Chapter 5. Knowledge-Driven Dimension Reduction and Reduced Order Surrogate Models

Table of Contents

Abstract

This chapter focuses on the knowledge-driven dimension reduction aspect of mechanistic data science. Two types of dimension reduction methods that are introduced in this chapter: clustering and reduced order modeling. Clustering aims to reduce the total number of data points in a dataset by grouping similar data points into clusters. The datapoints within a cluster are considered to be more like each other than datapoints in other clusters. There are multiple methods and algorithms for clustering. In the first part of this chapter, three clustering algorithms are presented, ranging from entry level to advanced level: the Jenks natural breaks, K-means clustering, and self-organizing map (SOM). Clustering is a form of dimension reduction that reduces the total number of data points. In the second part of this chapter, Singular Value Decomposition (SVD) and Principal Component Analysis (PCA) will be introduced as a reduced order modeling technique that reduce the number of features by eliminating redundant and dependent features, leading to a new set of principal features. The resulting model is called a

reduced order model. Proper Generalized Decomposition (PGD) is a higher order extension of PCA and will also be introduced.

Keywords: Dimension reduction, K-means Clustering, Self-organizing map (SOM), Reduced Order Surrogate Model, Singular Value Decomposition (SVD), Principal component analysis (PCA), Proper generalized decomposition (PGD), Spring Mass System, Variance, Covariance, Modal Superposition, Eigenvalues, Eigenvectors

5.1. Introduction

Clustering (sometimes called cluster analysis) is a general task of grouping a set of datapoints so that datapoints within the same group (called a cluster) are more similar to each other than datapoints in other clusters. Cluster analysis originated in the field of anthropology by Driver and Kroeber in 1932[1] and was later introduced to psychology by Joseph Zubin in 1938[2] and Robert Tryon in 1939[3]. It was famously used by Cattell beginning in 1943[4] for trait theory classification in personality psychology. In 1967, George Frederick Jenks proposed the Jenks optimization method[5], also called the Jenks natural breaks classification method, which is a data clustering method designed to determine the best arrangement of values into different classes. The term "k-means" was first used by James MacQueen in 1967[6]. In recent years, a lot of clustering methods have been developed including k-medians clustering[7], hierarchical clustering[8] (1977), and self-organizing map (2007)[9].

Reduced order modeling is a second type of dimension reduction technique that aims to reduce the number of features by eliminating redundant and dependent features and obtaining a set of principal features. Two classical and highly related dimension reduction methods, Singular Value Decomposition (SVD) and Principal Component Analysis (PCA), are introduced in

[1] Driver and Kroeber (1932). "Quantitative Expression of Cultural Relationships". University of California Publications in American Archaeology and Ethnology. Quantitative Expression of Cultural Relationships: 211–256

[2] Zubin, Joseph (1938). "A technique for measuring like-mindedness". The Journal of Abnormal and Social Psychology. 33 (4): 508–516.

[3] Tryon, Robert C. (1939). Cluster Analysis: Correlation Profile and Orthometric (factor) Analysis for the Isolation of Unities in Mind and Personality. Edwards Brothers.

[4] Cattell, R. B. (1943). "The description of personality: Basic traits resolved into clusters". Journal of Abnormal and Social Psychology. 38 (4): 476–506.

[5] Jenks, George F. 1967. "The Data Model Concept in Statistical Mapping", International Yearbook of Cartography 7: 186–190.

[6] MacQueen, J. B. (1967). Some Methods for classification and Analysis of Multivariate Observations. Proceedings of 5th Berkeley Symposium on Mathematical Statistics and Probability. 1. University of California Press. pp. 281–297.

[7] A. K. Jain and R. C. Dubes, Algorithms for Clustering Data. Prentice-Hall, 1988.

[8] D. Defays (1977). "An efficient algorithm for a complete-link method". The Computer Journal. British Computer Society. 20 (4): 364–366.

[9] Kohonen, Teuvo; Honkela, Timo (2007). "Kohonen Network". Scholarpedia. 2 (1): 1568.

this chapter. SVD was originally developed by differential geometry researchers Eugenio Beltrami and Camille Jordan in 1873 and 1874, respectively[10]. PCA was independently invented by Karl Pearson (1901)[11] and Harold Hotelling (1933)[12]. The resulting model after dimension reduction is called a reduced order model (ROM). Proper Generalized Decomposition (PGD) (2006)[13] is introduced as a generalized ROM that is a higher order extension of PCA and SVD.

5.2. Dimension reduction by clustering

5.2.1. Clustering in real life: Jogging

The analysis of jogging performance is used to as an application of clustering. Figure 1 below shows a sample of some jogging data collected by smartphone apps. The data include the jogging date, distance, duration, and the number of days since the last workout. The days since last workout vs. distance is plotted. The data were clustered into 5 groups using a k-means clustering algorithm (to be discussed later in this chapter), with each cluster indicated by the different colors.

[10] https://en.wikipedia.org/wiki/Singular_value_decomposition#History

[11] Pearson, K. (1901). "On Lines and Planes of Closest Fit to Systems of Points in Space". Philosophical Magazine. 2 (11): 559–572

[12] Hotelling, H. (1933). Analysis of a complex of statistical variables into principal components. Journal of Educational Psychology, 24, 417–441, and 498–520.

[13] Amine Ammar, Béchir Mokdad, Francisco Chinesta, Roland Keunings (2006). "A New Family of Solvers for Some Classes of Multidimensional Partial Differential Equations Encountered in Kinetic Theory Modeling of Complex Fluids". Journal of Non-Newtonian Fluid Mechanics.

Dataset:

Date	Duration (min)	Distance (km)	Days since last workout
Oct. 17, 2020	21.58	4.3	1
Nov. 14, 2020	9.25	1.9	18
Nov. 18, 2020	9	1.9	14
Nov. 23, 2020	8.93	1.9	5
Nov. 28, 2020	11.94	2.3	5
Nov. 29, 2020	14.05	2.8	1
…	…	…	…

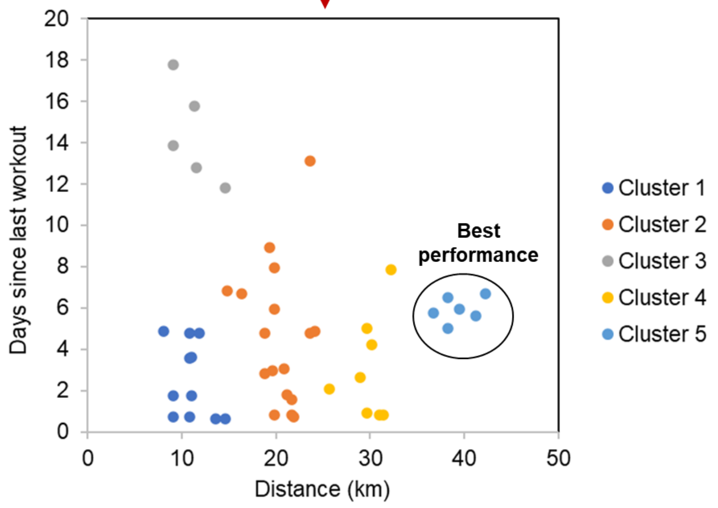


Figure 1 The jogging workout session dataset clustered into 5 distinct groups.

Clustering allows the identification of similar workout sessions and the relationship to overall performance. For example, the days since last workout vs. distance shows that the jogging distance was shortest after a large number of rest days (Cluster 3 in the Figure 1), but also after only a few days of rest. The longest jogging distances were achieved when there was approximately 6 days since last workout (Cluster 5 in the Figure 1). Therefore, clustering analysis shows that resting 5-7 days between workouts results in the best performance in terms of jogging distance.

5.2.2. Clustering for diamond price: from Jenks natural breaks to k-means clustering

Jenks natural breaks is a very intuitive data clustering algorithm proposed by George Frederick Jenks in 1967[14]. “Natural breaks” divides the data into ranges that minimize the variation within each range. To illustrate the idea of the Jenks natural breaks, consider four diamonds shown in

[14] Jenks, George F. 1967. "The Data Model Concept in Statistical Mapping", International Yearbook of Cartography 7: 186–190.

Figure 2. The goal of the clustering is to form two groups or clusters from these four diamonds based on their prices, i.e., \$300, \$400, \$1000, and \$1200.

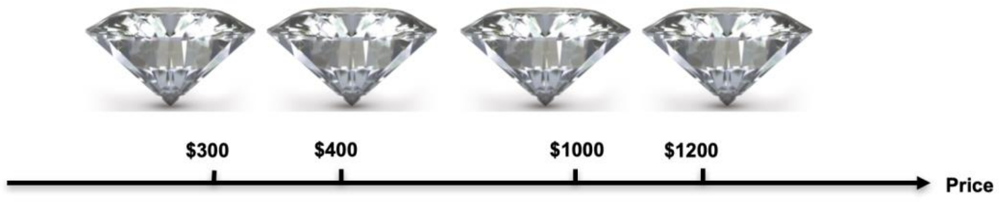


Figure 2 Classifying four diamonds with known prices.

The first step of the Jenks natural breaks is calculating the "sum of squared deviations from <u>array</u> mean (SDAM)" based on the data array (order from smallest to largest):

$$\text{array} = [300, 400, 1000, 1200] \tag{5.1}$$

The mean of the array can be computed first:

$$\text{mean} = (300 + 400 + 1000 + 1200)/4 \ = 725 \tag{5.2}$$

SDAM is equal to variance of the data:

$$\text{SDAM} = (300 - 725)^2 + (400 - 725)^2 + (1000 - 725)^2 + (1200 - 725)^2 = 587{,}500 \tag{5.3}$$

The second step is to compute all possible range combinations and calculate the "sum of squared deviations from <u>class</u> means" (SDCM) and identify the smallest one. For this diamond price data, there are three range combinations:

1. group 1: [300] and group 2: [400,1000,1200]

$$\text{SDCM} = (300 - 300)^2 + (400 - 867)^2 + (1000 - 867)^2 + (1200 - 867)^2 = 346{,}667 \tag{5.4}$$

2. group 1: [300, 400] and group 2: [1000,1200]

$$\text{SDCM} = (300 - 350)^2 + (400 - 350)^2 + (1000 - 1150)^2 + (1200 - 1150)^2 = 25{,}000 \tag{5.5}$$

3. group 1: [300, 400, 1000] and group 2: [1200]

$$\text{SDCM} = (300 - 567)^2 + (400 - 567)^2 + (1000 - 567)^2 + (1200 - 1200)^2 = 286{,}667 \tag{5.6}$$

Based on this calculation, combination 2 has the smallest SDCM, implying that it is the best clustering.

The final step is to calculate the "goodness of variance fit" (**GVF**)[15], defined as

$$\text{GVF} = \frac{\text{SDAM} - \text{SDCM}}{\text{SDAM}} \tag{5.7}$$

GVF ranges from 1 (perfect fit) to 0 (poor fit).

The GVF for range combination 2 is

[15] https://medium.com/analytics-vidhya/jenks-natural-breaks-best-range-finder-algorithm-8d1907192051

$$\frac{(587{,}500 - 25{,}000)}{587{,}500} = 0.96 \tag{5.8}$$

The GVF for range combination 1 is

$$\frac{587{,}500 - 346{,}667}{587{,}500} = 0.41 \tag{5.9}$$

In this data set, the range combination group 1: $[300, 400]$ and group 2: $[1000,1200]$ is best because it has the lowest SDCM of all possible combinations and has a GVF close to 1. The two groups are shown in Figure 3 highlighted with different colors (red and blue).

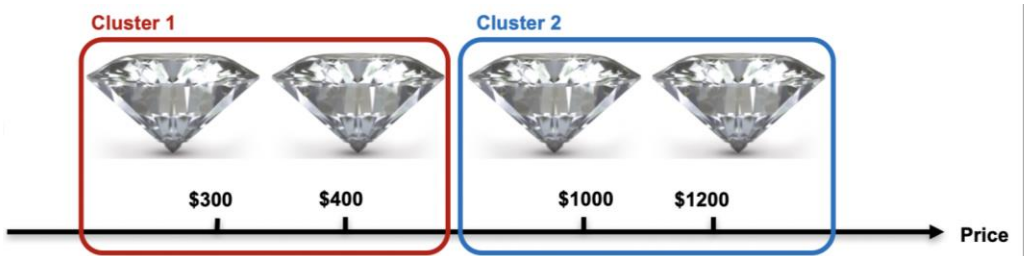
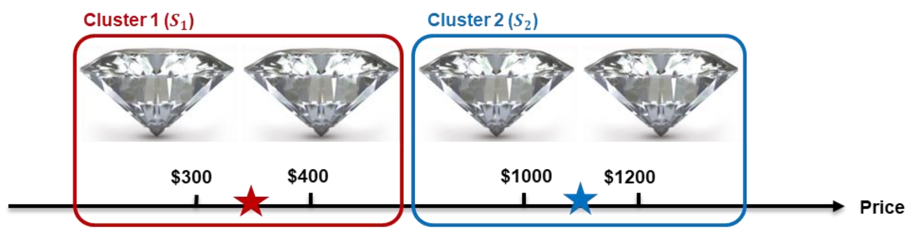


Figure 3 Clustering of diamonds based on Jenks Natural Breaks method

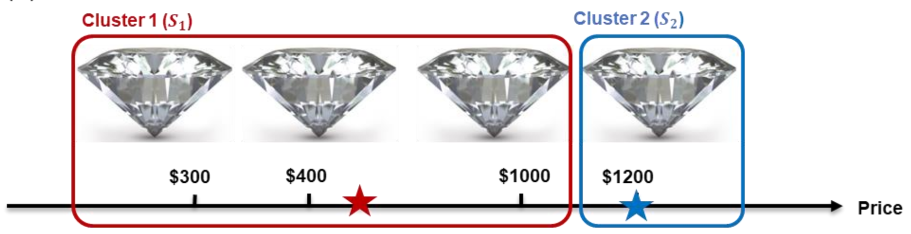
The goal of the Jenks natural breaks (and many other clustering methods) is to minimize the SDCM. A lower SDCM equates to better clustering because SDCM is related to the distance of the data points from the means of their clusters. Thus, a lower SDCM indicates that the data points within a cluster are close together and more similar. For example, Figure 4 shows two different clustering results for the diamond dataset. Figure 4(a) has an SDCM of 25,000 while Figure 4(b) has an SDCM of 286,667 (the calculations for mean and SDCM are also marked in the figure). It can be noticed that the data points are closer to the means (marked as stars with different colors for different clusters) in Figure 4(a) than that in Figure 4(b).

