# Computational Gram-Schmidt Orthogonalization Process

Jason Zhang

July 28, 2022

## 1 Gram-Schmidt Process

In linear algebra, the Gram-Schmidt process is an algorithm that orthogonalizes a set of given vectors in the Euclidean space $\mathbb{R}^n$. More specifically, given $\{\vec{\mathbf{x}}_1, \vec{\mathbf{x}}_2, \ldots, \vec{\mathbf{x}}_m\}$ is a set of subspace $U$ of $\mathbb{R}^n$, the algorithm constructs an orthonormal set $\{\vec{\mathbf{f}}_1, \vec{\mathbf{f}}_2, \ldots, \vec{\mathbf{f}}_m\}$ that spans the same subspace $U$ of $\mathbb{R}^n$.

### 1.1 Orthogonality

Two vectors, $\vec{\mathbf{x}}$ and $\vec{\mathbf{y}}$ are orthogonal if and only if $\vec{\mathbf{x}} \cdot \vec{\mathbf{y}} = 0$.

**Definition: Orthogonal and Orthonormal Sets**

*A set* $\{\vec{\mathbf{f}}_1, \vec{\mathbf{f}}_2, \ldots, \vec{\mathbf{f}}_m\}$ *of vectors in* $\mathbb{R}^n$ *is an* ***orthogonal set*** *if*

$$\vec{\mathbf{f}}_i \cdot \vec{\mathbf{f}}_j = 0 \text{ for all } i \neq j \text{ and } \vec{\mathbf{f}}_i \neq 0 \text{ for all } i$$

*A set* $\{\vec{\mathbf{f}}_1, \vec{\mathbf{f}}_2, \ldots, \vec{\mathbf{f}}_m\}$ *of vectors in* $\mathbb{R}^n$ *is an* ***orthonormal set*** *if the set is orthogonal and each* $\vec{\mathbf{f}}_i$ *is a unit vector.*

To convert an orthogonal into an orthonormal set, each vector in the orthogonal set is converted into its unit vector, namely normalized.

**Definition: Normalizing an Orthogonal Set**

*Given the orthogonal set* $\{\vec{\mathbf{f}}_1, \vec{\mathbf{f}}_2, \ldots, \vec{\mathbf{f}}_m\}$*, then the orthonormal set is* $\{\frac{1}{\|\vec{\mathbf{f}}_1\|}\vec{\mathbf{f}}_1, \frac{1}{\|\vec{\mathbf{f}}_2\|}\vec{\mathbf{f}}_2, \ldots, \frac{1}{\|\vec{\mathbf{f}}_m\|}\vec{\mathbf{f}}_m\}$*. The orthonormal set is the result of* ***normalizing*** *the orthogonal set.*

The following lemma is the basis to which the Gram-Schmidt process is constructed upon.

**Lemma: Orthogonal Lemma**

*Given the orthogonal set* $\{\vec{\mathbf{f}}_1, \vec{\mathbf{f}}_2, \ldots, \vec{\mathbf{f}}_m\}$ *in* $\mathbb{R}^n$ *and given a random vector* $\vec{\mathbf{x}}$ *in* $\mathbb{R}^n$*. An additional orthogonal vector in* $\mathbb{R}^n$, $\vec{\mathbf{f}}_{m+1}$ *can be found:*

$$\vec{\mathbf{f}}_{m+1} = \vec{\mathbf{x}} - \frac{\vec{\mathbf{x}} \cdot \vec{\mathbf{f}}_1}{\|\vec{\mathbf{f}}_1\|^2}\vec{\mathbf{f}}_1 - \frac{\vec{\mathbf{x}} \cdot \vec{\mathbf{f}}_2}{\|\vec{\mathbf{f}}_2\|^2}\vec{\mathbf{f}}_2 - \cdots - \frac{\vec{\mathbf{x}} \cdot \vec{\mathbf{f}}_k}{\|\vec{\mathbf{f}}_m\|^2}\vec{\mathbf{f}}_m$$

## 1.2 Gram-Schmidt Process

The above lemma generalizes the orthogonalization; the conversion of a single random vector into an orthogonal vector. Now, the Gram-Schmidt process orthogonalizes a set of given random vectors into its corresponding set of orthogonal vectors.

