A Survey on Gram-Schmidt Based Unsupervised Dimensionality Reduction: GFR and GFS

Cheng
Department of Electrical and System Engineering
Washington University in St. Louis
St. Louis, the United States
cheng.cynthia@wustl.edu

***Abstract*—This paper reflects on concepts learned in the ESE 2971 course, focusing on two advanced dimensionality reduction techniques: Gram-Schmidt Functional Reduction (GFR) and Gram-Schmidt Functional Selection (GFS), introduced by Yaghooti et al. These algorithms extend the Gram-Schmidt orthogonalization process to function spaces, offering the ability to capture and remove nonlinear dependencies in data. Unlike classical methods such as PCA, which are limited to linear variance structures, GFR and GFS enable a richer representation of unsupervised data. Through studying their theoretical framework and implementation, I gained insight into how orthogonal function projections and covariance analysis can lead to more effective feature extraction and selection in high-dimensional settings.**

I. Introduction

Talking about dimensionality reduction, I can't help but recall the third case study from our ESE105 class [5]. We were trying to create a classification algorithm to recognize handwritten digits using the MNIST dataset. One of the biggest challenges I encountered when implementing a K-Nearest Neighbors (KNN) algorithm was the high dimensionality of the data: each image is 28 by 28 pixels, leading to 784 features per image. Training the KNN model directly on this high-dimensional dataset would be computationally expensive and inefficient. With a basic understanding of dimensionality reduction, I applied k-means clustering to extract representative data points (centroids) from the training dataset for each digit. This effectively reduced the size of the dataset and eliminated redundancy, allowing my KNN classifier to operate more efficiently while preserving the essential structure of the data.

This experience introduced me to the practical significance of dimensionality reduction, which broadly refers to methods that represent data using fewer features while preserving important structures. In unsupervised representation learning, reducing dimensions helps reveal hidden patterns, simplifies downstream processing, and filters out redundant information [1]. Techniques like PCA project high-dimensional data into a lower-dimensional subspace defined by principal directions that capture the most variance. However, such linear methods can fall short when the data exhibits complex nonlinear dependencies.

In this report, I focus on two advanced dimensionality reduction methods: Gram-Schmidt Functional Reduction (GFR) and Gram-Schmidt Functional Selection (GFS). Both methods generalize the classical Gram-Schmidt orthogonalization process into function space, allowing them to identify and remove nonlinear redundancy in data. These methods are mathematically grounded yet practically powerful, and learning them made me realize how feature extraction and selection can go far beyond linear assumptions. This paper summarizes my learning process around these two algorithms and reflects on their significance in modern unsupervised learning. Especially, I will review our third case study from ESE 105 and reflect on the potential improvements that GFR and GFS may bring to that study.

II. Methods

A. Background and Foundation

1) Functions as Vectors in a Vector Space: A key mathematical insight behind Gram-Schmidt-based dimensionality reduction is that functions can be treated as vectors in a vector space. This might sound abstract at first, but it becomes intuitive when we recall that functions can be added together and multiplied by scalars—just like vectors. In fact, functions satisfy all axioms of a vector space: closure, associativity, commutativity of addition, existence of zero vector and additive inverse, and compatibility with scalar multiplication as seen in the examples from table I.

TABLE I: Functions vs. Vectors: Axioms of Vector Space

Axiom	**Vector Example**	**Function Example**
Addition	$[1,2]+[3,4]=[4,6]$	$f(x)+g(x)=x^2+\sin x$
Scalar Multiplication	$2\cdot[1,2]=[2,4]$	$3f(x)=3x^2$
Zero Element	$[0,0]$	$f(x)=0$
Additive Inverse	$-[1,2]=[-1,-2]$	$-f(x)$ such that $f(x)+(-f(x))=0$

This is important because it allows us to apply tools like the Gram-Schmidt process and inner product spaces to functions. Just like in Fourier transform, where a signal is expressed as a sum of sine and cosine waves, or Taylor series where a smooth function is approximated by polynomials, we can

view complex data structures as a linear combination of basis functions. If we identify a useful function family that spans our data space, we can describe the data efficiently using fewer functions—thereby reducing dimensionality while preserving meaningful structure.