(a)



Mean=[350, 1100]
SDCM = (*300* - *350*)^2+(*400* - 350)^2+(1000 - 1150)^2+(1200 - 1150)^2 = **25,000**

(b)



Mean=[567, 1100]
SDCM = (*300* - *567*)^2+(*400* - 567)^2+(1000 - 567)^2+(1200 - 1200)^2 = **286,667**

Figure 4 Visualization of two different clustering results: (a) $[300,400]$ *and* $[1000,1200]$ *and (b)* $[300,400,1000]$ *and* $[1200]$*. Each box indicates a cluster. The stars are mean of each cluster.*

From this example, it is evident that Jenks natural breaks method works well for 1D data. However, it is inefficient because it has to go through all the possible arranging combinations. For example, clustering 254 data points into 6 clusters has $C(253,5) = 8{,}301{,}429{,}675$ possible range combinations. It is very time consuming to calculate the means and SDCMs for the more than eight billion combinations.

Where does $C(253,5) = 8{,}301{,}429{,}675$ come from?

Recall: Stars and Bars combinatorics

Suppose *n* **stars** are to be divided into *k* distinguishable **groups**, in which each group contains at least one star. Additionally, the stars have a fixed order. *k* – 1 **bars** are needed to divide *n* stars into *k* groups. See the example below of 7 stars divided into 3 groups:

★|★|★★★★★ (*n* = 7, *k* = 3)

The position of the bars between the stars matters. There are *n* – 1 gaps between the stars, each which may or may not contain a bar. Thus, this becomes a simple combination problem, in which the total number of possibilities for **choosing which *k* – 1 gaps out of *n* – 1 gaps contain a bar** can be calculated.

This can be represented as $C(n-1, k-1)$, $\binom{n-1}{k-1}$, or $\frac{(n-1)!}{(k-1)!(n-k)!}$

To overcome the limitation of the Jenks natural breaks, a more efficient and widely used clustering method, called k-means clustering, is introduced. This clustering method divides the data points into k clusters (hence the "k" in k-means) and finds the center point for each of the clusters. The center points are moved around, and some points are moved from cluster to cluster until the tightest collection of data points is found for each cluster center point (or mean). When written mathematically, the goal of k-means clustering is to minimize the SDCM

$$\underset{\boldsymbol{\mu_j}}{\operatorname{argmin}} \sum_{j=1}^{K} \sum_{x_i \in \boldsymbol{S_j}} \left\| \boldsymbol{x_i} - \boldsymbol{\mu_j} \right\|^2 \tag{5.10}$$

where $\boldsymbol{x_i}$ is data point i in the j^{th} cluster $\boldsymbol{S_j}$, $\boldsymbol{\mu_j}$ is the center point (or mean) of the cluster $\boldsymbol{S_j}$, $\|\cdot\|^2$ is the square distance, and K is the number of clusters.

The same diamond data from the Jenks natural breaks example is used to explain k-means clustering with two clusters.

1. For the first iteration, the center points of the two clusters are randomly assigned, for example initial $\text{means} = [1100, 1200]$ (marked as stars in Figure 5).
2. The distance of each data point to each of the k means is calculated:

$$\text{Distance to the mean 1} = [800, 700, 100, 100] \tag{5.11}$$

$$\text{Distance to the mean 2} = [900, 800, 200, \quad 0] \tag{5.12}$$

3. Based on the distance to the mean, each data point is labeled as belonging to the nearest mean:

$$\text{Label} = [1, 1, 1, 2] \tag{5.13}$$

4. The means for the groups of labeled points then can be updated

$$\text{means} = [567, 1200] \tag{5.14}$$

5. The updated means in Step 4 are compared with the previous iteration. If the

difference is less than a specified tolerance, the optimum clustering is found. If not, the algorithm returns to step 2.

For this example, only three iterations are needed for the converged result of $\text{means} = [350, 1100]$ (see Figure 6). Note that k-means clustering ends up with the same result as the Jenks natural breaks.

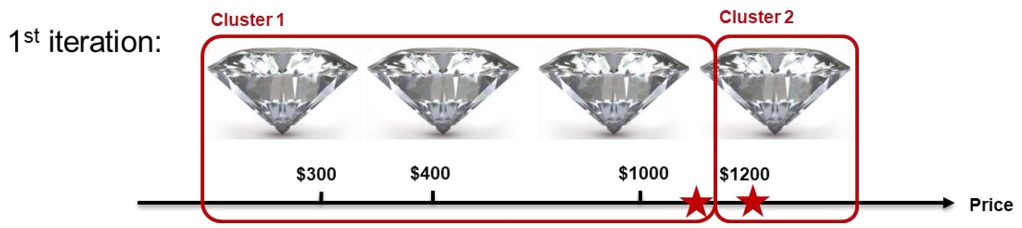


Figure 5 The initial means are randomly assigned at the first iteration

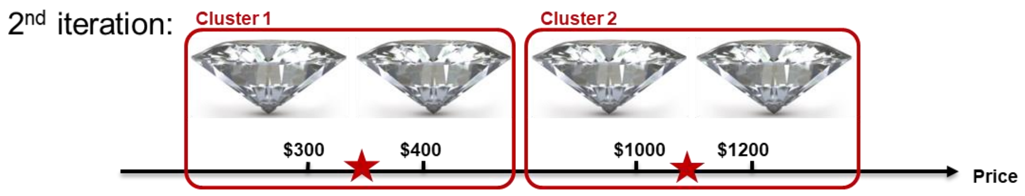
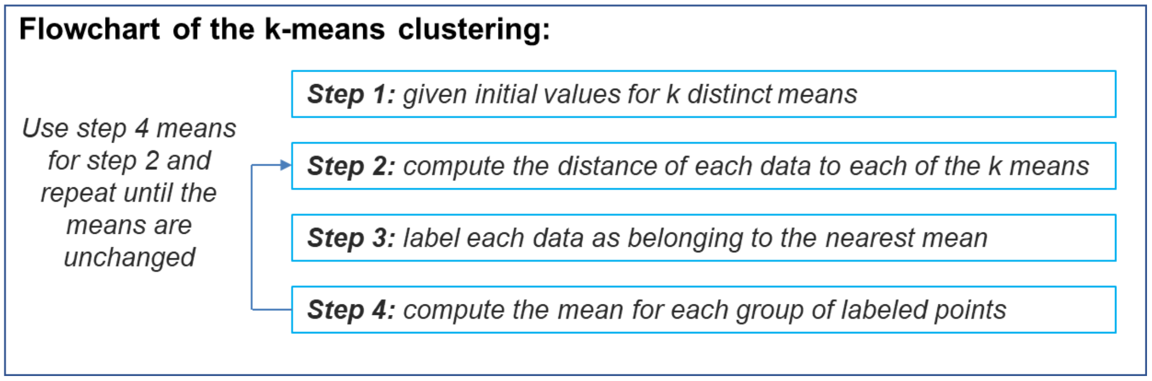


Figure 6 The cluster means at the second iteration

The k-means clustering is much more efficient than the Jenks natural breaks. In summary the procedure of the k-means clustering is shown in Figure 7.



Flowchart of the k-means clustering:

Step 1: *given initial values for k distinct means*

Step 2: *compute the distance of each data to each of the k means*

Step 3: *label each data as belonging to the nearest mean*

Step 4: *compute the mean for each group of labeled points*

Use step 4 means for step 2 and repeat until the means are unchanged

Figure 7 The procedure of the k-means clustering

5.2.3. K-means clustering for high-dimensional data

Higher dimension data can be clustered very similarly using k-means clustering. The dimension of the data is determined by the number of features. Table 1 lists the differences in clustering for 1D, 2D, and 3D data. It can be seen that for higher dimensional clustering the data points and the means are vectors instead of scalars.

Table 1. Comparing k-means clustering for one-, two-, and three-dimensional data.

x = data point, N = number data points, y = mean. **Bold** indicates vector.

	1D	**2D**	**3D**
Data	x	$\boldsymbol{x} = \begin{pmatrix} x_1 \\ x_2 \end{pmatrix}$	$\boldsymbol{x} = \begin{pmatrix} x_1 \\ x_2 \\ x_3 \end{pmatrix}$
Mean	$y = \frac{\sum x}{N}$	$\boldsymbol{y} = \frac{\sum \boldsymbol{x}}{N}$	$\boldsymbol{y} = \frac{\sum \boldsymbol{x}}{N}$
Distance	$\lvert x - y \rvert$	$\lVert \boldsymbol{x} - \boldsymbol{y} \rVert$	$\lVert \boldsymbol{x} - \boldsymbol{y} \rVert$

Example: clustering of diamonds based on multiple features

The diamond dataset from Chapter 2 is clustered based on multiple key features. Initially, two key features (price and carat) are used, and the data is divided into four clusters. The results of this 2D clustering are shown in Figure 8. Based on the k-means clustered data, the clusters of high-end diamonds (black cluster) and economical diamonds (yellow cluster) can be identified.

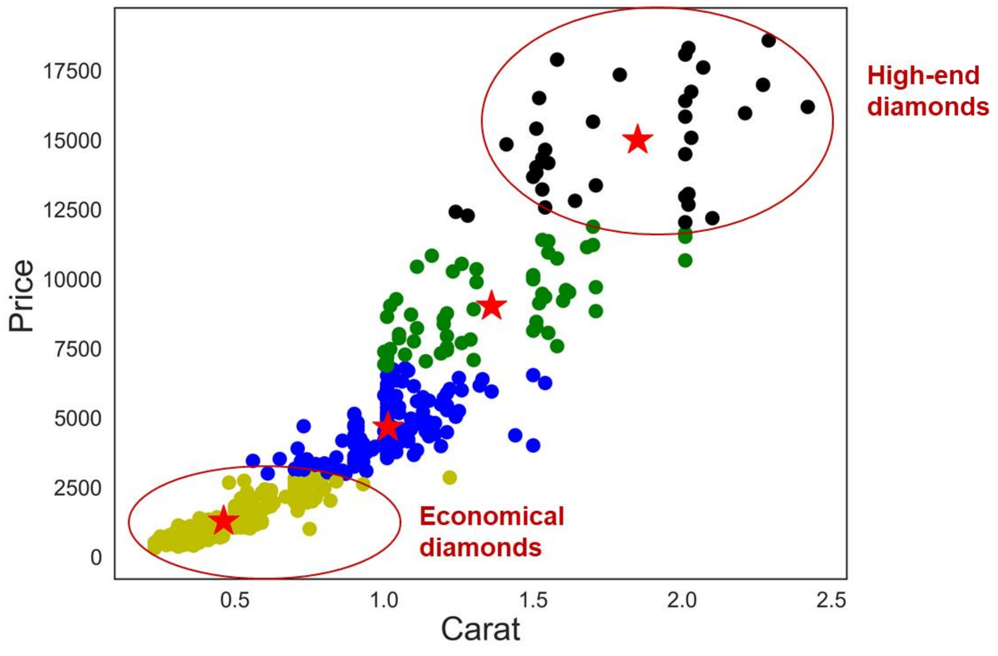


Figure 8 Diamond dataset is clustered into 4 groups based on carat and price. There are 539 data points used for the clustering.

Additional features can be used for clustering. Clustering for 3D data follows a similar method. Figure 9 shows the data for 2,695 diamonds clustered based on price, clarity, and carat.

