Elementary Operation and Gram Schmidt Process

Elementary operations (row or column)
Elementary **row** operations include : (1) swapping two rows, (2) scaling a row, (3) adding a row with the multiple of another row, which can be performed by multiplying N × M matrix A (i.e. N row data in the M[th] dimensional space) by the following N × N elementary row operation matrices on the left (i.e. A' = EA, where A' is the result after elementary row operations).

Operation 1 : swap row k and row l

$$E_{swap}(k,l) = \begin{bmatrix} 1 & 0 & \dots & 0 & \dots & 0 & \dots & 0 & 0 \\ 0 & 1 & \dots & 0 & \dots & 0 & \dots & 0 & 0 \\ \dots & \dots & \dots & \dots & \dots & \dots & \dots & \dots & \dots \\ 0 & 0 & \dots & 0 & \dots & 1 & \dots & 0 & 0 \\ \dots & \dots & \dots & \dots & \dots & \dots & \dots & \dots & \dots \\ 0 & 0 & \dots & 1 & \dots & 0 & \dots & 0 & 0 \\ \dots & \dots & \dots & \dots & \dots & \dots & \dots & \dots & \dots \\ 0 & 0 & \dots & 0 & \dots & 0 & \dots & 1 & 0 \\ 0 & 0 & \dots & 0 & \dots & 0 & \dots & 0 & 1 \end{bmatrix} \quad \text{or} \quad E_{swap}(k,l)_{i,j} = \begin{bmatrix} 1 & & i = j \neq k,l \\ 0 & & i = j = k,l \\ 1 & i \neq j & i = k, j = l \\ 1 & i \neq j & i = l, j = k \\ 0 & i \neq j & othterwise \end{bmatrix}$$

Operation 2 : row k = s × row k

$$E_{scale}(k,s) = \begin{bmatrix} 1 & 0 & \dots & 0 & \dots & 0 \\ 0 & 1 & \dots & 0 & \dots & 0 \\ \dots & \dots & \dots & \dots & \dots & \dots \\ 0 & 0 & \dots & s & \dots & 0 \\ \dots & \dots & \dots & \dots & \dots & \dots \\ 0 & 0 & \dots & 0 & \dots & 1 \end{bmatrix} \quad \text{or} \quad E_{scale}(k,s)_{i,j} = \begin{bmatrix} 1 & i = j \neq k \\ s & i = j \neq k \\ 0 & i \neq j \end{bmatrix}$$

Operation 3 : row k = row k + s × row l **(Please don't confuse with : row l = row k + s × row l)**

$$E_{add}(k,l,s) = \begin{bmatrix} 1 & 0 & \dots & 0 & \dots & 0 & \dots & 0 & 0 \\ 0 & 1 & \dots & 0 & \dots & 0 & \dots & 0 & 0 \\ \dots & \dots & \dots & \dots & \dots & \dots & \dots & \dots & \dots \\ 0 & 0 & \dots & 1 & \dots & s & \dots & 0 & 0 \\ \dots & \dots & \dots & \dots & \dots & \dots & \dots & \dots & \dots \\ 0 & 0 & \dots & 0 & \dots & 1 & \dots & 0 & 0 \\ \dots & \dots & \dots & \dots & \dots & \dots & \dots & \dots & \dots \\ 0 & 0 & \dots & 0 & \dots & 0 & \dots & 1 & 0 \\ 0 & 0 & \dots & 0 & \dots & 0 & \dots & 0 & 1 \end{bmatrix} \quad \text{or} \quad E_{add}(k,l,s)_{i,j} = \begin{bmatrix} 1 & i = j & \\ s & i \neq j & i = k, j = l \\ 0 & i \neq j & \end{bmatrix}$$

Remark 1
By applying transpose on the above, we have :

$$\begin{aligned} A' &= EA \\ A'^T &= A^T E^T \\ \Rightarrow \quad B' &= BF \\ B &= A^T \quad \text{input matrix of N column data in M[th] dimensional space} \\ B' &= A'^T \quad \text{output matrix of N column data in M[th] dimensional space} \\ F &= E^T \quad \text{elementary column matrix (generated by transposing elementary row matrix)} \end{aligned}$$

Thus elementary row operation can be performed by multiplying matrix A by elementary row matrix on the left, while elementary column operation can be performed by multiplying matrix B by elementary column matrix on the right.