**Theorem: Gram-Schmidt Orthogonalization Algorithm**

Given the set $\{\vec{\mathbf{x}}_1, \vec{\mathbf{x}}_2, \ldots, \vec{\mathbf{x}}_m\}$ in subspace $U$ of $\mathbb{R}^n$, the corresponding orthogonal set $\{\vec{\mathbf{f}}_1, \vec{\mathbf{f}}_2, \ldots, \vec{\mathbf{f}}_m\}$ can be successively constructed:

$$
\begin{aligned}
\vec{\mathbf{f}}_1 &= \vec{\mathbf{x}}_1 \\
\vec{\mathbf{f}}_2 &= \vec{\mathbf{x}}_2 - \frac{\vec{\mathbf{x}}_2 \cdot \vec{\mathbf{f}}_1}{\|\vec{\mathbf{f}}_1\|^2}\vec{\mathbf{f}}_1 \\
\vec{\mathbf{f}}_3 &= \vec{\mathbf{x}}_3 - \frac{\vec{\mathbf{x}}_3 \cdot \vec{\mathbf{f}}_1}{\|\vec{\mathbf{f}}_1\|^2}\vec{\mathbf{f}}_1 - \frac{\vec{\mathbf{x}}_3 \cdot \vec{\mathbf{f}}_2}{\|\vec{\mathbf{f}}_2\|^2}\vec{\mathbf{f}}_2 \\
&\vdots \\
\vec{\mathbf{f}}_k &= \vec{\mathbf{x}}_k - \frac{\vec{\mathbf{x}}_k \cdot \vec{\mathbf{f}}_1}{\|\vec{\mathbf{f}}_1\|^2}\vec{\mathbf{f}}_1 - \frac{\vec{\mathbf{x}}_k \cdot \vec{\mathbf{f}}_2}{\|\vec{\mathbf{f}}_2\|^2}\vec{\mathbf{f}}_2 - \cdots - \frac{\vec{\mathbf{x}}_k \cdot \vec{\mathbf{f}}_{k-1}}{\|\vec{\mathbf{f}}_{k-1}\|^2}\vec{\mathbf{f}}_{k-1}
\end{aligned}
$$

This successive construction is generalized as follows:

$$\vec{\mathbf{f}}_k = \vec{\mathbf{x}}_k - \sum_{i=1}^{k-1} \frac{\vec{\mathbf{x}}_k \cdot \vec{\mathbf{f}}_i}{\|\vec{\mathbf{f}}_i\|^2}\vec{\mathbf{f}}_i$$

The code functionality will resemble the above algorithm formula.

## 1.3 Example

Below provides an example that demonstrates the implementation of the Gram-Schmidt orthogonalization algorithm.

**Example: Laplace Expansion**

*Given the row space of* $A = \begin{bmatrix} 1 & 3 & -6 \\ 3 & 4 & 1 \\ 9 & 5 & 2 \end{bmatrix}$ *find an orthonormal basis.*

First, let $\vec{\mathbf{x}}_1$, $\vec{\mathbf{x}}_2$, $\vec{\mathbf{x}}_3$ denote the rows of A. Second, let $\vec{\mathbf{f}}_1$, $\vec{\mathbf{f}}_2$, $\vec{\mathbf{f}}_3$ denote the corresponding orthogonalized rows of A. Applying the Gram-Schmidt orthogonalization algorithm successively for each row:

$$\begin{aligned}
\vec{\mathbf{f}}_1 &= \vec{\mathbf{x}}_1 = (1, 3, -6) \\
\vec{\mathbf{f}}_2 &= \vec{\mathbf{x}}_2 - \frac{\vec{\mathbf{x}}_2 \cdot \vec{\mathbf{f}}_1}{||\vec{\mathbf{f}}_1||^2}\vec{\mathbf{f}}_1 = (3, 4, 1) - \frac{9}{\sqrt{46}}(1, 3, -6) = (2.804, 3.413, 2.174) \\
\vec{\mathbf{f}}_3 &= \vec{\mathbf{x}}_3 - \frac{\vec{\mathbf{x}}_3 \cdot \vec{\mathbf{f}}_1}{||\vec{\mathbf{f}}_1||^2}\vec{\mathbf{f}}_1 - \frac{\vec{\mathbf{x}}_3 \cdot \vec{\mathbf{f}}_2}{||\vec{\mathbf{f}}_2||^2}\vec{\mathbf{f}}_2 = (9, 5, 2) - \frac{12}{\sqrt{46}}(1, 3, -6) - \frac{46.649}{\sqrt{26}}(2.804, 3.413, 2.174) \\
&= (3.342, -2.352, -0.619)
\end{aligned}$$