2) Gram-Schmidt Orthogonalization over Function Spaces: The Gram-Schmidt process is a classic method for turning a set of linearly independent vectors into an orthonormal basis. When extended to functions, this means converting a set of functions into an orthonormal system under an inner product defined by:

$$\langle f, g \rangle = \mathbb{E}[f(X)g(X)]$$

This inner product is evaluated over the data distribution (which we do not need to know), and in practice, is approximated empirically using the dataset. At each iteration j, we generate a new function by removing its projections onto the already orthogonalized ones, and normalize it. Formally, given functions $f_1, \ldots, f_{j-1}$, we generate:

$$\tilde{f}_j = f_j - \sum_{i=1}^{j-1} \langle f_j, f_i \rangle f_i \quad , \quad \hat{f}_j = \frac{\tilde{f}_j}{\|\tilde{f}_j\|}$$

This process ensures that the resulting functions are orthonormal, meaning they capture distinct and independent structures in the data.

3) Feature Extraction vs. Feature Selection: Dimensionality reduction methods generally fall into two categories: feature extraction and feature selection.

- **Feature Extraction** involves creating new features as combinations of the original ones [2]. PCA is a classic example, where principal components are linear combinations of input features. In fact, what I implemented in my third case study is a practical example of feature extraction. By using k-means clustering, I replaced the full training dataset with its cluster centroids. These centroids represent synthesized data points, not a subset of original features, thus compressing the dataset while preserving its structure.
  - GFR is a feature extraction technique—it constructs new features that retain high variance but are derived from nonlinear transformations.
- **Feature Selection** involves choosing a subset of the original features based on specific criteria such as variance, redundancy, or mutual information [2]. This approach can be particularly useful in image recognition tasks like our third case study, where many corner pixels contribute little to the classification of handwritten digits. Eliminating those less informative pixels and focusing only on the most relevant ones could make the model more efficient. While we attempted to approximate this idea through a manually constructed decision boundary—by stacking together a limited number of example images—our method lacked the precision and theoretical grounding that a technique like GFS offers.
  - GFS is a selection method—it identifies original features that are most informative under a nonlinear projection and removes redundant ones.

B. Gram-Schmidt Functional Reduction (GFR)

GFR is a feature extraction technique that extends PCA by incorporating nonlinear relationships in the data using function spaces. It relies on the Gram-Schmidt orthogonalization process applied not to vectors, but to functions—specifically, functions constructed from linear projections of the data.

Before applying GFR, we start by standardizing the data matrix X, ensuring it has zero mean and unit variance. (Although I noticed that in some examples only centering is done, while in others both centering and normalization are applied.)

The core idea is similar to PCA: we extract directions that maximize variance. In PCA, this is done by computing the covariance matrix $\Sigma = \mathbb{E}[XX^\top]$, performing eigendecomposition, and taking the top eigenvectors. In GFR, however, we iteratively construct a function family $\mathcal{F}$, orthogonalize it using Gram-Schmidt, and update the data matrix after each step to remove captured structure.

Example Process: In one example, we use a simple function family consisting of increasing powers:

$$f_1(x) = x, \quad f_2(x) = x^2$$

First, we compute the projection of the data onto a direction ν, the leading eigenvector of the covariance matrix (alg. 1, lines 12–13), resulting in $z_1 = \langle \nu, X \rangle$ (alg. 1, line 14). We define the first function $f_1 = z_1$, and normalize it to obtain $\hat{f}_1 = \frac{z_1}{\|z_1\|}$ (alg. 1, line 15). This forms the first member of our orthonormal function family $\mathcal{F}$ (alg. 1, line 16). Next, we generate a new function $f_2 = z_1^2$, and center it by subtracting the mean to emphasize its dynamic variation rather than absolute magnitude (alg. 1, line 18). We perform Gram-Schmidt orthogonalization by projecting f_2 onto $\hat{f}_1$ and subtracting that projection (alg. 1, line 19), then normalize the result to get $\hat{f}_2$ (alg. 1, line 20). The function family is updated to include this second function (alg. 1, line 21). Finally, we update the data matrix by projecting X onto each function $f \in \mathcal{F}$ and subtracting these projections (alg. 1, line 23–25):