Remark 2
The above elementary row operations (and elementary column operations) can be cascaded to form a sequence of row or column operations, i.e. composition.

$$\begin{aligned} A' &= E_T E_{T-1} \dots E_3 E_2 E_1 A \\ A &= N \times M \text{ matrix, representing the input} && \text{i.e. N row data in dimension M} \\ A' &= N \times M \text{ matrix, representing the output} && \text{i.e. N row data in dimension M} \\ E_t &= N \times N \text{ matrix, representing the t[th] row operation, } \forall t \in [1,T] \end{aligned}$$

B' = $BF_1F_2F_3....F_{T-1}F_T$

B = $M \times N$ matrix, representing the input — i.e. N column data in dimension M

B' = $M \times N$ matrix, representing the output — i.e. N column data in dimension M

F_t = $N \times N$ matrix, representing the t^{th} column operation, $\forall t \in [1,T]$

The elementary row operations are applied to A in the order : E_1, E_2, E_3, … E_T, while the elementary column operations are applied to B in the order : F_1, F_2, F_3, … F_T, hence composition of elementary row operation is right associative (like the composition of function and composition of permutation), while composition of elementary column operation is left associative.

Remark 3

The inverse of elementary operation can be done by multiplying with the inverse of elementary matrices. The inverse of elementary matrices are also elementary matrices.

$E_{swap}(k,l)^{-1} = E_{swap}(k,l)$ remark : swapping of row k and row l can be reversed by swapping again)

$E_{scale}(k,s)^{-1} = E_{scale}(k,s^{-1})$ remark : scaling of row k by s can be reversed by scaling of row k by $1/s$

$E_{add}(k,l,s)^{-1} = E_{add}(k,l,-s)$ remark : adding multiple of row l to row k can be reversed by subtraction

Remark 4

All **N × N square matrices** can be represented as the result of applying : (1) a sequence of elementary row additions to an upper triangular matrix U, or (2) a sequence of elementary column additions to an upper triangular matrix U, or (3) a sequence of elementary column additions to a lower triangular matrix L, or (4) a sequence of elementary column additions to a lower triangular matrix L.

$A = E_T E_{T-1}....E_3 E_2 E_1 U$ where U is an upper triangular matrix

$A = E'_T E'_{T-1}...E'_3 E'_2 E'_1 L'$ where L' is a lower triangular matrix

$B = U'F'_1 F'_2 F'_3....F'_{T-1} F'_T$ where U' is an upper triangular matrix

$B = LF_1F_2F_3....F_{T-1}F_T$ where L is a lower triangular matrix

In general, upper triangular matrix is defined as matrix having zero element when row index is greater than column index, while lower triangular matrix is defined as matrix having zero element when row index is smaller than column index. The diagonal elements of triangular matrix are not necessarily one. (However, for LU decomposition, all the diagonal elements of L matrix are one.)

$U = (U_{i,j})$ such that $U_{i,j} = 0 \quad \forall i > j$ and $i,j \in [1,N]$ and $\exists n \in [1,N]$ such that $U_{n,n} \neq 1$

$L = (L_{i,j})$ such that $L_{i,j} = 0 \quad \forall i < j$ and $i,j \in [1,N]$ and $\exists n \in [1,N]$ such that $L_{n,n} \neq 1$

All these four decompositions are known as Gaussian elimination (yet the first one is the most common). Lets derive the elementary matrices for case 1 (case 2 – 4 are left as exercises, which are very similar to case 1). In the procedures below, every elementary addition operation makes use of the most updated result after all the previous elementary addition operations (i.e. $a_{i,j}$ in step t refers to the element of the i^{th} row and the j^{th} column of $E_{t-1}E_{t-2}...E_2E_1A$).

