mat& m); // 무수히 많은 경우, 해 반환
int only_lead(row& r);

int main()
{
    int eq_num = 0; // 식의 개수
    int var_num = 0; // 변수의 개수
    double pivot = 0; // 기준 변수

    cout << "연립 방정식의 식 개수를 입력하세요." << endl; // 식 개수 eq_num 입력받기
    cin >> eq_num;
    cout << "변수의 개수를 입력하세요." << endl; // 변수 개수 var_num 입력받기
    cin >> var_num;
    cout << "Augmented 행렬의 각 행의 내용을 차례대로 입력하세요." << endl; // 행렬의 각 원소들의 값 입력받기

    double** linear_eq = new double* [eq_num]; // 각 행렬의 내용 받을 2 차원 배열 선언
    for (int i = 0; i < eq_num; i++)
    {
        linear_eq[i] = new double[var_num + 1]; // 2 차원 배열 동적할당
    }

    for (int i = 0; i < eq_num; i++) {
        for (int j = 0; j < var_num + 1; j++) {
            cin >> linear_eq[i][j]; // 각 행렬 변수들 입력 받기
        }
    }

    cout << endl << "입력한 Augmented 행렬" << endl; // 입력받은 값 정렬 후 출력

    for (int i = 0; i < eq_num; i++) {
        for (int j = 0; j < var_num + 1; j++) {
            cout << setw(10) << linear_eq[i][j]; // 입력받은 값 출력
        }
        cout << endl;
    }

    int* leading = new int[eq_num]; // 각 행의 leading-1 의 위치를 저장할 배열 동적할당

    sort_array(linear_eq, eq_num, var_num, leading); // 정렬해주는 함수이자 정렬 후 각 행의 leading-1 의 위치 값을 배열에 넣어주는 함수

    cout << endl << "정렬한 행렬" << endl; // 입력받은 행렬을 정렬한 후 한번 더 출력
```

```
    for (int i = 0; i < eq_num; i++) {
        for (int j = 0; j < var_num + 1; j++) {
            cout << setw(10) << linear_eq[i][j]; // 입력받은 값 출력
        }
        cout << endl;
    }

    for (int i = 0; i < eq_num; i++) { // row echelon form
        sort_array(linear_eq, eq_num, var_num, leading); // 값이 바뀌면서 생기는 변수를 차단하기 위해 for 문 상단에서 한번씩 더 재정렬
        pivot = linear_eq[i][leading[i]]; // 기준 (leading-1 이 될 변수)
        if (zero_row(linear_eq[i], var_num) == 1) {
            for (int j = 0; j < var_num + 1; j++) {
                linear_eq[i][j] /= pivot; // 양변을 기준 변수로 나눠준다
                if (pivot < 0 && linear_eq[i][j] == 0) linear_eq[i][j] *= -1; // double data type 에서 0 에 음수를 곱했을 때 -0 이 출력되는 것을 방지
            }
        }
        for (int j = i + 1; j < eq_num; j++) {
            double lead = linear_eq[j][leading[i]]; // 각 행에서 그 다음 행에 뺄셈할 때 곱해줄 변수 (for 문을 돌면서 값이 바뀌는 것을 방지하기 위함)
            for (int k = i; k < var_num + 1; k++) {
                linear_eq[j][k] -= linear_eq[i][k] * lead; // 기준 변수의 계수를 각 행의 변수에 곱해준걸 밑에 식에 빼준다. (leading-1 과 같은 열에 있는 변수 0 으로 만들기)
            }
        }
    }

    sort_array(linear_eq, eq_num, var_num, leading); // row echelong form 을 만든 후 한번 더 정렬

    cout << "Row Echelon Form" << endl;

    for (int i = 0; i < eq_num; i++) {
        for (int j = 0; j < var_num + 1; j++) {
            cout << setw(10) << linear_eq[i][j]; // row echelon form 출력
        }
        cout << endl;
    }
    cout << endl;

    double** pivot_arr = new double* [eq_num]; // 각 열에서 reduced row echelon form 을 위해 곱해서 빼줄 변수들을 저장할 2 차원 배열 선언
    for (int i = 0; i < eq_num; i++)
    {
        pivot_arr[i] = new double[var_num + 1]; // 곱해줄 변수 2 차원 배열 동적할당
    }

    for (int i = 1; i < eq_num; i++) { // reduced row echelon form
        sort_array(linear_eq, eq_num, var_num, leading); // 값이 바뀌면서 생기는 변수를 차단하기 위해 for 문 상단에서 한번씩 더 재정렬
        for (int a = 0; a < eq_num; a++) {
```