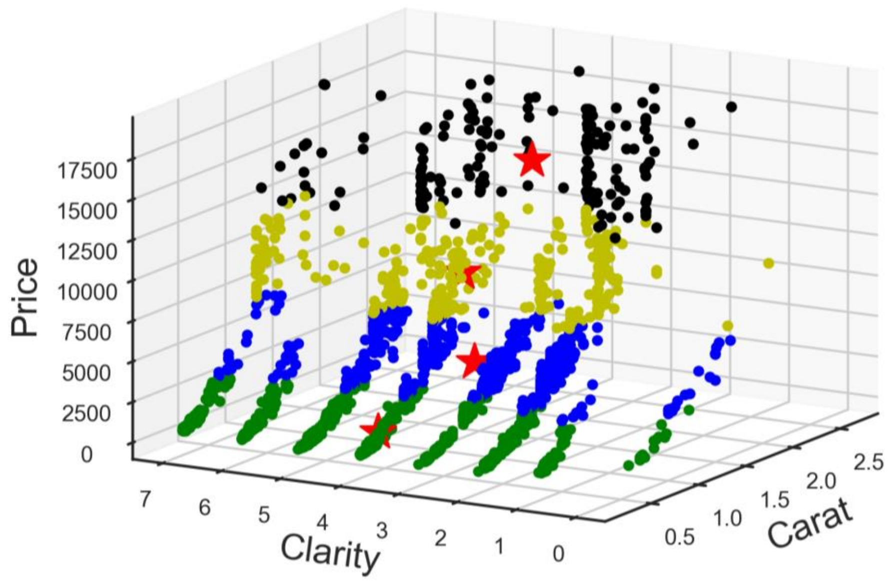


Figure 9 Diamonds are clustered into four groups based on price, clarity, and carat. There are 2695 data points used for the clustering. The order of the colors is randomly generated by the algorithm. It might be different for each run.

5.2.4. Determining the number of clusters

Before performing k-means clustering, the ideal number of clusters should be determined to properly represent the data. This can be accomplished using the "elbow method". This method is based on the goodness of variance fit (GVF) as discussed previously. The GVF is defined as

$$\mathrm{GVF} = \frac{\mathrm{SDAM} - \mathrm{SDCM}}{\mathrm{SDAM}} \tag{5.15}$$

where GVF ranges from 1 (perfect fit) to 0 (poor fit), SDAM is the sum of squared deviations from array mean, and SDCM is the sum of squared deviations from class mean.

If GVF is graphed versus the number of clusters, an "elbow" will appear in the curve when the effect of increasing k begins to have a decreased effect on the GVF. For example, as shown in Figure 10 the elbow appears to occur at $K = 3$ or 4 for the diamond data.

It is noted that the elbow method is somewhat subjective because the elbow occurs in a region and not a precise location. This is one of the limitations of k-means clustering. Other limitations of the k-means clustering will be discussed in the next section.

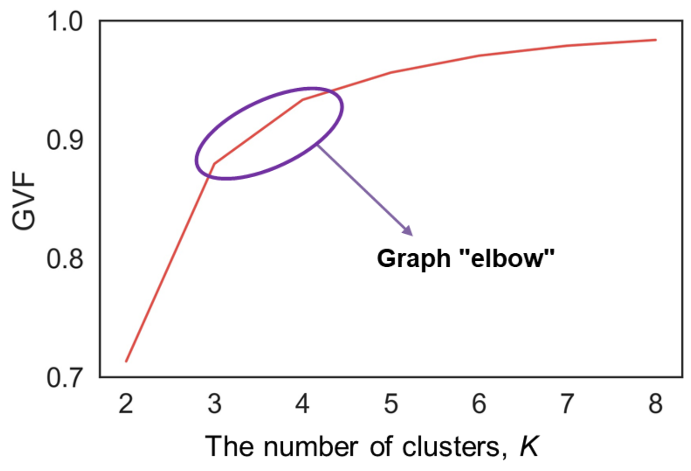


Figure 10 Using the elbow method on GVF vs. the number of clusters to determine the best number of clusters.

5.2.5. Limitations of k-means clustering

Although k-means clustering is a widely used way to reduce data dimension, there are a couple of notable limitations. First, the user must specify the number of clusters, K. Some methods such as elbow method can be used to determine the best number of clusters, but most of them are done visually by the user's judgment and not automatically by the method itself. This makes it difficult to completely automate the clustering process. Next, k-means clustering can only handle numerical or categorical data. It cannot handle images or words. Finally, k-means clustering assumes that we are dealing with spherical clusters and that each cluster has roughly an equal number of observations. Those limitations can be overcome (or partially overcome) by other types of advanced clustering methods such as Hierarchical Clustering/Dendrograms[16] and Self-organizing maps (SOM)[17], in which the number of clusters does not need to be specified before the clustering. The SOM and its applications will be introduced in the next section as an advanced topic.

5.2.6. Self-organizing map (SOM) [Advanced topic]

The self-organizing map (SOM) is an unsupervised machine learning algorithm that is able to map high-dimensional data to two-dimensional (2D) planes while preserving topology[18]. The main advantage of the SOM is that it can visualize high-dimensional data in the form of a low-dimensional map, which helps researchers to visually identify underlying relations between the

[16] https://en.wikipedia.org/wiki/Hierarchical_clustering

[17] Kohonen, Teuvo. "The self-organizing map." Proceedings of the IEEE 78.9 (1990): 1464-1480.

[18] Rauber, A., Merkl, D., & Dittenbach, M. (2002). The growing hierarchical self-organizing map: exploratory analysis of high-dimensional data. IEEE Transactions on Neural Networks, 13(6), 1331-1341.

features. As a tool to visualize high-dimensional datasets, the SOM is beneficial for the cluster analysis of engineering design problems as well.

The goal of a self-organizing map (SOM) is to reduce the dimension of data points by representing the topology of the data with fewer points, using a map of neurons, and converting the dataset into two dimensions. The topology refers to the relative distance between points, meaning that there will be more neurons in areas where data is more condensed. These neurons can be treated as mini-clusters. In the map, each neuron is connected to its surrounding neurons, which are referred to as "neighbors." Because the map preserves the topology of the data, neighboring neurons will describe a similar number of data points. The maps are always two-dimensional, even when used on higher dimensional data, allowing us to assess the clustering of higher dimensions in just two dimensions.

The goal of the SOM is to put "similar" datapoints into the same SOM neuron (mini-cluster) and weight $\boldsymbol{W_{ij}}$ of each neuron represents the average of the included datapoints. For example, an illustrative 8*8 SOM is shown in Figure 11. The labels of the X and Y axes are the integers $1 \leq i \leq SizeX$, and $1 \leq j \leq SizeY$, respectively.

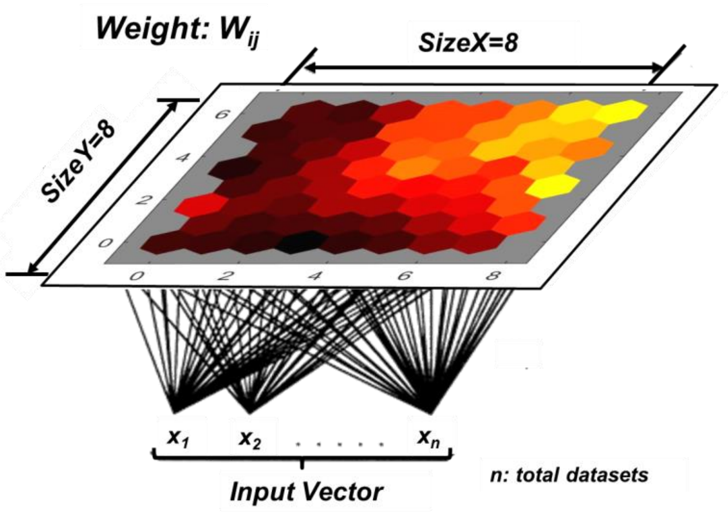


*Figure 11 An illustrative 8*8 SOM*

To achieve a convergent result, SOM uses competitive learning, rather than error-correction learning (like back propagation[19]). The training procedure can be described by a pseudo code as shown in Figure 12. Initially, the weights of the neurons are set to be $W_0^{i,j}$ with random number [0, 1]. The elements of each input vector are normalized linearly to [0, 1]. After initialization, the SOM is trained for a number of T epochs. For the current epoch t and each input vector $\boldsymbol{x_m}$, first, the best matching unit (BMU) is determined by calculating the distance between the input vector and each neuron weight using Equation (1) in Figure 12. The BMU is a map unit has the shortest distance to input vector $\boldsymbol{x_m}$. Second, the diameter of the neighborhood around the BMU can be determined by Equation (2) in Figure 12, where d(t) is a function that decreases monotonously

[19] Goodfellow, Bengio & Courville 2016, p. 200, "The back-propagation algorithm (Rumelhart et al., 1986a)

with time. The initial distance coefficient is d_0, and the decrease rate is λ. Third, the weights $W_m^{i,j}$ in the BMU and its neighborhoods are updated according to Equations (3) and (4) in Figure 12. In Equations (3) and (4), $h_{BMU,i,j}$ represents the Gaussian kernel function, where $\alpha(t)$ is a learning rate parameter, $r_{i,j}$ is the position of each unit, and r_{BMU} is the position of the BMU. The current epoch is finished after all the $\boldsymbol{x_m}$ have been calculated to the SOM. By selecting a large enough number of epochs, for example 100, the SOM can converge. When the training is finished, the map can reorder the original datasets while preserving the topological properties of the input space.

```
Program Self-Organizing Map
Initialize weights of neurons $\boldsymbol{W_0^{i,j}}$ with random number [0, 1]
Index of epoch: $\boldsymbol{t}$
Index of input vector in the dataset: $\boldsymbol{m}$
Do $\boldsymbol{t = 0}$ to $\boldsymbol{T}$ (Max index of epoch)
    Do $\boldsymbol{m = 1}$ to $\boldsymbol{n}$ (Max index of input vector)
      Find Best Matching Unit (BMU):
      $\boldsymbol{W_m^{BMU} = \underset{\substack{i=1...sizeX\\ j=1...sizeY}}{argmin} \left\| W_m^{ij} - x_m \right\|^2}$    (1)
      Determine diameter of neighborhood around BMU:
      $\boldsymbol{d(t) = d_0 exp\left(-\frac{t}{\lambda}\right)}$    (2)
      Update weights $\boldsymbol{W_{m+1}^{i,j}}$ in BMU and neighborhoods:
      $\boldsymbol{W_{m+1}^{ij} = W_m^{ij} + h_{BMU,i,j}\left(x_m - W_m^{ij}\right)}$    (3)
      $\boldsymbol{h_{BMU,i,j} = \alpha(t) exp\left(-\frac{\left\| r_{i,j} - r_{BMU} \right\|^2}{2d^2(t)}\right)}$    (4)
    End do
End do
```

Figure 12 SOM algorithm

An engineering example: data-driven design for additive manufacturing using SOM

The SOM algorithm is demonstrated for visualizing high-dimensional data in additive manufacturing (AM). These data are obtained from well-designed experimental measurements and multiphysics models. The SOM is introduced to find the relationships among process-structure-properties (PSP) in AM, including laser power, mass flow rate, energy density, dilution, cooling rate, dendrite arm spacing, and microhardness. These data-driven linkages between process, structure, and properties have the potential to benefit online process monitoring control in order to derive an ideal microstructure and mechanical properties. In addition, the design windows of process parameters under multiple objectives can be obtained from the visualized SOM. A schematic diagram of this work is shown in Figure 13.

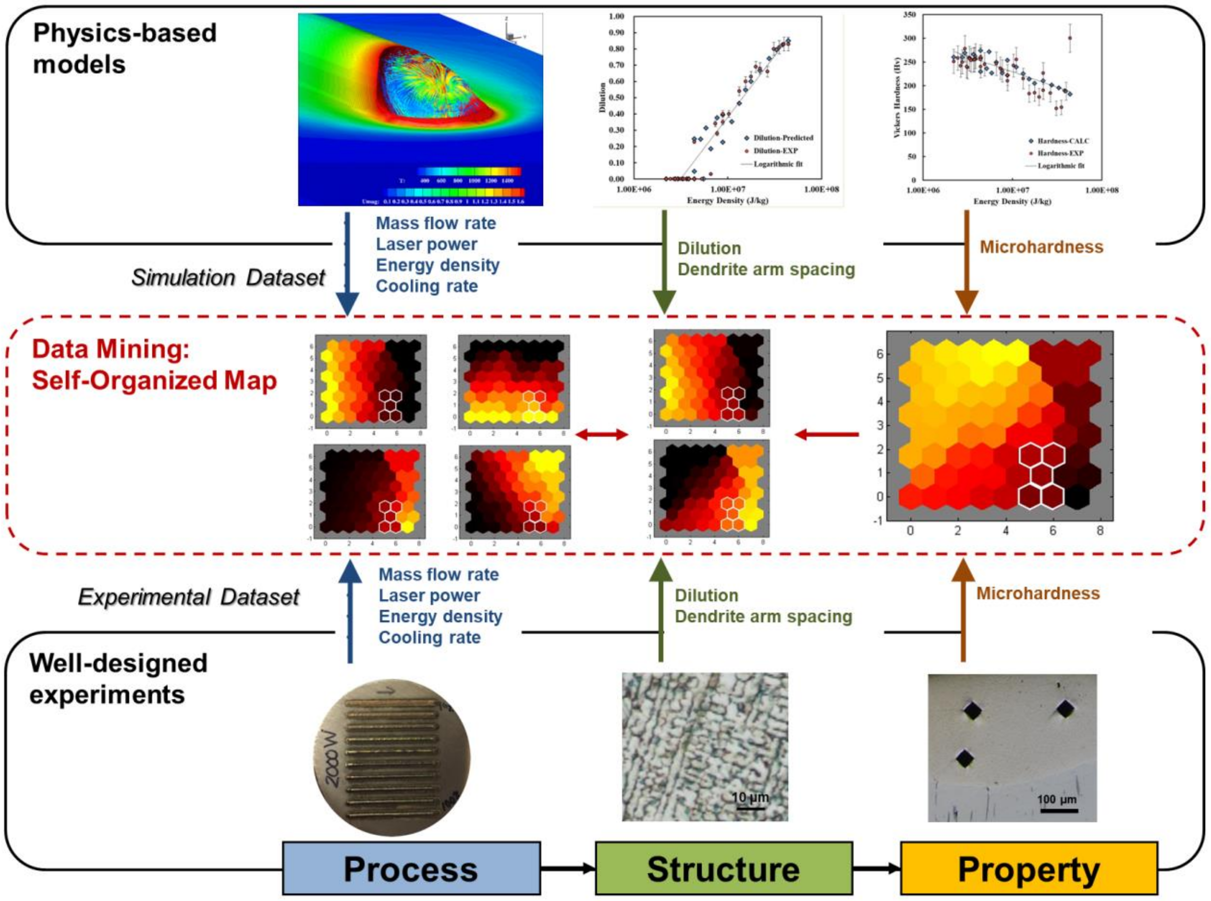


Figure 13 A schematic description of the workflow typically employed in current computational efforts (top row) and of experimental efforts (bottom row), along with a description of how this can be augmented with a data-mining approach to recover high-value PSP linkages of interest to material innovation efforts.