$E_1 = E_{add}(2,1,-a_{2,1}/a_{1,1})$ add multiple of row 1 to row 2, setting $a_{2,1}$ zero, valid only for $a_{1,1} \neq 0$

$E_2 = E_{add}(3,1,-a_{3,1}/a_{1,1})$ add multiple of row 1 to row 3, setting $a_{3,1}$ zero, valid only for $a_{1,1} \neq 0$

$E_3 = E_{add}(3,2,-a_{3,2}/a_{2,2})$ add multiple of row 2 to row 3, setting $a_{3,2}$ zero, valid only for $a_{2,2} \neq 0$

$E_4 = E_{add}(4,1,-a_{4,1}/a_{1,1})$ add multiple of row 1 to row 4, setting $a_{4,1}$ zero, valid only for $a_{1,1} \neq 0$

$E_5 = E_{add}(4,2,-a_{4,2}/a_{2,2})$ add multiple of row 2 to row 4, setting $a_{4,2}$ zero, valid only for $a_{2,2} \neq 0$

$E_6 = E_{add}(4,3,-a_{4,3}/a_{3,3})$ add multiple of row 3 to row 4, setting $a_{4,2}$ zero, valid only for $a_{3,3} \neq 0$ and so on.

Lets find T in terms of N. $T = 1+2+3+...+(N-1) = N(N-1)/2$

Remark 5

- Elementary row addition $E_{add}(k,l,s)$ is an upper triangular matrix if k < l, a lower triangular matrix otherwise.
- Elementary column addition $F_{add}(k,l,s)$ is a lower triangular matrix if k < l, a upper triangular matrix otherwise.

- Cascade of elementary row addition :

  $E_T(k_T,l_T,s_T)....E_3(k_3,l_3,s_3)E_2(k_2,l_2,s_2)E_1(k_1,l_1,s_1)$ is an upper triangular matrix if $k_t < l_t \quad \forall t \in [1,T]$

  $E_T(k_T,l_T,s_T)....E_3(k_3,l_3,s_3)E_2(k_2,l_2,s_2)E_1(k_1,l_1,s_1)$ is a lower triangular matrix if $k_t > l_t \quad \forall t \in [1,T]$

- Cascade of elementary column addition :

  $F_T(k_T,l_T,s_T)....F_3(k_3,l_3,s_3)F_2(k_2,l_2,s_2)F_1(k_1,l_1,s_1)$ is a lower triangular matrix if $k_t < l_t \quad \forall t \in [1,T]$

  $F_T(k_T,l_T,s_T)....F_3(k_3,l_3,s_3)F_2(k_2,l_2,s_2)F_1(k_1,l_1,s_1)$ is an upper triangular matrix if $k_t > l_t \quad \forall t \in [1,T]$

Gram Schmidt process

Gram Schmidt process is an algorithm for orthonormalizing a set of vector. Suppose we are given a set of N vectors in M dimensional space, i.e. A = {a_1, a_2, …, a_N} where $a_n \in \Re^M \ \forall n \in [1,N]$, Gram Schmidt process can help to find a set of R vectors in M dimensional space, i.e. Q = {q_1, q_2, …, q_K} where $q_k \in \Re^M$, $\forall k \in [1,K]$, and $<q_{k1},q_{k2}> = mag(q_{k1})mag(q_{k2})\delta_{k1,k2}$, $\forall k_1,k_2 \in [1,K]$ (i.e. Q is an **orthogonal** vector set), such that the span of vector set Q is the same as the span of vector set A, and K is known as the rank of vector set A. We can simplify vector set Q as unit vector set E = {e_1, e_2, …, e_R} where $e_k = q_k / |q_k|$, $\forall k \in [1,K]$ (i.e. E is an **orthonormal** vector set). Before we proceed, lets clearly define span and null space of vector set A. In short, span of a vector set is the space spanned by linear combination of the vectors in the set, while null space is the space that is **complement** of the span. The dimension of span is called rank.

$$span = \{v : v = \sum_{n=1}^{N} w_n a_n, w_n \in \Re^1, \forall n \in [1,N]\}$$
$$null = \{z :< z, v >= 0, \forall v \in span\}$$

Lets define dot product (inner product) between two vectors and projection operator (projection of vector v on vector u).