```
            for (int b = 0; b < var_num + 1; b++) {
                pivot_arr[a][b] = linear_eq[a][b]; // 각 열에서 그 다음 열에 뺄셈할 때 곱해줄 변수 (for 문을 돌면서 값이 바뀌는 것을 방지하기 위함)
            }
        }
        for (int j = 0; j < i; j++) {
            for (int k = 0; k < var_num + 1; k++) {
                linear_eq[j][k] -= linear_eq[i][k] * pivot_arr[j][leading[i]]; // 각 열의 변수에 기준 변수의 계수를 곱해준걸 윗 행 변수에 빼준다.
            }
        }
    }

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

    sort_array(linear_eq, eq_num, var_num, leading); // reduced row echelon form 재정렬

    for (int i = 0; i < eq_num; i++) {
        for (int j = 0; j < var_num + 1; j++) {
            cout << setw(10) << linear_eq[i][j]; // 결과 출력
        }
        cout << endl;
    }

    mat input;
    for (int i = 0; i < eq_num; i++)
    {
        row r;
        for (int j = 0; j < var_num + 1; j++) // 열의 크기는 변수 개수 + 1 이므로 var + 1
        {
            r.push_back(linear_eq[i][j]);
        }
        input.push_back(r);
    }

    //print(input);

    sol(input);

    for (int i = 0;i < eq_num; i++) {
        delete[] linear_eq[i];
    }
    delete[] linear_eq;

    for (int i = 0;i < eq_num; i++) {
        delete[] pivot_arr[i];
    }
    delete[] pivot_arr;


    return 0;
}
```

```
int zero_row(row& r)
{
    for (int i = 0; i < r.size() - 1; ++i) // 0 부터 변수개수까지(행렬 마지막 열 제외)
    {
        if (r[i] != 0)
        {
            return -1; // 0 이 아닌 게 하나라도 있으면 -1 이 return
            // 영행렬이 아닌 것임
        }
    }
    return 0; // 0 이 return 되면 해당 행은 영행렬인 것임
    // 영행렬인지 확인
}

int zero_mat(mat& m)
{
    int count = 0; // 영행렬의 개수
    for (int i = 0; i < m.size(); i++) // 0 부터 식개수-1 까지
    {
        if (zero_row(m[i]) == 0) // 만약 영행렬이면
        {
            count++; // 영행렬을 제외한 식 개수 세기
        }
    }
    return count;
}

int index_lead(row& r) // leading-1 이 처음 나올 때의 인덱스를 반환하는 함수
// 1 이 처음으로 나오면 그것은 leading-1 인 것임
{
    for (int i = 0; i < r.size() - 1; i++) // 0 번째 열부터 (마지막 열-1)의 범위에서 (마지막 열은 방정식의 상수값이므로 확인 할 필요 없음)
    {
        if (r[i] == 1) // 처음 나온 1 에 대해 index 반환
        {
            return i;
        }
    }
    return 0;
}

int inconsistent(mat& m)
{
    for (int i = 0; i < m.size(); i++) // 0 부터 식개수-1 까지
    {
        if (zero_row(m[i]) == 0) // 만약 m 행렬의 j 행이 영행렬인데(마지막 열 제외)
        {
            if (m[i][m[0].size() - 1] != 0) // 행렬의 j 행(영행렬인 부분) 마지막열이 0 이 아니면
            {
                // 모순 발생
                return -1;
```

```
            }
        }
    }
    return 0;
}


int consistent(mat& m)
{
    // 영행렬이 아닌 나머지 식의 수 : m.size() - zero_mat(m)
    // 변수의 수 : m[0].size() - 1
    // [영행렬이 아닌 나머지 식의 수 < 변수의 수]이면 해가 존재하지 않게 됨
    if (m.size() - zero_mat(m) < m[0].size() - 1)
    {
        return 1; // 해가 무수히 많음
    }
    else
    {
        return 0; // 유일해(consistent 함수를 이용하기 전에 inconsistent 함수를
이용할 것이므로 해가 없는 경우는 이미 배제 됨)
    }
}

int result(mat& m)
{
    if (inconsistent(m) == -1) // 반환값이 -1 이면 해당 reduced row echelon
form 의 해가 존재하지 않는 것
    {
        return 3; // 해가 없는 경우에는 3 을 반환
    }
    else
    {
        if (consistent(m) == 1) // consistent 의 반환값이 1 이면 해가 무수히 많은
경우임
        {
            return 2; // 해가 무수히 많은 경우에는 2 를 반환
        }
        else //consistent == 0 인 경우(유일해)
        {
            return 1; // 유일해인 경우에는 1 을 반환
        }
    }
}

void print(mat& m)
{
    for (int i = 0; i < m.size(); i++)
    {
        for (int j = 0; j < m[0].size(); j++)
        {
            cout << m[i][j] << " ";
        }
        cout << endl;
    }
}
```