Sixty single-track AM experiments using various process parameters were conducted for data generation. Materials characterization in this case included cooling rate measurements, dilution measurements, dendrite arm spacing measurements, and hardness testing. In addition, a computational thermal-fluid dynamics (CtFD) model was developed to simulate the AM process. In total, 25 simulation cases with various laser power levels and mass flow rates were computed. For each case, the structures and properties observed were the melt pool geometry, dilution, cooling rate, secondary dendrite arm spacing (SDAS), and microhardness. Details of experiments and simulations can be found in Refs[20, 21].

The SOM Toolbox in Matlab[22] was used to simultaneously visualize high-dimensional datasets and design process parameters. Using physics-based simulations and experimental measurements, seven-dimensional (7D) AM dataset was obtained for data mining including laser power, mass flow rate, energy density, cooling rate, dilution, SDAS, and microhardness. The Matlab code for SOM is shown below. The file 'Training data from AM (all).xlsx' includes the 7D AM dataset.

Matlab code for SOM:

```
Dataset=xlsread('Training data from AM (all).xlsx','data');

SOM = selforgmap([8 8],100,6);

SOM = train(SOM, Dataset');

view(SOM);

y = SOM(Dataset');

classes = vec2ind(y);
```

The simulation and experimental data points were used as input vectors to train a single 8 × 8 SOM indistinctively. It is found that the 8 × 8 SOM has the best performance. If the map size is too small, the map resolution is very low; however, an SOM that is too large results in overfitting. Trained SOMs are shown in Figure 14. The relation between the PSP variables can be understood visually. For example, the mass flow rate and SDAS are positively correlated, as the component planes of the mass flow rate and SDAS have similar values at similar positions. Conversely, the mass flow rate and dilution are highly negatively correlated. Thus, according to the visualized SOM in Figure 14, several results are obtained (1) the mass flow rate, more than laser power, greatly contributes to the cooling rate and SDAS; (2) the dilution and microhardness depend on both the mass flow rate and laser power; (3) the microhardness is dominated by the dilution, rather than by the SDAS or cooling rate.

Obtained through the data-mining approach, these relations provide valuable insight into the complex underlying physical phenomena and material evolution during the AM process. In addition, it is possible to obtain the desired process parameter window with multiple objective microstructure and property ranges. In this study, the objective dilution is from 0.1 to 0.3. In this

[20] Gan, Z., Li, H., Wolff, S.J., Bennett, J.L., Hyatt, G., Wagner, G.J., Cao, J. and Liu, W.K., 2019. Data-driven microstructure and microhardness design in additive manufacturing using a self-organizing map. Engineering, 5(4), pp.730-735.

[21] Wolff, S.J., Gan, Z., Lin, S., Bennett, J.L., Yan, W., Hyatt, G., Ehmann, K.F., Wagner, G.J., Liu, W.K. and Cao, J., 2019. Experimentally validated predictions of thermal history and microhardness in laser-deposited Inconel 718 on carbon steel. Additive Manufacturing, 27, pp.540-551.

[22] Vesanto, J., Himberg, J., Alhoniemi, E. and Parhankangas, J., 2000. SOM toolbox for Matlab 5 (Vol. 57, p. 2). Technical report.

range of dilution, the solidified track can avoid both lack of fusion due to low dilution and property degradation due to high dilution[23]. The SDAS should be minimized and the microhardness should be maximized in order to maintain good mechanical properties. An iteration procedure through all the units is undertaken in order to seek units that satisfy these restrictions. An objective cluster that includes multiple units can be selected as a white wireframe, as shown in Figure 14. Thus, the. following desired process parameters can be obtained: a laser power ranging from $1000\ W$ to $1100\ W$ and a mass flow rate ranging from $22.4\ g \cdot min^{-1}$ to $24.8\ g \cdot min^{-1}$. The desired energy density, which is defined as laser power divided by mass flow rate, ranges from $2.4 \times 10^6\ \mathrm{J \cdot kg^{-1}}$ to $2.9 \times 10^6\ \mathrm{J \cdot kg^{-1}}$. The SOM approach can be applied to a broad variety of PSP datasets for AM and other data-intensive processes. Data-driven relationships between process, structure, and property can provide online monitoring and process control to derive ideal microstructure and mechanical properties.

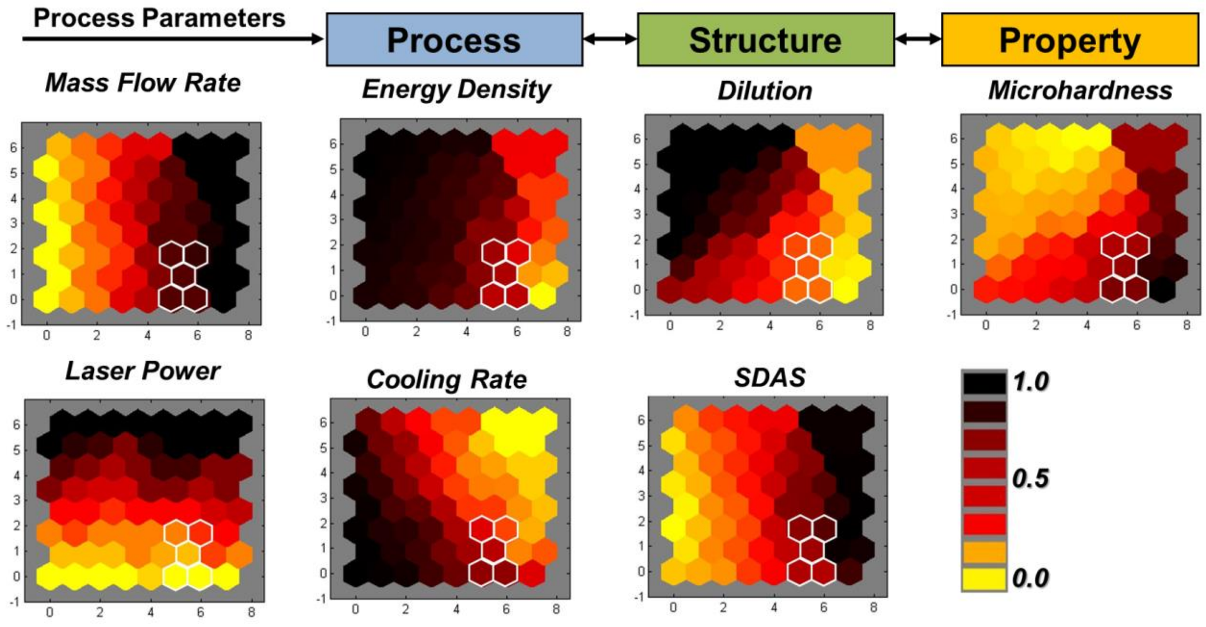


Figure 14 Contour plots of all design variables with the optimized design window outlined by a white wireframe.

5.3. Reduced order surrogate models

5.3.1. A first look at principal component analysis (PCA)

Principal component analysis (PCA) is a dimension reduction or reduced order modeling technique that seeks to determine the directions or components with maximum variability. This can be explained conceptually by considering the images of the teapot in Figure 15. After observing the teapot from all angles, it can be seen that the orientation showing the most variation is the one that goes from the handle to the spout (see the direction with the blue arrow).

[23] Mukherjee, T., Zuback, J. S., De, A., & DebRoy, T. (2016). Printability of alloys for additive manufacturing. Scientific reports, 6(1), 1-8

The directions showing the second and third most variation are bottom to top (see red arrow), and from handle to spout viewed from the top (see green arrow) in Figure 15. These directions are vectors, and the technical terms used to describe the directions of these vectors are principal components. In this example, the blue arrow is the first principal component.

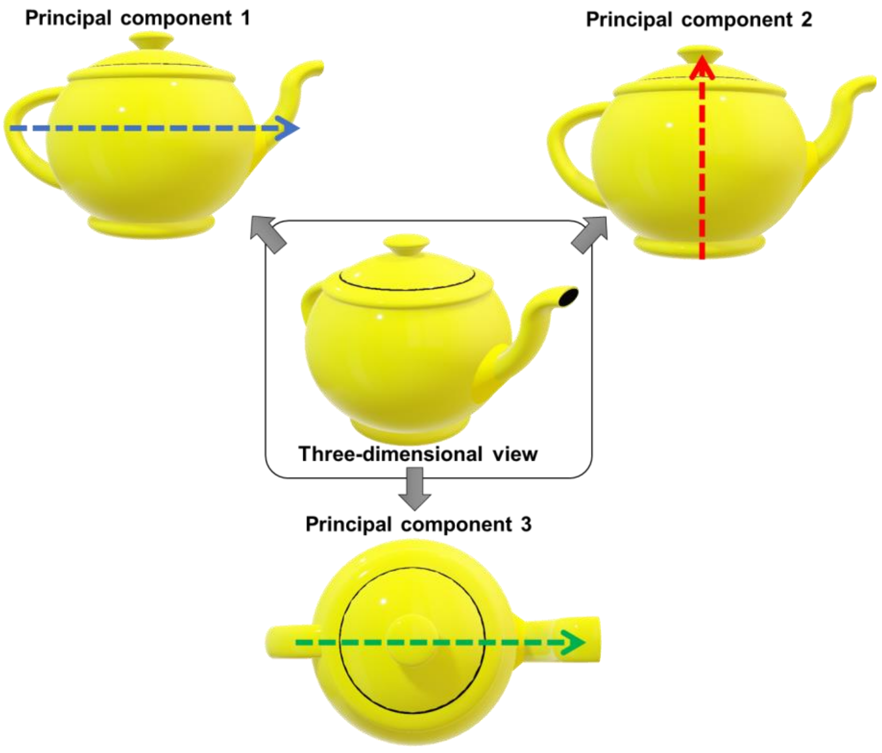


Figure 15 Images of the teapot to conceptually explained principal component

The principal components for a set of data can be computed and printed using the following Python code (assume 3 principal components are desired):

Python code for PCA:

```
from sklearn.decomposition import PCA


pca = PCA(n_components=3,svd_solver='auto')
coordinates = pca.fit_transform(B)
print("principal components=\n",pca.components_)
```

Principal component analysis can be demonstrated using the steel material property data shown in Table 2. There are four datapoints in the dataset (i.e., four kinds of steel). Each datapoint includes two features that are material properties (elongation and UTS) of the steel.

