선형대수 HW3

학번: 2017041930 이름: 김선동

HW#3. Gram-Schmidt Orthogonalization 구현하기

구하는 알고리즘은 아래 수식과 같다. 주어진 벡터로 orthogonal basis인 $u_1 \dots u_k$를 구하고 그를 normalize하여 orthonormal basis인 $q_1..q_k$를 구해주면 된다. 구현은 c++언어로 하였으며 컴파일은 visual c++ 중 제일 최신버전으로 하였다.

$$\mathbf{u}_1 = \mathbf{v}_1, \qquad q_1 = \frac{\mathbf{u}_1}{\|\mathbf{u}_1\|}$$

$$\mathbf{u}_2 = \mathbf{v}_2 - \operatorname{proj}_{\mathbf{u}_1}(\mathbf{v}_2), \qquad q_2 = \frac{\mathbf{u}_2}{\|\mathbf{u}_2\|}$$

$$\mathbf{u}_3 = \mathbf{v}_3 - \operatorname{proj}_{\mathbf{u}_1}(\mathbf{v}_3) - \operatorname{proj}_{\mathbf{u}_2}(\mathbf{v}_3), \qquad q_3 = \frac{\mathbf{u}_3}{\|\mathbf{u}_3\|}$$

$$\mathbf{u}_4 = \mathbf{v}_4 - \operatorname{proj}_{\mathbf{u}_1}(\mathbf{v}_4) - \operatorname{proj}_{\mathbf{u}_2}(\mathbf{v}_4) - \operatorname{proj}_{\mathbf{u}_3}(\mathbf{v}_4), \qquad q_4 = \frac{\mathbf{u}_4}{\|\mathbf{u}_4\|}$$

$$\vdots \qquad\qquad \vdots$$

$$\mathbf{u}_k = \mathbf{v}_k - \sum_{j=1}^{k-1} \operatorname{proj}_{\mathbf{u}_j}(\mathbf{v}_k), \qquad q_k = \frac{\mathbf{u}_k}{\|\mathbf{u}_k\|}$$

1. 프로그램 Input 양식과 Outputs

```
Input the number of dimension of vector and the number of vectors
```

프로그램을 실행하면, 위와 같이 vector의 dimension과 vector의 개수를 입력하라고 지시한다.
이 때, 순서는 반드시 vector의 dimension을 먼저 입력해주어야 한다.
또한 과제 조건으로, vector의 수가 차원보다 크거나 같기에 뒤에 입력할 숫자를 조건에 맞게 입력해야 한다.
만약 vector의 dimension이 3이고 vector의 개수가 3이라면 아래와 같이 입력하면 된다.

```
3 3
Input the elements of the vectors
```

그 다음으로는 앞서 입력한 size에 맞게 vector의 값들을 차례로 입력해주면 된다.
만약 v_1 = [-3, -1, 1], v_2 = [2, 4, -3], v_3 = [1, 1, 1]이라면 아래와 같이 입력해주면 된다.

```
Input the elements of the vectors
-3 -1 1
2 4 -3
1 1 1

q1 vector: -0.904534 -0.301511 0.301511
q2 vector: -0.418511 0.763167 -0.492366
q3 vector: 0.0816497 0.571548 0.816497
Verify that Q vectors are orthonormal
Verified
```

그 후, 연산 결과값들이 위와 같이 출력된다.
Orthonormal basis vector들이 순서대로 출력된다.
또한, 그 결과가 맞는 지 검증하기 위해 내적이 0인지 확인하는 작업을 거쳐 만약 0이라면 "Verified"를 출력하고
그렇지 않다면 "Wrong answer"를 출력한다.

1. 프로그램 Input 양식과 Outputs

- 과제에서 주어진 vector의 dimension의 범위는 2이상 10이하이며, 개수는 dimension값 이상이면 된다.
- Orthonormal vector basis를 담는 2차원 vector container는 최대 100개의 벡터를 저장할 수 있게 하였다.
- 상한이 따로 조건에 제시되어 있지 않아서 최대 100으로 임의로 결정하였다.
- 이에 따라, input에 있어서 만일 vector의 개수가 100을 초과한다면 저장하지 못할 수 있다.
- 허나 구한 q 벡터들은 orthonormal하지 않을 것이므로 개수의 큰 의미는 없다고 생각한다.