```
void sol(mat& m)
{
    if (result(m) == 1) // return 값이 1 이면 유일해
    {
        cout << "\n 유일해이다" << endl;
        unique(m); // 유일해일 때의 해의 값을 출력하기 위해 unique 함수 이용
    }
    else if (result(m) == 2) // return 값이 2 이면 무수히 많은 해
    {
        cout << "\n 해가 무수히 많다" << endl;
        infinite(m); // 해가 무수히 많을 때의 해의 값을 출력하기 위해 infinite 함수
이용
    }
    else // return 값이 3 이면 해 x
    {
        cout << "\n 해가 존재하지 않는다" << endl;
    }
}

void unique(mat& m) // 유일해인 경우 해를 출력
{
    for (int i = 0; i < m[0].size() - 1; i++)
    {
        cout << "X" << i + 1 << " = " << m[i][m[0].size() - 1] << endl;
    }
}

void infinite(mat& m) // 해가 무수히 많은 경우 해를 출력
{
    if (zero_mat(m) == m.size())
    {
        for (int i = 0; i < m[0].size() - 1; i++)
        {
            cout << "X" << i + 1 << " = " << "t" << i + 1 << endl;
        }
        return;
    }
    int* arr = new int[m[0].size() - 1]; // leading/free 인지 구분하기 위한 배열
생성
    for (int i = 0; i < m[0].size() - 1; i++)
    {
        arr[i] = 0; // 배열을 0 으로 초기화
    }
    for (int i = 0; i <= m.size() - zero_mat(m) - 1; i++) // 영행렬이 아닌 마지막
식(m.size() - zero_mat(m) - 1)에서 부터 첫 번째 행까지 역순으로 올라가며 해를 구함
    {
        arr[index_lead(m[i])] = 1; // leading 1 이 존재하는 변수는 leading
변수이므로 1 로 바꿔줌
        cout << "X" << (index_lead(m[i]) + 1) << " = "; // 변수 출력
        if (m[i][m[0].size() - 1] != 0 || only_lead(m[i]) == 0) /// 등호(=)
오른쪽의 값이 0 이면 출력할 필요 없으므로 출력 x, 등호 오른쪽 값이 0 인 경우 해당 행에
leading-1 하나만 존재하면 0 을 출력해야됨
        {
```

```
            cout << m[i][m[0].size() - 1];
        }
        //i 행에서 leading-1 인 변수의 값을 구함
        //m[i][m[0].size() - 1]은 해당 행의 맨 마지막 열에 위치하는 상수 값
        for (int j = index_lead(m[i]) + 1; j < m[0].size() - 1; j++)
        {
            if (m[i][j] == 0) // m[i][j]가 0 이면 출력할 필요가 없으므로 출력x
            {
                continue;
            }
            if (m[i][j] < 0 && m[i][m[0].size() - 1] != 0) // m[i][j]의 값이
0 보다 작은 경우 이항을 통해 부호가 +가 되므로 따로 +를 출력
             //m[i][j]가 >0 인 경우에는 이항 되면서 (-) 부호가 붙으므로 따로 출력해줄
필요x
            {
                cout << "+";
            }
            cout << -m[i][j] << "*t" << (j + 1); // m[i][j]에 (-)를 붙인 값(이항
되면서 (-)가 붙음)과 j+1 번째 변수의 곱(*) 출력
        }
        cout << endl;
    }
    for (int i = 0; i < m[0].size() - 1; i++) // free var 출력
    {
        if (arr[i] == 0)
        {
            cout << "X" << i + 1 << " = " << "t" << i + 1 << endl;
        }
    }
    delete[] arr;
}

int zero_row(double arr[], int size) { // 배열에서 각 행에 0 으로 이루어진 행이있나
체크해주는 함수
    for (int i = 0; i < size; i++) {
        if (arr[i] != 0) return 1; // 0 이 아닌 값이 하나라도 나오면 1 을 반환
    }
    return 0; // 0 인 행이면 0 을 반환
}

void swap_row(double arr1[], double arr2[], int size) { // 각 행의 위치를 서로
바꿔주는 함수
    double* alt = new double[size]; // 중간에 바뀔 값을 저장해주는 배열 동적할당
    for (int n = 0; n < size; n++) {
        alt[n] = arr1[n];
        arr1[n] = arr2[n];
        arr2[n] = alt[n];
    }
}
// sort_array(linear_eq, eq_num, var_num, leading);

void sort_array(double** arr, int size1, int size2, int count[]) { // 배열의
순서를 정렬해주는 함수이자 각 행의 leading-1 의 위치를 저장해주는 함수
    for (int i = 0; i < size1; i++) count[i] = 0;
```