Now, the orthogonalized row space of A is:

$$\begin{bmatrix} 1 & 3 & -6 \\ 2.804 & 3.413 & 2.174 \\ 3.342 & -2.352 & -0.619 \end{bmatrix}$$

## 2 Code Functionality

The program written in the C computer programming language functions the Gram-Schmidt orthogonalization algorithm. The complete code is linked here. The program prompts the vector space dimension, the number of vectors and vector entries. Dynamic memory allocation is applied to store the vectors into a matrix, where each row is a vector and the number of columns is the vector dimension. The concerned block of code (written in C) is provided below:

```c
for (int k = 1; k <= numberOfVectors; k++){
    for (int col = 0; col < dimension; col++){
        orthogonalVM[k-1][col] = originalVM[k-1][col];
    }

    if (k > 1){
       for (int iteration = 0; iteration < dimension; iteration++){
            tempVectorSum[iteration] = 0;
       }
       for (int i = 1; i < k; i++){
            dotProduct = dotProd(originalVM, orthogonalVM, dimension,
                  k-1, i-1);

            magnitudeSquared = fabs(vectorMagnitude(orthogonalVM,
                  dimension, i-1) * vectorMagnitude(orthogonalVM,
                  dimension, i-1));

            division = dotProduct / magnitudeSquared;

            vectorScalarMultiplication(orthogonalVM, dimension,
                  i-1, tempScalarMultiplyVector, division);
```

```
18                  vectorAddition(tempVectorSum, tempScalarMultiplyVector,
19                        dimension, tempVectorSum);
20              }
21              vectorSubtraction(originalVM, tempVectorSum, dimension, k-1,
22                    orthogonalVM, k-1);
23          }
24      }
```

`originalVM` is a dynamically allocated 2D matrix where the number of rows are the different vectors and the number of columns are the vector space. The vectors stored inside `originalVM` are the original vectors prompted by the user.

`orthogonalVM` is a dynamically allocated 2D matrix with the same dimensions as `originalVM`. `orthogonalVM` is initially empty such that the orthogonalized vectors are stored into `orthogonalVM`.

## 2.1 Copying First Term

From the Gram-Schmidt Orthogonalization Theorem, the formula was provided:

$$\vec{\mathbf{f}}_k = \vec{\mathbf{x}}_k - \sum_{i=1}^{k-1} \frac{\vec{\mathbf{x}}_k \cdot \vec{\mathbf{f}}_i}{\|\vec{\mathbf{f}}_i\|^2} \vec{\mathbf{f}}_i$$

It becomes evident that the first term of the $k^{th}$ orthogonal vector, $\vec{\mathbf{f}}_k$ is the $k^{th}$ original vector, $\vec{\mathbf{x}}_k$. As such, lines 1-4 will first set $\vec{\mathbf{f}}_k = \vec{\mathbf{x}}_k$.

```
1       for (int k = 1; k <= numberOfVectors; k++){
2           for (int col = 0; col < dimension; col++){
3               orthogonalVM[k-1][col] = originalVM[k-1][col];
4           }
```

The first for loop (line 1) with index `k`, iterates through each original vector, namely each row of the `originalVM`. The second for loop (line 2) with index `col`, iterate the entries of the $k^{th}$ original vector in `originalVM`.

Recall that the first term of each orthogonal vector is equal to its original vector. When iterating through the $k^{th}$ vector of `originalVM`, the entries are copied into the same location of `orthogonalVM`, conducted in line 3.