$$d_j = X - \sum_{f \in \mathcal{F}} \mathbb{E}[Xf] \cdot f$$

This residual matrix d_j (alg. 1, line: 26) retains the unexplained structure and may be reused in subsequent iterations to extract further nonlinear features.

The iterative process terminates when the leading eigenvalue falls below a predefined threshold, indicating that little additional variance remains to be captured by new features.

General Framework: Formally, at each iteration we compute:

$$f_j = \langle \nu_j, X \rangle, \quad \text{then normalize:} \quad \hat{f}_j = \frac{f_j}{\|f_j\|}$$

Algorithm 1 Simplified Implementation of GFR (Gram-Schmidt Functional Reduction)

1: **Input:** Synthetic data matrix $X_{\text{original}} \in \mathbb{R}^{2\times 5}$ where $x_2 = x_1^2$
2: **Output:** Orthonormal function family $\mathcal{F}$, residual data matrix d_j, diagnostic plots
3: Define $x_1 \leftarrow [1, 2, 3, 4, 5]$
4: Define $x_2 \leftarrow x_1^2$
5: Stack to form data matrix: $X_{\text{original}} \leftarrow [x_1; x_2]$
6: **Center the data:**
7: Compute column-wise mean: $X_{\text{mean}} \leftarrow \text{mean}(X_{\text{original}})$
8: $X \leftarrow X_{\text{original}} - X_{\text{mean}}$
9: **Compute covariance and eigenvectors:**
10: $\Sigma \leftarrow \text{cov}(X)$
11: $(\lambda, V) \leftarrow \text{eig}(\Sigma)$
12: Select principal direction v_1 (highest eigenvalue)
13: **Project data to function space:**
14: $z_1 \leftarrow v_1 \cdot X$
15: Normalize: $\hat{f}_1 \leftarrow z_1/\|z_1\|$
16: Initialize function family $\mathcal{F} \leftarrow \{\hat{f}_1\}$
17: **Construct second function:**
18: $z_1^2 \leftarrow z_1^2 - \text{mean}(z_1^2)$
19: Orthogonalize: $\tilde{f}_2 \leftarrow z_1^2 - \langle z_1^2, \hat{f}_1\rangle \hat{f}_1$
20: Normalize: $\hat{f}_2 \leftarrow \tilde{f}_2/\|\tilde{f}_2\|$
21: Update $\mathcal{F} \leftarrow \mathcal{F} \cup \{\hat{f}_2\}$
22: **Compute residuals after projection:**
23: $d_j \leftarrow X$
24: **for** each f in $\mathcal{F}$ **do**
25: $\quad d_j \leftarrow d_j - (\mathbb{E}[X \cdot f]) \cdot f$
26: **end for**
27: **Plot:**
28: Scatter original data x_1 vs. x_2
29: Plot PCA direction vector v_1
30: Scatter z_1 vs. z_1^2
31: Scatter residuals $d_j[0]$ vs. $d_j[1]$

We iteratively build the function family $\mathcal{F} = \{\hat{f}_1, \hat{f}_2, \ldots\}$, performing orthogonalization and normalization as we go. The key point is that instead of just projecting onto eigenvectors like in PCA, GFR constructs an orthogonal function basis that captures nonlinear structure in the data.

This process enables us to uncover richer directions of variance and perform dimensionality reduction more effectively when data relationships are nonlinear. One of the things I found especially elegant is that GFR still retains a linear structure in the feature space, but generalizes it by projecting onto functions—not raw vectors.

C. Gram-Schmidt Functional Selection (GFS)

As summarized in table II, compared to GFR, which creates new features based on data projections, GFS instead operates in a more conservative and interpretable way—it works directly with raw input features and selects those that are most informative. This method is designed for feature selection, not feature extraction.