```
    int colCount = 0; // 열 카운터
    int rowCount = 0; // 행 카운터
    while (rowCount < size1) { // 열 카운터가 식의 개수보다 작을 때 실행
        colCount = 0; // 행 카운터 초기화
        while (colCount < size2) { // 행 카운터가 변수의 개수보다 작을 때 실행
            if (arr[rowCount][colCount] != 0) break; // 각 변수가 0 이 아니면
break, 0 이면 다음 변수로 넘어감 (즉, 0 의 개수를 세어줌)
            colCount++;
            count[rowCount]++;
        }
        rowCount++;
    }
    int temp = 0;
    int min_num = 0; //선택정렬을 위한 최솟값 인덱스를 만든다
    for (int j = 0; j < size1; j++) {
        min_num = j; //현 위치를 최솟값 인덱스로 우선 초기화
        for (int k = j + 1; k < size1; k++) {
            if (count[min_num] > count[k]) min_num = k; //가장 작은수의 인덱스를
저장
        }
        swap_row(arr[j], arr[min_num], size2 + 1); // 각 행의 위치 바꿔주기
        temp = count[j]; // 행의 위치가 바뀜에 따라 같이 바뀌어야할 leadding-1 의
값도 temp 를 활용하여 위치 바꾸기
        count[j] = count[min_num];
        count[min_num] = temp;
    }
}

int only_lead(row& r) // 해당 행에 leading 1 변수만 있는지를 확인하기 위한 함수
{
    for (int j = index_lead(r) + 1; j < r.size() - 1; j++)
    {
        if (r[j] != 0) // lead-1 인 변수 뒤에 하나라도 0 이 아닌 수가 있으면 leading
1 만 있는 게 x
        {
            return -1;
        }
    }
    return 0;
}
```

풀이 예제

(1) Case1(유일해 존재)

```
연립 방정식의 식 개수를 입력하세요.
3
변수의 개수를 입력하세요.
3
Augmented 행렬의 각 행의 내용을 차례대로 입력하세요.
1 1 2 2
1 2 -1 -3
2 -1 1 7

입력한 Augmented 행렬
         1         1         2         2
         1         2        -1        -3
         2        -1         1         7

정렬한 행렬
         1         1         2         2
         1         2        -1        -3
         2        -1         1         7
Row Echelon Form
         1         1         2         2
         0         1        -3        -5
         0         0         1         1

Reduced Row Echelon Form
         1         0         0         2
         0         1         0        -2
         0         0         1         1

유일해이다
X1 = 2
X2 = -2
X3 = 1
```

(2) Case2(무수히 많은 해 존재)

```
연립 방정식의 식 개수를 입력하세요.
3
변수의 개수를 입력하세요.
4
Augmented 행렬의 각 행의 내용을 차례대로 입력하세요.
1 0 8 -5 6
0 1 4 -9 3
0 0 1 1 2

입력한 Augmented 행렬
         1         0         8        -5         6
         0         1         4        -9         3
         0         0         1         1         2

정렬한 행렬
         1         0         8        -5         6
         0         1         4        -9         3
         0         0         1         1         2
Row Echelon Form
         1         0         8        -5         6
         0         1         4        -9         3
         0         0         1         1         2

Reduced Row Echelon Form
         1         0         0       -13       -10
         0         1         0       -13        -5
         0         0         1         1         2

해가 무수히 많다
X1 = -10+13*t4
X2 = -5+13*t4
X3 = 2-1*t4
X4 = t4
```

(3) Case3(해가 존재하지 않음)

```
연립 방정식의 식 개수를 입력하세요.
3
변수의 개수를 입력하세요.
3
Augmented 행렬의 각 행의 내용을 차례대로 입력하세요.
1 2 -3 4
3 -1 5 2
4 1 2 -2

입력한 Augmented 행렬
         1         2        -3         4
         3        -1         5         2
         4         1         2        -2

정렬한 행렬
         1         2        -3         4
         3        -1         5         2
         4         1         2        -2
Row Echelon Form
         1         2        -3         4
         0         1        -2   1.42857
         0         0         0        -8

Reduced Row Echelon Form
         1         0         1   10.2857
         0         1        -2   12.8571
         0         0         0        -8

해가 존재하지 않는다
```