Table 2 Steel material property data[24]

Steel Material	Elongation	Ultimate Tensile Strength (UTS) (MPa)
ASTM A36	0.3	350
API 5L X52	0.21	450
High strength alloy steel	0.18	760
Boron steel	0.07	1500

A reduced order model including only one feature can be constructed using PCA. Based on the principal component (the orientation indicating the most variation) computed from PCA, the data points can be projected to the principal component that is treated as a new coordinate. This creates a new reduced order model to represent the original dataset. The procedure of PCA is

(1) calculate the mean of each feature of the data:

$$\text{Elongation: average} = 0.19 \tag{5.17}$$

$$\text{UTS: average} = 766.25 \tag{5.18}$$

(2) normalize data for each feature around the center point and put into a matrix $\boldsymbol{B}$

$$\boldsymbol{B} = \begin{bmatrix} 0.3-0.19 & 350-766.25 \\ 0.21-0.19 & 450-766.25 \\ 0.18-0.19 & 760-766.25 \\ 0.07-0.19 & 1500-766.25 \end{bmatrix} = \begin{bmatrix} 0.11 & -416.25 \\ 0.02 & -316.25 \\ -0.01 & -6.25 \\ -0.12 & 733.75 \end{bmatrix} \tag{5.19}$$

(3) compute the eigenvalues and eigenvectors for the covariance matrix of $\boldsymbol{B}$ (this step requires linear algebra, see Section 5.4)

$$\boldsymbol{\lambda} = \begin{bmatrix} \lambda_1 \\ \lambda_2 \end{bmatrix} = \text{eigenvalues} = \begin{bmatrix} 2.70\text{e}+05 \\ 9.20\text{e}-04 \end{bmatrix} \tag{5.20}$$

$$\boldsymbol{P} = [\boldsymbol{p_1} \quad \boldsymbol{p_2}] = \text{eigenvectors} = \begin{bmatrix} -1.73\text{e}-04 & 0.999 \\ 0.999 & 1.73\text{e}-04 \end{bmatrix} \tag{5.21}$$

where the eigenvectors are the principal components (e.g., arrows of the teapot in Figure 15). The magnitude of each principal direction is indicated by the eigenvalue. Inspection of the eigenvalues shows that the first eigenvalue is 9 orders of magnitude larger than the second. The first principal component for the steel material properties is computed and plotted in Figure 16. The blue points are the original raw data, and the red arrow represents the first principal component considering only the first eigenvalue. A reduced order model $\boldsymbol{R}$ can be created by projecting the datapoints to the new coordinate: the principal component (eigenvector)

$$\boldsymbol{R} = \boldsymbol{B}\boldsymbol{p_1} \tag{5.22}$$

[24] https://www.api5lx.com/api5lx-grades/

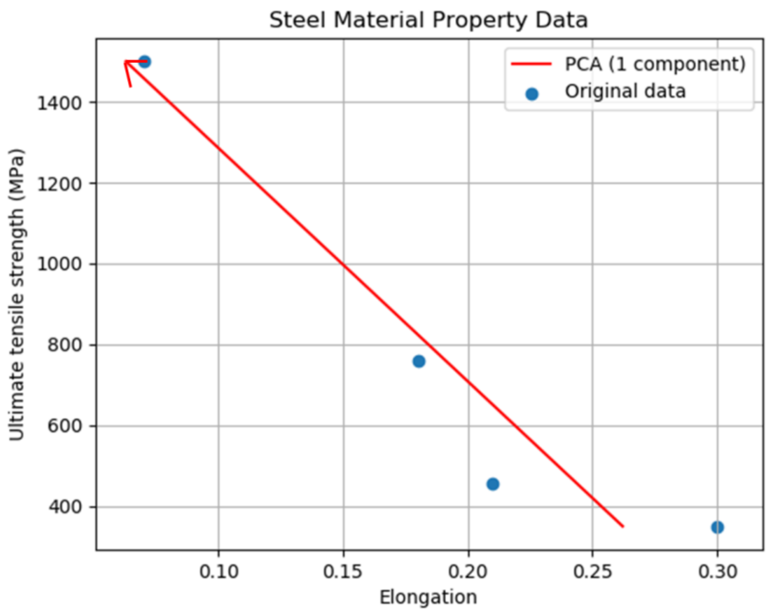


Figure 16 The first principal component of the steel material properties

An executable Python code for PCA:

```
import numpy as np
import matplotlib.pyplot as plt
from sklearn.decomposition import PCA

#  Data for steels
A = np.array([[0.3,350],[0.21,455],[0.18,760],[0.07,1500]])

pca = PCA(1)   # PCA using 1 component
pca.fit(A)

print("eigenvectors = ", pca.components_)
print("eigenvalues = ", pca.explained_variance_)

B = pca.transform(A)
A_pca = pca.inverse_transform(B)

#  Plot out data
plt.scatter(A[:,0],A[:,1])
plt.plot(A_pca[:,0],A_pca[:,1],'r-')
plt.show()
```

5.3.2. Understanding PCA by singular value decomposition (SVD) [Advanced topic]

Singular value decomposition (SVD) is amongst the most important matrix factorizations and is the foundation for many other data-driven methods (such as PCA).

Recall matrix multiplication

The concept of matrix multiplication is important for understanding SVD and PCA. Thus, the multiplication operation of matrix is recalled in this section. Consider a two-by-two matrix $\boldsymbol{A}$

$$\boldsymbol{A} = \begin{bmatrix} 1 & 2 \\ -1 & 1 \end{bmatrix} \tag{5.23}$$

The values in matrix $\boldsymbol{A}$ are assigned arbitrarily. The matrix $\boldsymbol{A}$ can be multiplied with a vector $\boldsymbol{x} = \begin{bmatrix} 1 \\ 2 \end{bmatrix}$ to get another vector $\boldsymbol{y}$

$$\boldsymbol{y} = \boldsymbol{Ax} = \begin{bmatrix} 1 & 2 \\ -1 & 1 \end{bmatrix} \begin{bmatrix} 1 \\ 2 \end{bmatrix} = \begin{bmatrix} 5 \\ 1 \end{bmatrix} \tag{5.24}$$

To illustrate the meaning of the multiplication $\boldsymbol{y} = \boldsymbol{Ax}$, this operation is plotted in Figure 17. After the multiplication with matrix $\boldsymbol{A}$, the original vector $\boldsymbol{x}$ is transformed to $\boldsymbol{y}$. This transform can be divided into two parts: (1) rotate the vector $\boldsymbol{x}$ by angle θ, and (2) stretch the resulting vector to $\boldsymbol{y}$. Thus, multiplying a matrix by a vector ($\boldsymbol{Ax}$) simply means two “actions”: rotating and stretching this vector.

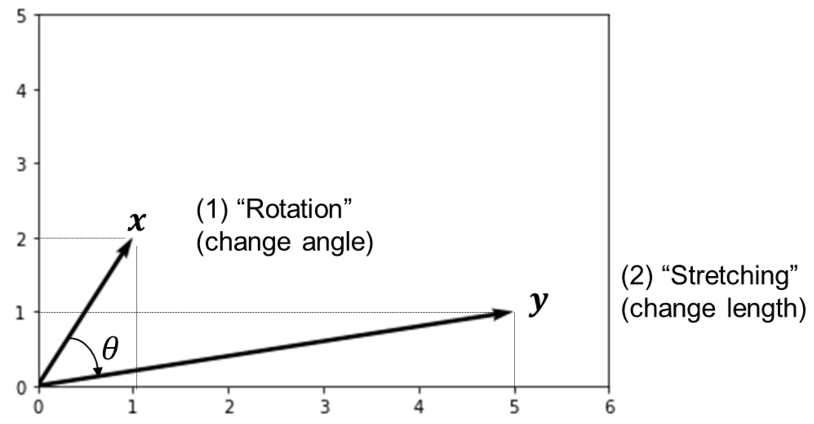


Figure 17 Illustration of a matrix multiplication

Singular value decomposition

The SVD decomposes any real matrix $\boldsymbol{A}$ into three matrices (a complex matrix can also be decomposed by SVD, but in this book, the real matrix is the only focus):

$$\boldsymbol{A} = \boldsymbol{U\Sigma V^T} \tag{5.25}$$

where $\boldsymbol{U}$ and $\boldsymbol{V}$ are orthogonal matrices, i.e., $\boldsymbol{U}^T\boldsymbol{U} = \boldsymbol{I}$ and $\boldsymbol{V}^T\boldsymbol{V} = \boldsymbol{VV}^T = \boldsymbol{I}$, and $\boldsymbol{\Sigma}$ is a diagonal matrix including zero off-diagonals. The superscript T denotes the transpose of the matrix. For example, consider a matrix $\boldsymbol{A}$ with assigned values:

$$\boldsymbol{A} = \begin{bmatrix} 1 & 2 \\ -1 & 0 \end{bmatrix} \rightarrow \boldsymbol{A}^T = \begin{bmatrix} 1 & -1 \\ 2 & 0 \end{bmatrix} \tag{5.26}$$

The SVD can be very easily implemented using Matlab or Python as shown in the below code section:

```
Matlab code for SVD:
[U, S, V]=SVD(A);


Python code for SVD:
U, S, VT=svd(A,full_matrices=False);
```

To explain the matrices in the SVD, decompose a 2 × 2 matrix:

$$\boldsymbol{A} = \begin{bmatrix} 1 & 2 \\ -1 & 1 \end{bmatrix} \tag{5.27}$$

Using SVD, matrix $\boldsymbol{A}$ can be decomposed into three matrices:

$$\boldsymbol{A} = \begin{bmatrix} 1 & 2 \\ -1 & 1 \end{bmatrix} = \begin{bmatrix} 0.957 & 0.29 \\ 0.29 & -0.957 \end{bmatrix} \begin{bmatrix} 2.3 & 0 \\ 0 & 1.3 \end{bmatrix} \begin{bmatrix} 0.29 & 0.957 \\ 0.957 & -0.29 \end{bmatrix} \tag{5.28}$$

To illustrate the meaning of the matrices $\boldsymbol{U}, \boldsymbol{\Sigma}$ and $\boldsymbol{V}$, a vector $\boldsymbol{x}$ is used to multiply with the matrix $\boldsymbol{A}$**,** which is equivalent to multiplying with the three decomposed matrices. This process can be visualized in Figure 18. Through the SVD original operation $\mathbf{Ax}$ can be decomposed into three separated operations: (1) $\mathbf{V^T x}$, (2) $\mathbf{\Sigma V^T x}$, and (3) $\mathbf{U\Sigma V^T x}$ that is equal to $\mathbf{Ax}$. In Figure 18, there is a vector $\boldsymbol{x} = [3\ 2]^T$. The operation (1) rotates the vector $\boldsymbol{x}$ to $\mathbf{V^T x} = [2.78\ 2.29]^T$. It is a rotation because it only changes the direction of the vector and the module of the two vectors remain identical, i.e., $||\boldsymbol{x}|| = ||\mathbf{V^T x}||$. Multiplying the matrix $\mathbf{\Sigma}$ stretches the vector $\mathbf{V^T x}$ to $\mathbf{\Sigma V^T x} = [6.41\ 2.99]^T$as (2) in Figure 18. It is an anisotropic stretching in x and y directions. The stretching factor in the x direction and y direction is equal to the first and second diagonal in matrix $\mathbf{\Sigma}$, respectively. The operation (3) is another rotation from vector $\mathbf{\Sigma V^T x}$ to the final vector $\mathbf{U\Sigma V^T x} = \mathbf{Ax} = [7\ -1]^T$. Demonstrating by this example, the SVD decomposes the original matrix into three matrices: a "rotation" matrix $\boldsymbol{U}$, a "stretching" matrix $\mathbf{\Sigma}$, and another "rotation" matrix $\mathbf{V^T}$. The diagonal components in matrix $\mathbf{\Sigma}$ are called singular values representing the stretching factors at different coordinates. The singular values are typically ordered from the largest to the smallest. Vertical vectors in $\boldsymbol{U}$ and $\boldsymbol{V}$ are called singular vectors.

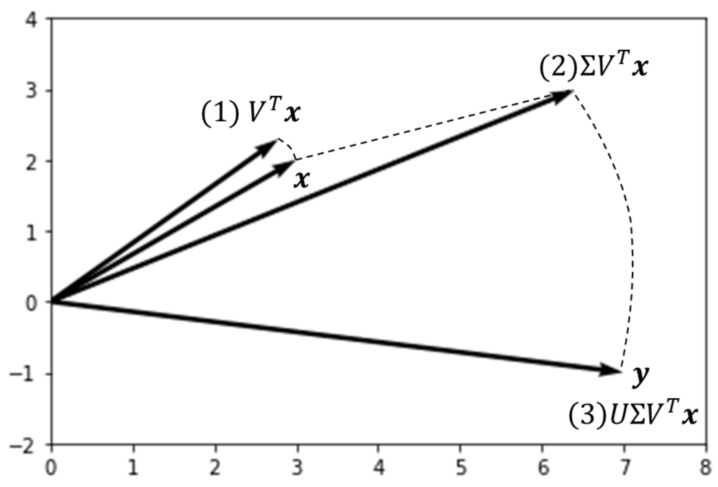


Figure 18 Visualization of $\boldsymbol{Ax}=\boldsymbol{U\Sigma V^T x}$ *(A video is available in the E-book)*

Matrix order reduction by SVD truncation

The singular values in matrix $\boldsymbol{\Sigma}$ are defined as $\sigma_1, \sigma_2, \ldots, \sigma_m$. The rows and columns corresponding to specific singular value(s) in $\boldsymbol{U}$, $\boldsymbol{\Sigma}$, and $\boldsymbol{V^T}$ can be truncated to reduce the order of the original matrix. These truncated matrices give an approximation of the original $\boldsymbol{A}$ matrix. A simple example is used to demonstrate the SVD truncation. Consider a three-dimensional matrix $\boldsymbol{A}$:

$$\boldsymbol{A} = \begin{bmatrix} 1 & 2 & 3 \\ 6 & 5 & 4 \\ 8 & 7 & 9 \end{bmatrix} \tag{5.29}$$

The matrix $\boldsymbol{A}$ is also equal to a product of three matrices based on SVD:

$$\boldsymbol{A} = \boldsymbol{U\Sigma V^T} = \begin{bmatrix} -0.20 & 0.61 & -0.75 \\ -0.51 & -0.72 & -0.45 \\ -0.83 & 0.29 & 0.47 \end{bmatrix} \begin{bmatrix} 16.73 & 0 & 0 \\ 0 & 2.14 & 0 \\ 0 & 0 & 0.58 \end{bmatrix} \begin{bmatrix} -0.59 & -0.52 & -0.60 \\ -0.63 & -0.15 & 0.75 \\ 0.48 & -0.83 & 0.24 \end{bmatrix} \tag{5.30}$$