The key idea behind GFS is to identify which features in the dataset are actually contributing unique information. Unlike GFR:

- GFS does not create any new transformed features (z_j, z_j^2, etc.).
- It operates entirely in the raw feature space.
- Redundancy is iteratively removed from the dataset through functional orthogonalization (alg. 2, line 14).

At each step, we select the feature that contributes the most remaining variance (alg. 2, line 9) and then project it out (along with its nonlinear correlations) from the dataset (alg. 2, line 14). The formula to update the data matrix is:

$$d_j = d_{j-1} - \sum_{f\in\mathcal{F}} \mathbb{E}[d_{j-1} f] \cdot f$$

Similar to GFR, We repeat this until the maximum remaining variance (i.e., the largest diagonal value in the empirical covariance matrix) drops below a threshold ε^2 (alg. 2, line 6), meaning there's not much more new information to extract.

TABLE II: Comparison Between GFR and GFS

Aspect	GFR	GFS
Purpose	Feature extraction	Feature selection
Creates new features?	Yes (e.g., z_1, z_1^2)	No (uses raw features)
Projection space	Function of data projections	Original coordinates
Redundancy removal	From new features	From original features
Interpretability	Harder	Easier (original feature indices preserved)

Algorithm 2 Gram-Schmidt Functional Selection (GFS)

1: **Input:** Centered dataset $X \in \mathbb{R}^{d\times n}$, threshold $\varepsilon^2 > 0$
2: **Output:** Selected feature indices $S = \{s_1, \ldots, s_m\}$
3: Initialize $d_0 = X$, $\mathcal{F} = \emptyset$
4: **for** $j = 1$ to d **do**
5: $\quad$ Compute feature-wise variance: $\sigma_j = \mathbb{E}[d_{j-1} d_{j-1}]$
6: $\quad$ **if** $\|\sigma_j\|_\infty \le \varepsilon^2$ **then**
7: $\quad\quad$ **Break**
8: $\quad$ **end if**
9: $\quad$ Select index of max variance: $s_j = \arg\max_i \sigma_{j,i}$
10: $\quad$ Let $f_j = d_{j-1}[s_j]$
11: $\quad$ Orthogonalize f_j w.r.t. previous $\mathcal{F}$
12: $\quad$ Normalize: $\hat{f}_j = f_j/\|f_j\|$
13: $\quad$ Update $\mathcal{F} \leftarrow \mathcal{F} \cup \{\hat{f}_j\}$
14: $\quad$ Update: $d_j = d_{j-1} - \sum_{f\in\mathcal{F}} \mathbb{E}[d_{j-1} f] f$
15: **end for**

This process is similar to PCA in that we use eigen-analysis to identify the most informative direction (feature), but instead of projecting data into a new subspace, we just clean the current space. The functional projection step removes linear and nonlinear dependencies among features. As a result, the

remaining data matrix becomes increasingly orthogonalized with respect to selected features.

III. Results and Discussion

A. Comparison with Principal Component Analysis (PCA)

PCA is often the first tool people think of when it comes to dimensionality reduction [4]. At its core, PCA computes the covariance matrix of the dataset, then performs eigendecomposition to find the directions—called principal components—that capture the most linear variance in the data. By projecting data onto the top k principal components, we reduce dimensionality while retaining as much variance as possible.

But here's the catch: PCA is inherently a linear technique. It works great if the important patterns in the data happen to be aligned along straight lines or planes. However, real-world data (especially images, signals, or human-generated patterns) often exhibits nonlinear structure. That's where PCA starts falling short.

1) Where PCA Falls Short: Consider a simple case: data points lying on a parabola. PCA will find the line that minimizes orthogonal projection error—likely the axis of the parabola—but it won't see the curve. Any variance along that curvature will be interpreted as noise or compressed away. In image data like handwritten digits, subtle nonlinear traits—like how curved a "3" is versus how open a "5" is—can be critical for classification, but PCA may miss them.