2. Code Summary

1번 슬라이드에서 살펴본 알고리즘을 코드로 구현해보자.
Gram-Schmidt 함수를 보기 전에 기본적인 함수들을 구현해주었다.
임의의 vector들을 받고 그들을 출력하는 함수(①), 정규화하는 함수(②), 내적하는 함수(③)들이다.

① 벡터들을 출력해주는 함수

```
void printVector(vector<vector<double>> v, int n, int numOfVecters)
{
    for (int i = 0; i < numOfVecters; i++)
    {
        cout << "q" << i+1 << " vector: ";
        for (int j = 0; j < n; j++)
        {
            cout << v[i][j] << " ";
        }
        cout << "₩n";
    }
}
```

② 벡터를 정규화해주고 반영해주는 함수

```
void normalize(vector<vector<double>>& v, int idx)
{
    double sum = 0;
    for (int i = 0; i < v.size(); i++)
    {
        sum += pow(v[idx][i], 2);
    }
    double lengthOfVector = sqrt(sum);
    for (int i = 0; i < v.size(); i++)
    {
        v[idx][i] /= lengthOfVector;
    }
}
```

③ 벡터들을 내적해주는 함수

```
double dotProduct(vector<double> v1, vector<double> v2, int n)
{
    double value = 0;
    for (int i = 0; i < n; i++)
    {
        value += (v1[i] * v2[i]);
    }
    return value;
}
```

· Vector들의 저장은 vector container를 사용하였으며 2D vector를 사용하여 input vector들을 관리해주었다.

이제 Gram-Schmidt 함수(④)를 살펴보자.

```
void gramSchmidt(vector<vector<double>> &q, const vector<vector<double>> A, int n, int numOfVecters)
{
    // q0 얻음
    normalize(q, 0);
    // q1부터 qn구해야 함
    for (int i = 1; i < numOfVecters; i++)
    {
        //qi에서 뺄값 계산
        for (int j = 0; j < i; j++)
        {
            double coefficient = dotProduct(q[j], A[i], n);
            vector<double> tmp;
            for (int k = 0; k < n; k++)
            {
                tmp.push_back(q[j][k] * coefficient);
            }
            for (int j = 0; j < n; j++)
            {
                q[i][j] -= tmp[j];
            }
        }
        normalize(q, i);
    }
}
```

④ Gram-Schmidt 함수 설명

- 우선, 첫 번째 orthonormal basis vector인 q_1는 정규화만 해주면 되기 때문에 loop문 전에 해주었다.
(구현 편의상 인덱스는 0부터 사용함, 그러나 1번 슬라이드 알고리즘에 따라 설명하겠다.)

- 이제 q_2부터 q_n까지 구해주면 되는데 이 때, n은 user에게 받은 input이다.

- Vector의 개수만큼 연산을 해주어야 하기 때문에 제일 바깥의 for문은 numOfVectors만큼 반복한다.

- Orthogonal basis를 구하는 과정에서 원래 vector와 빼는 부분을 분리하여 계산해주었다.

- 이 때 매개변수로 들어온 q는 input vector들과 같은 값(원래 vector들의 값)을 갖고 있기 때문에 빼는 부분만을 계산해서 빼면 Orthogonal basis를 구할 수 있다.

- q_n을 구할 때, 필수적으로 q_{n-1}부터 q_1과 원래 vector인 a_n을 내적해주어야 하는데 이는 결과값이 상수이므로 dotProduct함수를 사용하여 coefficient 변수에 넣어주었다. For문에서 구한 coefficient들을 매번 해당하는 vector q와 곱해준 다음 tmp vector에 저장하고, 원래 vector들의 값을 담고 있는 q에서 tmp값들을 빼서 orthogonal basis들을 구해주었다.

- 각 iteration마다 normalize를 해주어 orthonormal basis vector를 구해주었다.