Since the third singular value (0.58) is smaller than the other two (16.73 and 2.14), a truncated SVD can be built by truncating the third singular value in matrix $\boldsymbol{\Sigma}$ and corresponding column in matrix $\boldsymbol{U}$ (third column) and corresponding row in matrix $\boldsymbol{V^T}$ (third row). It is called the first order reduction of the matrix $\boldsymbol{A}$:

$$\begin{aligned} \boldsymbol{A} \approx \boldsymbol{A'} &= \begin{bmatrix} -0.207 & 0.618 \\ -0.516 & -0.727 \\ -0.831 & 0.297 \end{bmatrix} \begin{bmatrix} 16.734 & 0 \\ 0 & 2.145 \end{bmatrix} \begin{bmatrix} -0.595 & -0.527 & -0.608 \\ -0.639 & -0.150 & 0.755 \end{bmatrix} \\ &= \begin{bmatrix} 1.21 & 1.63 & 3.11 \\ 6.13 & 4.78 & 4.01 \\ 7.87 & 7.23 & 8.93 \end{bmatrix} \end{aligned} \tag{5.31}$$

The first order reduction $\boldsymbol{A'}$ is an approximate of the original $\boldsymbol{A}$. A further order reduction can be conducted by truncating the second singular value in matrix $\boldsymbol{\Sigma}$ and corresponding column in matrix $\boldsymbol{U}$ and corresponding row in matrix $\boldsymbol{V^T}$:

$$\boldsymbol{A} \approx \boldsymbol{A}'' = \begin{bmatrix} -0.207 \\ -0.516 \\ -0.831 \end{bmatrix} [16.734][-0.595 \quad -0.527 \quad -0.608] = \begin{bmatrix} 2.06 & 1.83 & 2.11 \\ 5.13 & 4.54 & 5.24 \\ 8.27 & 7.33 & 8.45 \end{bmatrix} \tag{5.32}$$

In this example, the matrix $\boldsymbol{A}$'s can be represented by a product of three vectors, with little loss in R^2 [25](see Table 3). In fact, after two reductions, the R^2 score is still above 0.9. The R^2 score is a metric to quantify the similarity of two datasets (two matrices in this case). If $R^2 = 1$, it indicates the two matrices are identical. If $R^2 = 0$, it indicates that the two matrices are quite different. Mean Square Error (MSE) is defined as

$$\mathrm{MSE} = \frac{1}{n}\sum_{i=1}^{n}(A_i - A_i')^2 \tag{5.33}$$

where A_i is the i^{th} component in matrix $\boldsymbol{A}$ and n is the number of components.

Table 3 MSE and R^2 of Reduced $\boldsymbol{A}$ Matrices.

Matrix		**Mean Square Error (MSE) due to order reduction**	$\boldsymbol{R^2}$
Original $\boldsymbol{A}$	$\boldsymbol{A} = \begin{bmatrix} 1 & 2 & 3 \\ 6 & 5 & 4 \\ 8 & 7 & 9 \end{bmatrix}$	0	1
1st order reduction	$\boldsymbol{A} \approx \boldsymbol{A}' = \begin{bmatrix} 1.21 & 1.63 & 3.11 \\ 6.13 & 4.78 & 4.01 \\ 7.87 & 7.23 & 8.93 \end{bmatrix}$	0.038	0.992
2nd order reduction	$\boldsymbol{A} \approx \boldsymbol{A}'' = \begin{bmatrix} 2.06 & 1.83 & 2.11 \\ 5.13 & 4.54 & 5.24 \\ 8.27 & 7.33 & 8.45 \end{bmatrix}$	0.549	0.923

Example: spring-mass harmonic oscillator

A spring-mass system shown in Figure 19 is ideally a one-dimensional system. For an ideal system, when the mass is released a small distance away from equilibrium (i.e., the spring is stretched), the mass will oscillate along the length of the spring indefinitely at a set frequency. However, the reality is that the measured results are not one dimensional because the mass swings back and forth, and the camera recording the motion is not held perfectly still.

[25] https://en.wikipedia.org/wiki/Coefficient_of_determination

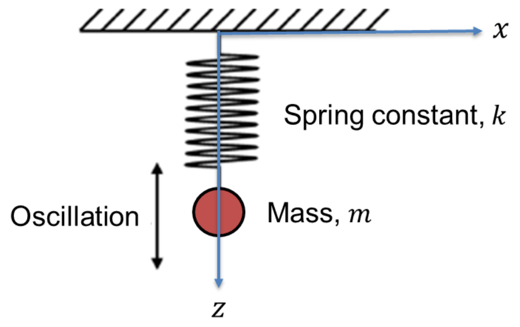


Figure 19 A schematic of ideal spring-mass system

The actual three-dimensional motion is recorded using three cameras (see Figure 20). This 3D motion capture enables more accurate identification of system properties such as spring constant or damping factor. Moreover, it benefits the analysis of system uncertainties and sensitivities. The position of the ball tracked by each camera is depicted in each panel. The local coordinate system can be defined for each phone and record the projected displacement of the spring with respect to each coordinate spring (see Figure 21). The time-dependent displacement of the spring from each camera video is extracted using an opensource software: Tracker[26]. The software interface is shown in Figure 22. The goal is to get one-dimensional motion data from the two-dimensional projection data collected from three different angles (different cameras). This new basis will filter out the noise and reveal hidden structure (i.e., determine the unit basis vector along the z-axis). After the important dimension (axis) is identified, it is possible to estimate the key properties of the system, such as damping coefficient and effect of mass, from the collected noisy data.

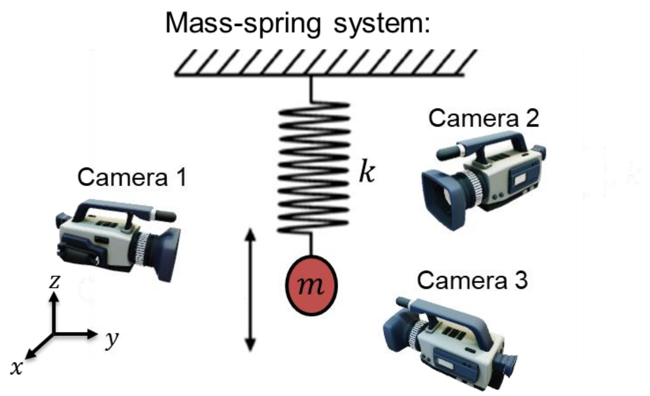


Figure 20 A spring-mass motion example. The position of a ball attached to a spring is recorded using three cameras 1, 2 and 3. The projected position of the ball tracked by each camera is depicted in each panel.

[26] https://physlets.org/tracker/

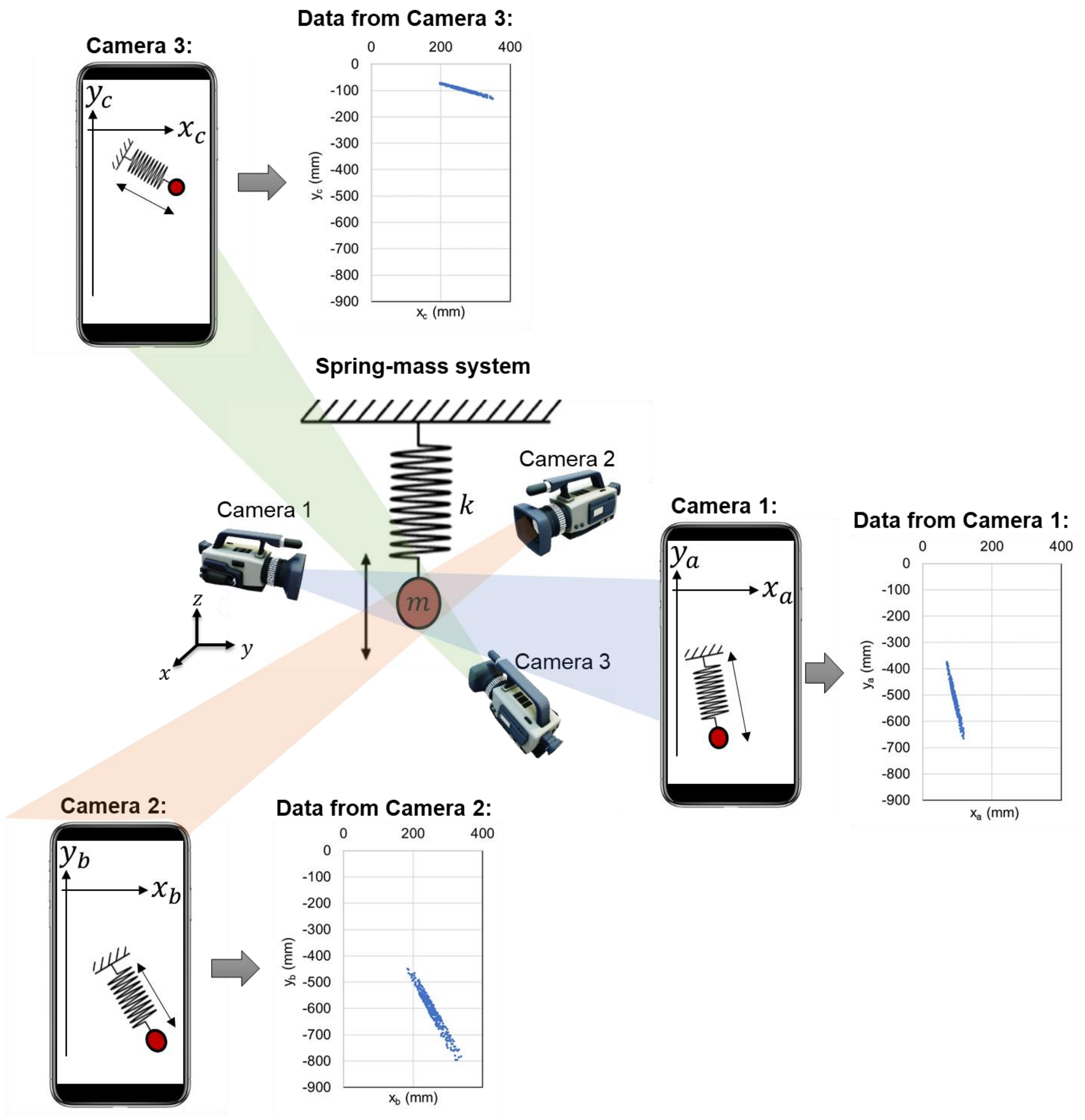
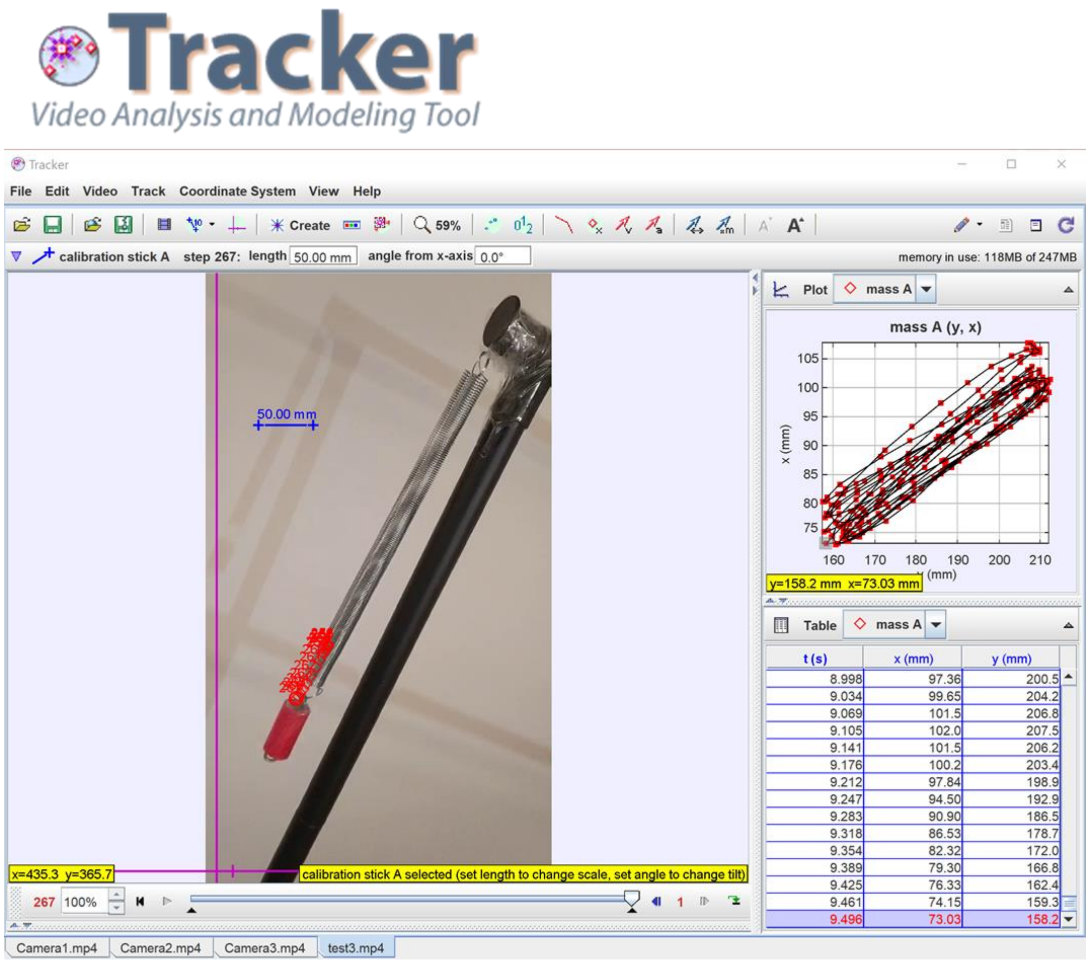


Figure 21 Projection data from three different cameras. x and y are local coordinates. Shaded regions in different colors indicate the projection paths (A video is available in the E-book).