$< v,u >$ = $\vec{v} \cdot \vec{u}$ for vector notation

$< v,u >$ = $v^T u$ for column matrix

$< v,u >$ = $v u^T$ for row matrix

$proj_u(v)$ = $\frac{< v,u >}{< u,u >} u$ for any vector u

$proj_u(v)$ = $< v,u > u$ for unit vector u

Gram Schmidt process is simple, for all $n \in [1,N]$, we simply pick components of vector a_n that are orthogonal to all previously found orthogonal vectors q_1, q_2, q_3, …, q_{n-1} as q_n. Here we assume that (1) the number of vectors in vector set A is smaller than (or equals to) the dimension of vector (i.e. $N \le M$) and (2) the number of vectors in vector set A equals to its rank (i.e. N = K), therefore q_n always has a non zero magnitude.

$$q_1 = a_1 \quad \text{and} \quad e_1 = q_1 / \|q_1\|$$
$$q_2 = a_2 - proj_{q_1}(a_2) \quad \text{and} \quad e_2 = q_2 / \|q_2\|$$
$$q_3 = a_3 - proj_{q_1}(a_3) - proj_{q_2}(a_3) \quad \text{and} \quad e_3 = q_3 / \|q_3\|$$
$$\dots$$
$$q_N = a_N - proj_{q_1}(a_N) - proj_{q_2}(a_N) - proj_{q_3}(a_N) - \dots - proj_{q_{N-1}}(a_N) \quad \text{and} \quad e_N = q_N / \|q_N\|$$

We can also express vector set A in terms of vector set Q.

$$a_1 = q_1 = proj_{q_1}(a_1) = proj_{e_1}(a_1)$$
$$a_2 = q_2 + proj_{q_1}(a_2) = proj_{q_1}(a_2) + proj_{q_2}(a_2) = proj_{e_1}(a_2) + proj_{e_2}(a_2)$$
$$a_3 = q_3 + proj_{q_1}(a_3) + proj_{q_2}(a_3) = proj_{q_1}(a_3) + proj_{q_2}(a_3) + proj_{q_3}(a_3) = proj_{e_1}(a_3) + proj_{e_2}(a_3) + proj_{e_3}(a_3)$$
$$\dots$$
$$a_N = a_N + proj_{q_1}(a_N) + proj_{q_2}(a_N) + proj_{q_3}(a_N) + \dots + proj_{q_{N-1}}(a_N) = \sum_{n=1}^{N} proj_{e_n}(a_N)$$

In general, (1) the number of vectors in vector set A may be larger than the dimension of vector (i.e. N > M), and (2) the number of vectors in vector set A may be larger than its rank (i.e. N > K), the algorithm should be enhanced to handle vanished intermediate q_n, as shown the following psuedo code.

```
unsigned short gram_schmidt_process(const std::vector<data>& A, std::vector<data>& Q) // Rank is returned.
{
    Q.push_back(A[0]);
    for(unsigned short n=1; n!=N; ++n)
    {
        temp = find_orthogonal(A[n],Q);
        if (temp!=zero_matrix) Q.push_back(temp);
    }
    return Q.size();
}

data find_orthogonal(const data& an, const std::vector<data>& Q)
{
    data result = an;
    for(std::vector<data>::const_iterator q=Q.begin(); q!=Q.end(); ++q)
    {
        result -= projection(an, *q);
    }
    return result;
}
```

Gram Schmidt process cannot be used to find the null space. Yet, it can be used to (1) derive QR decomposition, (2) derive conjugate gradient and (3) derive volume of parallelepiped.