⑤ verifyOrthonormal 함수 설명

```
bool verifyOrthonormal(vector<vector<double>>& q, int num)
{
    int result;
    for (int i = 0; i < num; i++)
    {
        for (int j = 1; j < num; j++)
        {
            if (i == j)
                continue;
            result = dotProduct(q[i], q[j], num);
            if (result != 0)
                return false;
        }
    }
    return true;
}
```

```
bool verifyOrthonormal(vector<vector<double>>& q, int n, int numOfVectors)
{
    bool flag = true;
    double result;
    for (int i = 0; i < numOfVectors; i++)
    {
        for (int j = 1; j < numOfVectors; j++)
        {
            if (i == j || i > j)
                continue;
            result = dotProduct(q[i], q[j], n);
            cout << result << "Wn";
            if (abs(result) > EPS)
                flag = false;
        }
    }
    return flag;
}
```

위 함수는 구해진 orthonormal basis vector들을 내적하여 모두 0이 나오는지 확인하고, 모두 0이면 true를 반환하고 0이 아니면 false를 반환하는 함수이다.
Orthonormal basis vector들은 서로 직교하기 때문에 내적한 값은 0이어야 한다.
그러나, round-off error가 발생하여 내적이 0임에도 불구하고 0이 아닌 값으로 계산되어 직교하지 않는다고 판단하는 경우가 많아서 내적한 값이 일정한 값(매우 작은 수, 여기서는 1.0e-13으로 정함)보다 크면 직교하지 않는다고 판단하였다. 그 코드는 오른쪽과 같다.

⑥ main문 설명

```
int main()
{
    int n, numOfVecters;
    cout << "Input the number of dimension of vector and the number of vectors" << endl;
    cin >> n >> numOfVecters;                // numOfVectors >= n
    cout << "Input the elements of the vectors" << endl;
    for (int i = 0; i < numOfVecters; i++)
    {
        for (int j = 0; j < n; j++)
        {
            cin >> a[i][j];
        }
    }
    q = a;
    cout << "Wn";
    gramSchmidt(q, a, n, numOfVecters);
    printVector(q, n, numOfVecters);
    cout << "Verify that Q vectors are orthonormal" << endl;
    if (verifyOrthonormal(q, n))
    {
        cout << "Verified" << endl;
    }
    else
    {
        cout << "Wrong answer" << endl;
    }

}
```

- 먼저, vector의 dimension과 vector의 개수를 입력받는다.

- 그 후 vector의 개수와 차원만큼 원소를 입력받는다.

- 구하게 될 Orthonormal basis vector들을 담을 q는 어차피 원래 vector값에서 뺄셈이 이루어지기 때문에 input값을 복사해주었다.

- gramSchmidt함수를 통해 orthonormal basis vecto들을 구해 q에 넣어준다.

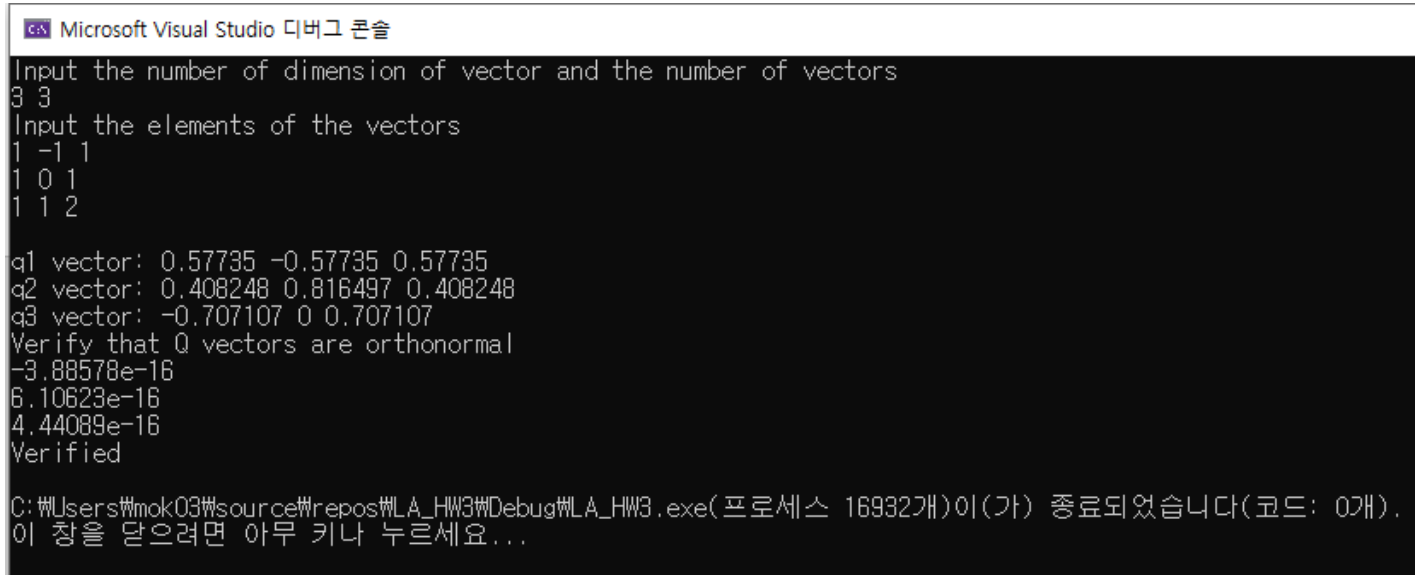
- q에 담긴 vector들을 출력해준다.