All the local projection data can be put into a single data matrix. The data matrix is six dimensional:

$$\boldsymbol{X} = [\boldsymbol{x_a}\ \ \boldsymbol{y_a}\ \ \boldsymbol{x_b}\ \ \boldsymbol{y_b}\ \ \boldsymbol{x_c}\ \ \boldsymbol{y_c}] \tag{5.34}$$

where $\boldsymbol{x_a}$ and $\boldsymbol{y_a}$ are local displacement vectors (each vector includes positions at different time) from the first camera, $\boldsymbol{x_b}$ and $\boldsymbol{y_b}$ are local displacement vectors from the second camera, and $\boldsymbol{x_c}$ and $\boldsymbol{y_c}$ are local displacement vectors from the third camera. The vectors are vertical. This system can be ideally described by a single direction (i.e., one-dimensional system). Therefore, there should be some redundancy between the six measurements. The SVD and PCA will be used to reduce the redundancy and recover the dominant one-dimensional data.


Tracker
Video Analysis and Modeling Tool
File Edit Video Track Coordinate System View Help
Create
59%
calibration stick A step 267: length 50.00 mm angle from x-axis 0.0°
memory in use: 118MB of 247MB
50.00 mm
Plot
mass A
mass A (y, x)
x (mm)
y (mm)
y=158.2 mm x=73.03 mm
x=435.3 y=365.7
calibration stick A selected (set length to change scale, set angle to change tilt)
267
100%
Camera1.mp4
Camera2.mp4
Camera3.mp4
test3.mp4

t (s)	x (mm)	y (mm)
8.998	97.36	200.5
9.034	99.65	204.2
9.069	101.5	206.8
9.105	102.0	207.5
9.141	101.5	206.2
9.176	100.2	203.4
9.212	97.84	198.9
9.247	94.50	192.9
9.283	90.90	186.5
9.318	86.53	178.7
9.354	82.32	172.0
9.389	79.30	166.8
9.425	76.33	162.4
9.461	74.15	159.3
9.496	73.03	158.2

Figure 22 Using an opensource software, Tracker[27]*, to extract local displacement of the mass from a filmed video. (A video is available in the E-book)*

5.3.3. Principal component analysis [Advanced topic]

Principal component analysis (PCA) pre-processes the data by mean subtraction before performing the SVD. To apply PCA to reduce redundancy between data coordinates, a covariance matrix is required to be built from the data matrix $\boldsymbol{X}$.

Variance and Covariance

The concept of variance and covariance must be introduced before it can be applied. Variance describes how much a variable varies with respect to its mean. Consider two vectors, $\boldsymbol{x}$ and $\boldsymbol{y}$ (each vector's mean is assumed to be zero),

$$\boldsymbol{x} = [x_1, x_2, \dots, x_n] \tag{5.35}$$

[27] https://physlets.org/tracker/

$$\boldsymbol{y} = [y_1, y_2, \dots, y_n] \tag{5.36}$$

The variance of $\boldsymbol{x}$ and $\boldsymbol{y}$ can be defined as

$$\sigma_x^2 = \frac{1}{n-1}(\boldsymbol{x}\boldsymbol{x}^T) = \frac{1}{n-1}(x_1^2 + x_2^2 + \cdots + x_n^2) \tag{5.37}$$

$$\sigma_y^2 = \frac{1}{n-1}(\boldsymbol{y}\boldsymbol{y}^T) = \frac{1}{n-1}(y_1^2 + y_2^2 + \cdots + y_n^2) \tag{5.38}$$

Variance is proportional to the square of the magnitude of a (zero-mean) vector.

Covariance is expressed very similarly to variance, which is proportional to the inner product of two vectors:

$$\sigma_{xy}^2 = \frac{1}{n-1}(\boldsymbol{x}\boldsymbol{y}^T) = \frac{1}{n-1}(x_1y_1 + x_2y_2 + \cdots + x_ny_n) \tag{5.39}$$

Therefore, the covariance is positive and large if the two vectors point to the same direction. Otherwise, the covariance is small if the two vectors are perpendicular.

Consider the matrix of data described previously that has 6 features (each vector's mean is assumed to be zero):

$$\boldsymbol{X} = [\boldsymbol{x_a} \;\; \boldsymbol{y_a} \;\; \boldsymbol{x_b} \;\; \boldsymbol{y_b} \;\; \boldsymbol{x_c} \;\; \boldsymbol{y_c}] \tag{5.40}$$

A covariance matrix $\boldsymbol{C_X}$ can be built to find how each column varies with others (if the covariance between the rows of the matrix is of interest, the definition of covariance matrix is $\boldsymbol{C_X} = \frac{1}{n-1}\boldsymbol{X}\boldsymbol{X}^T$):

$$\boldsymbol{C_X} = \frac{1}{n-1}\boldsymbol{X}^T\boldsymbol{X} = \begin{bmatrix} \sigma_{x_ax_a}^2 & \sigma_{x_ay_a}^2 & \sigma_{x_ax_b}^2 & \sigma_{x_ay_b}^2 & \sigma_{x_ax_c}^2 & \sigma_{x_ay_c}^2 \\ \sigma_{y_ax_a}^2 & \sigma_{y_ay_a}^2 & \sigma_{y_ax_b}^2 & \sigma_{y_ay_b}^2 & \sigma_{y_ax_c}^2 & \sigma_{y_ay_c}^2 \\ \sigma_{x_bx_a}^2 & \sigma_{x_by_a}^2 & \sigma_{x_bx_b}^2 & \sigma_{x_by_b}^2 & \sigma_{x_bx_c}^2 & \sigma_{x_by_c}^2 \\ \sigma_{y_bx_a}^2 & \sigma_{y_by_a}^2 & \sigma_{y_bx_b}^2 & \sigma_{y_by_b}^2 & \sigma_{y_bx_c}^2 & \sigma_{y_by_c}^2 \\ \sigma_{x_cx_a}^2 & \sigma_{x_cy_a}^2 & \sigma_{x_cx_b}^2 & \sigma_{x_cy_b}^2 & \sigma_{x_cx_c}^2 & \sigma_{x_cy_c}^2 \\ \sigma_{y_cx_a}^2 & \sigma_{y_cy_a}^2 & \sigma_{y_cx_b}^2 & \sigma_{y_cy_b}^2 & \sigma_{y_cx_c}^2 & \sigma_{y_cy_c}^2 \end{bmatrix} \tag{5.41}$$

It is noted that values along the diagonal are the variance of the features (coordinates). The off-diagonals are the covariance between pairs of features. The covariance matrix is symmetric.

The goal of PCA is to get a diagonal covariance matrix because this means minimizing the covariance between different features, i.e., reducing the redundancy between different features. Specifically, the goal is to find a matrix $\boldsymbol{P}$ to transform the data matrix $\boldsymbol{X}$ to $\boldsymbol{XP}$ so that the covariance matrix of $\boldsymbol{XP}$, i.e., $\boldsymbol{C}_{XP}$, is diagonal.

Identifying intrinsic dimension of spring-mass system using PCA/SVD

Referring to the previous example, to apply PCA for dimension reduction the data matrix has to be centralized by subtracting the mean of each vector:

$$\boldsymbol{B} = \boldsymbol{X} - \begin{bmatrix} 1 \\ \dots \\ 1 \end{bmatrix} \overline{\boldsymbol{x}}^T \tag{5.42}$$

where $\boldsymbol{X}$ is the data matrix and each component of the $\overline{\boldsymbol{x}}$ is the mean of each data feature. For example, if $\boldsymbol{X} = [\boldsymbol{x_a}\ \ \boldsymbol{y_a}]$ $\boldsymbol{X}$ and $\boldsymbol{B}$ are plotted in Figure 23.

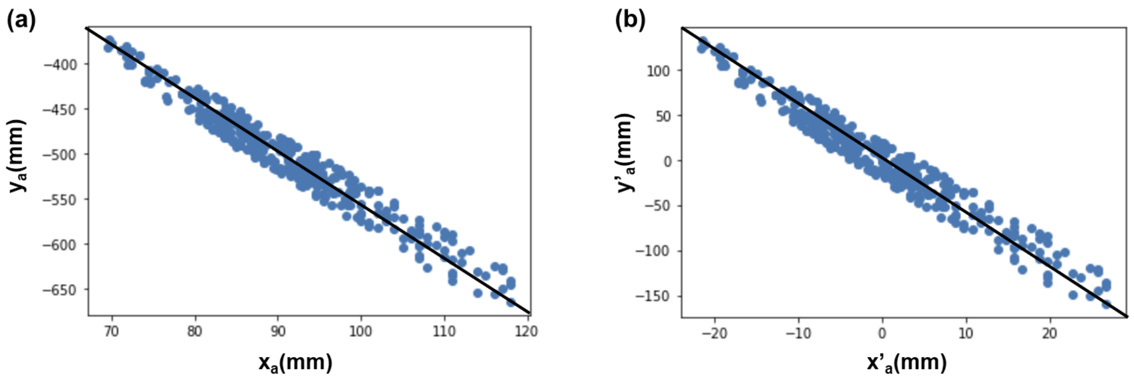


Figure 23 Plots of (a) 2D data matrix and (b) mean-subtracted data.

The principal components can be obtained by solving the eigenvectors and eigenvalues problem (eigenvectors are indeed the principal components, see Appendix A). However, this solution is sometimes computationally expensive. A more efficient way is to solve the SVD for the mean-subtracted data matrix directly and obtain the principal components since SVD and PCA are mathematically consistent (Appendix B). Thus, the mean-subtracted data matrix $\boldsymbol{B}$ is divided into three matrices by SVD:

$$\boldsymbol{B} = \boldsymbol{U\Sigma V^T} \tag{5.43}$$

The $\boldsymbol{V}$ is the matrix that can transform the data matrix to a new matrix with diagonal covariance matrix, i.e., $\boldsymbol{C}_{BV}$ is a diagonal matrix. This means the column vectors of the matrix $\boldsymbol{BV}$ is independent (the proof is provided in Appendix B). For this case, the covariance matrix of the matrix $\boldsymbol{BV}$ is

$$\boldsymbol{C_{BV}} = \begin{bmatrix} 10294 & 0 & 0 & 0 & 0 & 0 \\ 0 & 188 & 0 & 0 & 0 & 0 \\ 0 & 0 & 55 & 0 & 0 & 0 \\ 0 & 0 & 0 & 4.3 & 0 & 0 \\ 0 & 0 & 0 & 0 & 4.2 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1.3 \end{bmatrix} \tag{5.44}$$

Python code for computing covariance matrix:

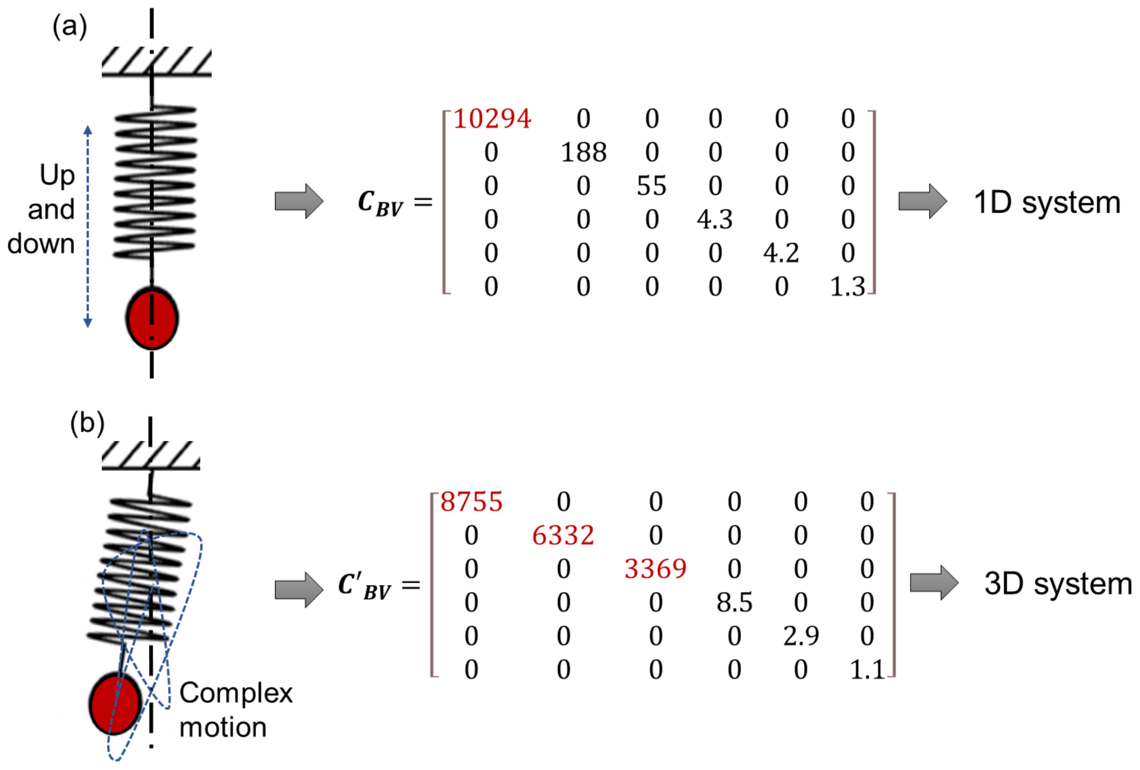
```
U, S, VT=svd(B,full_matrices=False)
Y=B.dot(VT.T)
np.cov(Y.T)
```

Based on the diagonals in the covariance matrix, it is reasonable to conclude that the spring-mass system of interest is intrinsically one dimensional because the first diagonal is an order of magnitude larger than the others. As a comparison, the covariance matrix of the data matrix $\boldsymbol{B}$ is also shown as

$$\boldsymbol{C_B} = \begin{bmatrix} 110 & -607 & -304 & -723 & 323 & -118 \\ -607 & 3510 & -1678 & 3983 & -1786 & 649 \\ -304 & -1678 & 923 & -2071 & 890 & -325 \\ -723 & 3983 & -2071 & 4927 & -2114 & 771 \\ 323 & -1786 & -890 & -2114 & 979 & -358 \\ -118 & 649 & -325 & 771 & -358 & 136 \end{bmatrix} \tag{5.45}$$

It can be seen that the none of the values are zero or mush smaller than others, which means the column vectors of the data matrix are all dependent.