2) How GFR and GFS Outperform PCA: Both GFR and GFS were developed to overcome PCA's blind spots. They generalize the idea of projection: instead of projecting onto plain vectors (like PCA), they project onto *functions* of the data. That's the real game-changer. These functions can include nonlinear transformations like squares, cross-terms, or other basis functions.

- **GFR** constructs a new orthonormal function basis and iteratively removes projections from the data. It captures nonlinear variance and allows for more expressive feature extraction. It works especially well when you suspect hidden, compound relationships among features.
- **GFS**, on the other hand, works directly with original features and selects the ones that explain the most nonlinear variance—removing redundant features that may not look correlated at first glance, but become redundant when considered under nonlinear mappings.

3) Results from the Original Paper: According to the results presented in the original paper [3], GFR consistently reduces residual variance more efficiently than PCA, often achieving the same level of variance retention with significantly fewer features. For example, in one synthetic test using LTF labels, GFR initially performs similarly to PCA when the number of extracted features is low. But as more features are included, GFR begins to clearly outperform PCA. When the number of extracted features reaches 30, GFR achieves a classification accuracy of 84.28%, compared to 80.95% for PCA. Even more impressively, GFR's advantage isn't limited to synthetic data—it also preserves classification accuracy better than PCA on real-world benchmarks like image recognition and genomics.

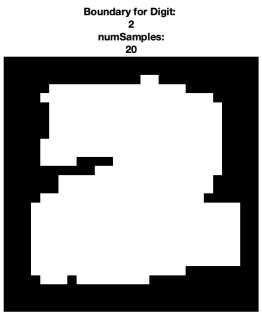

Fig. 1: A sample decision boundary for class 2 using 20 randomly selected digit 2 training images

B. How GFS Could Improve Our Decision Boudnary Method in the Third Case Study

In our third case study, we built a simple decision boundary method for classifying digits — the one where we stacked together a few training images for each digit and used a thresholded dot product to make predictions as seen in fig. 1 [5]. The logic was: if the input image looks like the average of a few samples from a digit class, it's probably that digit. It worked with in average 60% or 70% accuracy - which is not bad for how primitive it was. We fully acknowledge that method had some serious limitations.

First, the samples we chose were pretty arbitrary. We didn't consider which pixels were actually informative for distinguishing digits, and instead treated all pixels equally. Second, the way we built each class boundary—by just summing a few binary images—didn't filter out any redundant or noisy information. And finally, we had to manually tune thresholds based on some trial and error.

Here's where I think GFS could help. Instead of treating every pixel the same, GFS lets us identify which pixel positions actually matter—the ones that change the most and aren't just repeating patterns we've already accounted for. For example, if we're building a boundary for the digit "3", GFS might help us focus on the curved top and bottom regions, while ignoring the corners or background pixels that stay blank no matter the digit.

More concretely, this is how I imagine it could work:

- We start with the full set of flattened training images.
- Apply GFS to select the features (pixels) that carry the most information after removing redundancy.
- Build our decision boundary w_k using only those selected features—not the full image.

The score function could still look like this:

$$\text{score}(x) = w_k^\top x$$

But this time, w_k would be grounded in real statistical structure instead of just being a rough visual average of a few images. That would make the classifier:

- **More stable**, because we're not randomly picking samples anymore.
- **More interpretable**, since we know why each pixel is part of the decision.
- **Probably more accurate**, especially at lower threshold values where noise tends to dominate.

That said, there's still one tricky part: GFS needs a function family—like polynomials—to define redundancy, and it's not obvious which one works best for MNIST. So we'd still have to test that. But compared to our stacking method, this feels way more grounded and scalable. If I were to redo that boundary method today, I'd definitely consider integrating GFS into the pipeline.

C. How GFR Could Improve Our KNN Method in the Third Case Study

Also, I realized that while k-means clustering helped us reduce the size of the training set for KNN, it still relied on raw pixel values—which carry a lot of redundancy and noise [5]. Our clusters were built directly on the original 784-dimensional space, and even though we preprocessed the tilt of digits, there's no guarantee that the most meaningful variation in handwriting was actually captured by the Euclidean distance in that space.