- verifyOrthonormal함수를 통해 orthonormal한지 아닌지 다시 검증해본다.

3. Examples - ①

$$A = \begin{bmatrix} 1 & 1 & 1 \\ -1 & 0 & 1 \\ 1 & 1 & 2 \end{bmatrix}$$

우선 3*3 system의 예시를 들어보자. (m == n)
만약 v_1 = [1, -1, 1], v_2 = [1, 0, 1], v_3 = [1, 1, 2]이라면 왼쪽과 같은 행렬을 이룬다.
이를 프로그램에 입력해보면 다음과 같은 결과가 나온다.



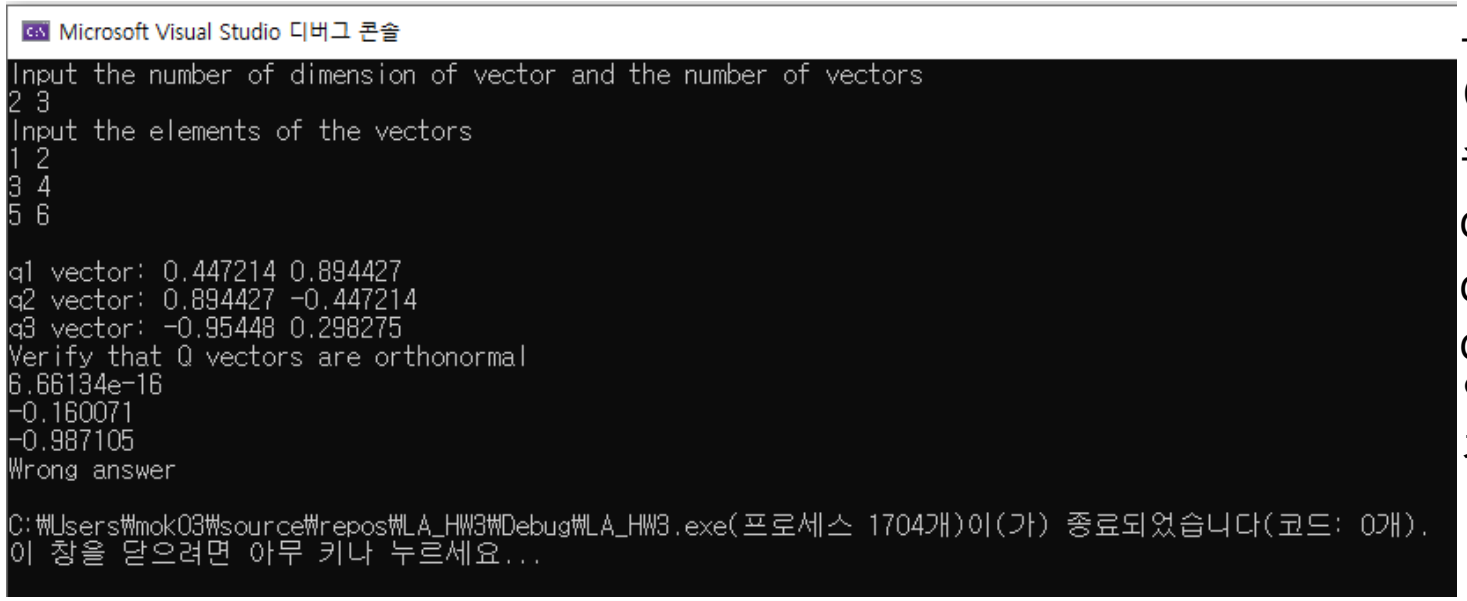
구한 orthonormal vector basis는 각각 q1, q2, q3이며 소수점 다섯 번째 자리까지 표현하고 있다.
손으로 구해보면
q1 = $(\sqrt{3}/3, -\sqrt{3}/3, \sqrt{3}/3)$
q2 = $(\sqrt{6}/6, \sqrt{6}/3, \sqrt{6}/6)$
q3 = $(-\sqrt{2}/2, 0, \sqrt{2}/2)$인데 여섯 번째자리까지 표현하면(반올림) 왼쪽의 결과와 같음을 확인할 수 있다.