## 2.2 Scalar Computation

The second part of the code calculates the scalar (line 10-15; blue section), the scalar multiplication (line 16-17; red section), and the summation (line 18-22; purple section).

$$\vec{\mathbf{f}}_k = \vec{\mathbf{x}}_k - \sum_{i=1}^{k-1} \frac{\vec{\mathbf{x}}_k \cdot \vec{\mathbf{f}}_i}{\|\vec{\mathbf{f}}_i\|^2} \vec{\mathbf{f}}_i$$

```
5           if (k > 1){
6              for (int iteration = 0; iteration < dimension; iteration++){
7                   tempVectorSum[iteration] = 0;
```

```
8           }
9           for (int i = 1; i < k; i++){
10              dotProduct = dotProd(originalVM, orthogonalVM, dimension,
11                    k-1, i-1);

12              magnitudeSquared = fabs(vectorMagnitude(orthogonalVM,
13                    dimension, i-1) * vectorMagnitude(orthogonalVM,
14                    dimension, i-1));

15              division = dotProduct / magnitudeSquared;

16              vectorScalarMultiplication(orthogonalVM, dimension,
17                    i-1, tempScalarMultiplyVector, division);

18              vectorAddition(tempVectorSum, tempScalarMultiplyVector,
19                    dimension, tempVectorSum);
20           }
21           vectorSubtraction(originalVM, tempVectorSum, dimension, k-1,
22                 orthogonalVM, k-1);
23        }
24     }
```

To provide context of utilized functions:

`dotProd` returns the dot product of two input vectors
`magnitudeSquared` returns the magnitude and squares the result of an input vector
`vectorScalarMultiplication` returns a scalar multiplied vector
`vectorAddition` returns the sum of two input vectors
`vectorSubtraction` returns the difference of two input vectors

Scalar Calculation (Line 10-15):

Line 10-11 computes the dot product; $\vec{\mathbf{x}}_k \cdot \vec{\mathbf{f}}_i$
Line 12-14 computes the magnitude and squares the result; $\|\vec{\mathbf{f}}_i\|^2$
Line 15 divides the dot product and magnitude squared values; $\frac{\vec{\mathbf{x}}_k \cdot \vec{\mathbf{f}}_i}{\|\vec{\mathbf{f}}_i\|^2}$

Scalar Multiplication (Line 16-17):

Line 16-17 multiplies the value from line 15, $\frac{\vec{\mathbf{x}}_k \cdot \vec{\mathbf{f}}_i}{\|\vec{\mathbf{f}}_i\|^2}$, with the orthogonal vector $\vec{\mathbf{f}}_i$; $\frac{\vec{\mathbf{x}}_k \cdot \vec{\mathbf{f}}_i}{\|\vec{\mathbf{f}}_i\|^2}\vec{\mathbf{f}}_i$. The resulting vector is stored into a temporary location, named `tempScalarMultiplyVector`.

Summation (Line 18-22):

Line 7, `tempVectorSum` is a 1D array, a vector with entries initially initialized to 0.

Line 18-20 adds the two vectors, `tempScalarMultiplyVector` and `tempVectorSum`. The idea of line 18 is to sum `tempScalarMultiplyVector` values for all iterations starting from `i = 1` to `k - 1`. Hence, line 18 resembles the compound assignment operator, `tempVectorSum` += `tempScalarMultiplyVector`.

Line 21-22 subtracts the sum stored in `tempVectorSum`, $\sum_{i=1}^{k-1} \frac{\vec{\mathbf{x}}_k \cdot \vec{\mathbf{f}}_i}{\|\vec{\mathbf{f}}_i\|^2} \vec{\mathbf{f}}_i$ with the original $k^{th}$ vector, $\vec{\mathbf{x}}_k$; $\vec{\mathbf{x}}_k - \sum_{i=1}^{k-1} \frac{\vec{\mathbf{x}}_k \cdot \vec{\mathbf{f}}_i}{\|\vec{\mathbf{f}}_i\|^2} \vec{\mathbf{f}}_i$

# 3 Computational Expense

The time complexity to conduct the Gram-Schmidt orthogonalization algorithm is $\mathcal{O}(nm^2)$ where $n$ is the dimensionality and $m$ is the number of vectors. First, the computation expense of the dot product, magnitude squared, and the scalar multiplication are $\mathcal{O}(n)$. Second, the orthogonalization process of summation and subtraction scales the expense by $m$ times, where $m$ is the number of vectors, $\mathcal{O}(nm)$. Third, the orthogonalization process must be conducted up to $m$ times. Thus, $\mathcal{O}(nm^2)$ approximates the Gram-Schmidt orthogonalization algorithm.