This is where GFR could come in as a game-changer. Instead of clustering on raw pixel data, we could first apply GFR to project the dataset into a function space that captures nonlinear relationships between pixels. In this transformed space, both the cluster centroids and the test images would exist in a cleaner, orthogonal basis—one constructed from directions that maximize variance. The orthogonalization step in GFR ensures that each new component adds genuinely new information, helping us preserve essential differences while eliminating redundancy. More importantly, if we can define a function family that fits the dataset well, GFR can help us extract nuanced and meaningful features—like curve smoothness, stroke direction, or loop closures—that aren't easily captured by raw pixel values. These high-level traits often play the vital role in digit classification. Then we can run k-means in the new space to extract centroids. In this way, the centroids wouldn't just represent raw pixel averages but nonlinear feature patterns, potentially boosting both speed and accuracy—especially for those digits that can easily confuse the classifier in a purely pixel-wise context, like "3" or "5."

We could also use GFR directly as a preprocessing step for KNN without using K-means at all. If we take the raw image vectors, project them through a set of orthonormal nonlinear functions built via Gram-Schmidt (e.g., using z, z^2, etc.), and compute residuals as in the GFR process, we get new transformed feature vectors. Feeding these into KNN might make the Euclidean distance actually reflect perceptual similarity, rather than just pixel-wise alignment.

What I also like about GFR here is its flexibility—we could keep tuning the function family to adapt to the actual shape variation in the digits. Instead of assuming that distance in raw image space is meaningful, GFR helps us to see a function space where more nuanced relation lives. As we mentioned in potential application of GFS, this flexibility may also comes with a downside - which is the cost for us to search for a proper family of function.

IV. Conclusion

Through this paper, I've explored GFR and GFS not just as abstract mathematical tools, but as practical methods for dimensionality reduction that address some of the real limitations I encountered in earlier coursework. Both methods build on the familiar idea of projection, but extend it into function space—opening up the possibility of capturing nonlinear patterns that PCA can't reach.

Learning these methods helped me rethink how we treat features in high-dimensional datasets. GFR showed me how we can extract richer representations from data by iteratively removing structured variance using nonlinear transformations. GFS, on the other hand, offered a smarter way to filter out redundant features based on functional importance.

If I had known about these methods earlier, I would have seriously considered using them in my third case study. Not only could GFR and GFS have improved classification accuracy, but they also would have grounded our modeling process in something more principled than visual heuristics.

In short, GFR and GFS gave me new ways to think about data, structure, and learning—and that's exactly what I hoped to gain from this research experience.

References

[1] P. Schneider and F. Xhafa, "Machine learning: ML for eHealth systems," in *Anomaly Detection and Complex Event Processing over IoT Data Streams*, P. Schneider and F. Xhafa, Eds. Academic Press, 2022, pp. 149–191. doi: 10.1016/B978-0-12-823818-9.00019-5.

[2] A. Singh, "Difference between Feature Selection and Feature Extraction," GeeksforGeeks, Aug. 30, 2022. [Online]. Available: https://www.geeksforgeeks.org/difference-between-feature-selection-and-feature-extraction/

[3] B. Yaghooti, N. Raviv, and B. Sinopoli, "Gram-Schmidt Methods for Unsupervised Feature Extraction and Selection," arXiv preprint arXiv:2311.09386, Nov. 2023. [Online]. Available: https://arxiv.org/abs/2311.09386

[4] A. B. Hall et al., "Genomic diversity of 39,950 global SARS-CoV-2 isolates and their implications for the pandemic," *Nature Reviews Genetics*, vol. 23, pp. 563–576, 2022. doi: 10.1038/s43586-022-00184-w.

[5] N. Hsieh and C. Cheng, "Classifying Handwritten Digits Through Single Boolean and Multi-class Classifiers," Department of Electrical and Systems Engineering, Washington University in St. Louis, 2024.