이제 구한 vector들의 직교성을 증명해보자. 각 vector들을 내적하면 0이 나와야 한다.
q1과 q2, q1과 q3, q2와 q3의 내적이 세 줄에 걸쳐 출력된다. 이 값들이 직교함에도 0이 나오지 않는 이유는 round-off error 때문이다. 자릿수를 컴퓨터가 전부 표현할 수 없기 때문에 내림이나 올림, 반올림 등이 발생하며 결과적으로 결과값은 0이 나올 수 없다. 손으로 계산하면 0이 나온다. 그래서 결과값이 1.0e-13보다 작다면 0으로 처리하게끔 하여 Verified라고 출력된 것이다.

3. Examples - ②

$$A = \begin{bmatrix} 1 & 3 & 5 \\ 2 & 4 & 6 \end{bmatrix}$$

이제 m보다 n이 큰 경우를 살펴보자. 간단하게 2*3 system의 예시를 들어보자. (m < n)
만약 v_1 = [1, 2], v_2 = [3, 4], v_3 = [5, 6] 이라면 왼쪽과 같은 행렬을 이룬다.
이를 프로그램에 입력해보면 다음과 같은 결과가 나온다.



구한 orthonormal vector basis는 각각 q1, q2, q3이며 소수점 다섯 번째 자리까지 표현하고 있다.
손으로 구해보면

$q1 = (\sqrt{5}/5, -2\sqrt{5}/5)$

$q2 = (2\sqrt{5}/5, -\sqrt{5}/5)$

q3은 zero vector이외의 직교하는 벡터를 구할 수 없다. 여섯 번째자리까지 표현하면(반올림) 왼쪽의 결과와 같음을 확인할 수 있다.

이제 구한 vector들의 직교성을 증명해보자. 각 vector들을 내적하면 0이 나와야 한다.
q1과 q2, q1과 q3, q2와 q3의 내적이 세 줄에 걸쳐 출력된다. 앞서 정한 기준인 1.0e-13보다 매우 큰 값들이 내적 결과값으로 출력되었다. 이는 내적이 0이 아님을 나타내며 즉 직교하지 않는다는 것을 알 수 있다. 이를 뒷받침하는 근거로는 linear combination이 있다. 원래 행렬 A의 두 vector로 나머지 하나의 vector를 표현, 즉 선형조합으로 나타낼 수 있다면 이는 같은 평면에 위치하므로 직교하지 않는다는 것을 알 수 있다.

Example 2번의 A행렬을 분석해보자.
우리는 이 행렬의 두 vector를 뽑아서 나머지 하나의 vector를 표현할 수 있음을 알고 있다.
v1과 v2를 뽑아서 v3를 선형조합으로 나타내 보자. 위 식을 아래와 같이 나타낼 수 있다.

$$A = \begin{bmatrix} 1 & 3 & 5 \\ 2 & 4 & 6 \end{bmatrix}$$

$$\begin{aligned} a + 3b &= 5 \\ 2a + 4b &= 6 \end{aligned}$$

식을 풀어보면 a=-1, b=2를 구할 수 있다. 즉 v1에 -1을 곱한 것에 v2에 2를 곱한 것을 더하면 v3를 구할 수 있다는 것이다. 위와 같이 linear combination으로 나타내었으니 마지막 vector인 v3는 v1과 v2가 이루는 평면 위에 있는 vector라는 것을 알 수 있으며 마찬가지로, q1, q2, q3또한 직교하지 않음을 알 수 있다.

3. Examples - ③

선택 Microsoft Visual Studio 디버그 콘솔

```
Input the number of dimension of vector and the number of vectors
10 10
Input the elements of the vectors
1 2 3 4 5 6 7 8 9 10
11 12 13 14 15 16 17 18 19 20
21 22 23 24 25 26 27 28 29 30
31 32 33 34 35 36 37 38 39 40
41 42 43 44 45 46 47 48 49 50
51 52 53 54 55 56 57 58 59 60
61 62 63 64 65 66 67 68 69 70
71 72 73 74 75 76 77 78 79 80
81 82 83 84 85 86 87 88 89 90
91 92 93 94 95 96 97 98 99 100

q1 vector: 0.0509647 0.101929 0.152894 0.203859 0.254824 0.305788 0.356753 0.407718 0.458682 0.509647
q2 vector: 0.58554 0.48795 0.39036 0.29277 0.19518 0.09759 -2.42697e-16 -0.09759 -0.19518 -0.29277
q3 vector: 0.502459 0.502459 0.439652 0.314037 0.188422 0.0942111 0.125615 -0.0942111 -0.188422 -0.314037
q4 vector: -0.502459 -0.502459 -0.439652 -0.314037 -0.188422 -0.0942111 -0.125615 0.0942111 0.188422 0.314037
q5 vector: -0.502459 -0.502459 -0.439652 -0.314037 -0.188422 -0.0942111 -0.125615 0.0942111 0.188422 0.314037
q6 vector: -0.502459 -0.502459 -0.439652 -0.314037 -0.188422 -0.0942111 -0.125615 0.0942111 0.188422 0.314037
q7 vector: -0.502459 -0.502459 -0.439652 -0.314037 -0.188422 -0.0942111 -0.125615 0.0942111 0.188422 0.314037
q8 vector: -0.502459 -0.502459 -0.439652 -0.314037 -0.188422 -0.0942111 -0.125615 0.0942111 0.188422 0.314037
q9 vector: -0.502459 -0.502459 -0.439652 -0.314037 -0.188422 -0.0942111 -0.125615 0.0942111 0.188422 0.314037
q10 vector: -0.502459 -0.502459 -0.439652 -0.314037 -0.188422 -0.0942111 -0.125615 0.0942111 0.188422 0.314037
Verify that Q vectors are orthonormal
-3.60822e-16
0.0448135
-0.0448135
-0.0448135
-0.0448135
-0.0448135
-0.0448135
-0.0448135
-0.0448135
0.98683
-0.98683
-0.98683
-0.98683
-0.98683
-0.98683
-0.98683
-0.98683
-1
```

```
-1
-1
-1
-1
-1
-1
-1
1
1
1
1
1
1
1
1
1
1
1
1
1
1
1
1
1
1
1
1
Wrong answer

C:₩Users₩mok03₩source₩repos₩LA_HW3₩Debug₩LA_HW3.exe(프로세스 22056개)이(가) 종료되었습니다(코드: 0개).
이 창을 닫으려면 아무 키나 누르세요...
```

Example 3번의 A행렬을 분석해보자. 이 행렬은 10*10 system이다.
v1 = (1 2 3 4 5 6 7 8 9 10)
v2 = (11 12 13 14 15 16 17 18 19 20)
v3 = (21 22 23 24 25 26 27 28 29 30)
v4 = (31 32 33 34 35 36 37 38 39 40)
v5 = (41 42 43 44 45 46 47 48 49 50)
v6 = (51 52 53 54 55 56 57 58 59 60)
v7 = (61 62 63 64 65 66 67 68 69 70)
v8 = (71 72 73 74 75 76 77 78 79 80)
v9 = (81 82 83 84 85 86 87 88 89 90)
v10 = (91 92 93 94 95 96 97 98 99 100)
위의 10개 vector들을 입력해주었다. 결과값은 pg11과 같으며 내적이 0이 아니므로 직교하지 않는다는 것을 알 수 있다. 이제 linear combination으로도 증명을 해보자.
간단하게 v2에서 v1을 빼면 (10 10 10 10 10 10 10 10 10 10)이라는 vector를 얻을 수 있는데 이를 v2에서 하나 더할 때마다 다음 vector와 같게 된다. 즉, v1과 v2가 형성한 평면 위에 나머지 vector들이 있다.
결과적으로 v3, v4, v5, v6, v7, v8, v9, v10 모두 v1과 v2의 linear combination으로 표현 가능하다.

감사합니다!