The same procedure can be conducted to identify the intrinsic dimension of the spring-mass system. In practice, the intrinsic dimension of the spring-mass system highly depends on the initial condition of the ball. As shown in Figure 24(a) if the ball is released very closed to the axis of symmetry (dot dash line in the figure). The motion of the ball is up and down along the symmetry. In this case one coordinate is good enough to describe the ball's motion, i.e., the intrinsic dimension is 1D. Correspondingly, the data matrix (captured by three cameras) can be transformed so that the first diagonal of the covariance matrix is at least one order of magnitude larger than the others. However, if the ball is released a distance away from the axis of symmetry. The motion of the ball becomes complicate, and it is a three-dimensional motion. In this case, the covariance matrix of the transformed data matrix is shown in the Figure 24(b). The first three diagonals are at least one order of magnitude larger than the other diagonals indicating that the intrinsic dimension of the system of interest is three.



$$\boldsymbol{C_{BV}} = \begin{bmatrix} 10294 & 0 & 0 & 0 & 0 & 0 \\ 0 & 188 & 0 & 0 & 0 & 0 \\ 0 & 0 & 55 & 0 & 0 & 0 \\ 0 & 0 & 0 & 4.3 & 0 & 0 \\ 0 & 0 & 0 & 0 & 4.2 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1.3 \end{bmatrix}$$

$$\boldsymbol{C'_{BV}} = \begin{bmatrix} 8755 & 0 & 0 & 0 & 0 & 0 \\ 0 & 6332 & 0 & 0 & 0 & 0 \\ 0 & 0 & 3369 & 0 & 0 & 0 \\ 0 & 0 & 0 & 8.5 & 0 & 0 \\ 0 & 0 & 0 & 0 & 2.9 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1.1 \end{bmatrix}$$

Figure 24 Intrinsic dimension identified by covariance matrix and SVD/PCA

5.3.4. Proper generalized decomposition (PGD) [Advanced topic]

From SVD to PGD

Proper generalized decomposition (PGD) is an alternative method to SVD of finding the principal components (or called modes) and reducing the dimensionality of the dataset. To introduce the mathematical concept of the PGD, the SVD is recalled first. Consider the matrix form of SVD for a three-by-three matrix $\boldsymbol{A}$:

$$\boldsymbol{A} = \boldsymbol{U\Sigma V^T} = [\boldsymbol{u_1} \quad \boldsymbol{u_2} \quad \boldsymbol{u_3}] \begin{bmatrix} \sigma_1 & 0 & 0 \\ 0 & \sigma_2 & 0 \\ 0 & 0 & \sigma_3 \end{bmatrix} \begin{bmatrix} \boldsymbol{v_1}^T \\ \boldsymbol{v_2}^T \\ \boldsymbol{v_3}^T \end{bmatrix} \tag{5.46}$$

where $\boldsymbol{u_1}$, $\boldsymbol{u_2}$ and $\boldsymbol{u_3}$ are three column vectors of matrix $\boldsymbol{U}$, $\boldsymbol{v_1}$, $\boldsymbol{v_2}$ and $\boldsymbol{v_3}$ are three column vectors of matrix $\boldsymbol{V}$, and σ_1, σ_2, and σ_3 are singular values. The vector form of SVD can be derived from this matrix form as

$$\boldsymbol{A} = [\boldsymbol{u_1} \quad \boldsymbol{u_2} \quad \boldsymbol{u_3}] \begin{bmatrix} \sigma_1 & 0 & 0 \\ 0 & \sigma_2 & 0 \\ 0 & 0 & \sigma_3 \end{bmatrix} \begin{bmatrix} \boldsymbol{v_1}^T \\ \boldsymbol{v_2}^T \\ \boldsymbol{v_3}^T \end{bmatrix} = \boldsymbol{u_1}\sigma_1\boldsymbol{v_1}^T + \boldsymbol{u_2}\sigma_2\boldsymbol{v_2}^T + \boldsymbol{u_3}\sigma_3\boldsymbol{v_3}^T \tag{5.47}$$

where $\boldsymbol{u_1}$, $\boldsymbol{u_2}$ and $\boldsymbol{u_3}$ are orthonormal vectors and $\boldsymbol{v_1}$, $\boldsymbol{v_2}$ and $\boldsymbol{v_3}$ are principal components (they are also orthonormal vectors). The PGD follows the similar form as SVD but does not constrain the principal components to be orthonormal. For example, the three-by-three matrix $\boldsymbol{A}$ can be decomposed by PGD as

$$\boldsymbol{A} = \boldsymbol{f_1 g_1}^T + \boldsymbol{f_2 g_2}^T + \boldsymbol{f_3 g_3}^T + \cdots + \boldsymbol{f_n g_n}^T = [\boldsymbol{f_1} \quad \boldsymbol{f_2} \quad \dots \quad \boldsymbol{f_n}] \begin{bmatrix} \boldsymbol{g_1}^T \\ \boldsymbol{g_2}^T \\ \dots \\ \boldsymbol{g_n}^T \end{bmatrix} \tag{5.48}$$

where the vectors $\boldsymbol{f_1}$, $\boldsymbol{f_2}$,… $\boldsymbol{f_n}$, and $\boldsymbol{g_1}$, $\boldsymbol{g_2}$, … $\boldsymbol{g_n}$ are not necessarily orthonormal. That is where the name "generalized" comes from. It is noted that PGD is an iterative algorithm rather than an analytical solution like SVD, which means the first term $\boldsymbol{f_1}$ and $\boldsymbol{g_1}$ are computed first, and then $\boldsymbol{f_2}$ and $\boldsymbol{g_2}$, and so on. There are two types of PGD strategies: incremental approach[28] and modal superposition[29].

A matrix decomposition example using incremental PGD

Consider a two-by-two matrix

$$\boldsymbol{A} = \begin{bmatrix} 1 & 2 \\ -1 & 1 \end{bmatrix} \tag{5.49}$$

[28] Modesto, D., Zlotnik, S., & Huerta, A. (2015), Proper generalized decomposition for parameterized Helmholtz problems in heterogeneous and unbounded domains: application to harbor agitation. Computer Methods in Applied Mechanics and Engineering.

[29] Bro, R., 1997. PARAFAC. Tutorial and applications. Chemometrics and intelligent laboratory systems, 38(2), pp.149-171.

A n-order PGD approximates the original matrix by decomposing it into the form:

$$\boldsymbol{A} = \boldsymbol{f_1}\boldsymbol{g_1}^T + \boldsymbol{f_2}\boldsymbol{g_2}^T + \boldsymbol{f_3}\boldsymbol{g_3}^T + \cdots + \boldsymbol{f_n}\boldsymbol{g_n}^T \tag{5.50}$$

Step (1) Rewrite $\boldsymbol{A^n}$ in a recursive form:

$$\boldsymbol{A}^n = \boldsymbol{A}^{n-1} + \boldsymbol{f_n}\boldsymbol{g_n}^T \tag{5.51}$$

$$\boldsymbol{A}^{n-1} = \boldsymbol{f_1}\boldsymbol{g_1}^T + \boldsymbol{f_2}\boldsymbol{g_2}^T + \cdots + \boldsymbol{f_{n-1}}\boldsymbol{g_{n-1}}^T \tag{5.52}$$

Step (2) The initial recursive term is assumed to be zero:

$$\boldsymbol{A}^0 = \begin{bmatrix} 0 & 0 \\ 0 & 0 \end{bmatrix} \tag{5.53}$$

Step (3) The residual $\boldsymbol{R_0}$ is defined as the difference of $\boldsymbol{A}$ and $\boldsymbol{A^{n-1}}$. For example,

$$\boldsymbol{R_0} = \boldsymbol{A} - \boldsymbol{A}^0 = \begin{bmatrix} 1 & 2 \\ -1 & 1 \end{bmatrix} - \begin{bmatrix} 0 & 0 \\ 0 & 0 \end{bmatrix} = \begin{bmatrix} 1 & 2 \\ -1 & 1 \end{bmatrix} \tag{5.54}$$

Step (4) The first components $\boldsymbol{f_1}$ and $\boldsymbol{g_1}$ are computed by

$$\boldsymbol{f_1} = \boldsymbol{R_0}\boldsymbol{g_1}(\boldsymbol{g_1}^T\boldsymbol{g_1})^{-1} \tag{5.55}$$

$$\boldsymbol{g_1} = \boldsymbol{R_0^T}\boldsymbol{f_1}\left(\boldsymbol{f_1}^T\boldsymbol{f_1}\right)^{-1} \tag{5.56}$$

A random initial of $\boldsymbol{g_1}$ is set and then the above two equations are used to compute $\boldsymbol{f_1}$ and updated $\boldsymbol{g_1}$.

For example, if the random initial of $\boldsymbol{g_1} = [1 \quad 1]^T$ (this step should be converged to the same result no matter what the initial value is assigned.), the $\boldsymbol{f_1}$ and updated $\boldsymbol{g_1}$ can be computed as

$$\boldsymbol{f_1} = \begin{bmatrix} 1 & 2 \\ -1 & 1 \end{bmatrix} \begin{bmatrix} 1 \\ 1 \end{bmatrix} \left([1\ 1] \begin{bmatrix} 1 \\ 1 \end{bmatrix}\right)^{-1} = \begin{bmatrix} 1.5 \\ 0 \end{bmatrix} \tag{5.57}$$

$$\boldsymbol{g_1} = \begin{bmatrix} 1 & -1 \\ 2 & 1 \end{bmatrix} \begin{bmatrix} 1.5 \\ 0 \end{bmatrix} \left([1.5\ 0] \begin{bmatrix} 1.5 \\ 0 \end{bmatrix}\right)^{-1} = \begin{bmatrix} 0.667 \\ 1.333 \end{bmatrix} \tag{5.58}$$

This process is repeated multiple times until the changes in $\boldsymbol{f_1}$ and $\boldsymbol{g_1}$ are below a certain value. That indicates the step (4) is converged. Two convergence factors, $C_{1(i)}^f$ and $C_{1(i)}^g$, associated with $\boldsymbol{f}$ and $\boldsymbol{g}$ are defined as

$$C_{1(i)}^f = \frac{\text{norm}\left(\boldsymbol{f_{1(i)}} - \boldsymbol{f_{1(i-1)}}\right)}{\text{norm}\left(\boldsymbol{f_{1(i)}}\right)} \tag{5.59}$$

$$C_{1(i)}^g = \frac{\text{norm}\left(\boldsymbol{g_{1(i)}} - \boldsymbol{g_{1(i-1)}}\right)}{\text{norm}\left(\boldsymbol{g_{1(i)}}\right)} \tag{5.60}$$

where i is the iteration number. Table 4 shows the values of $\boldsymbol{f_1}$, $\boldsymbol{g_1}$ and convergence factors at different iterations. The iteration is finalized when the convergence factors are less than a specified value (10^{-6} in this case). Then the converged $\boldsymbol{f_1}$ and $\boldsymbol{g_1}$ can be obtained as

$$\boldsymbol{f_1} = \begin{bmatrix} 1.468 \\ 0.443 \end{bmatrix} \tag{5.61}$$

$$\boldsymbol{g_1} = \begin{bmatrix} 0.436 \\ 1.437 \end{bmatrix} \tag{5.62}$$

Table 4 The values and convergence factors of $\boldsymbol{f_1}$ and $\boldsymbol{g_1}$ for seven iterations.

i	1	2	3	4	5	6	7
$\boldsymbol{f_1}$	$\begin{bmatrix}1.5\\0\end{bmatrix}$	$\begin{bmatrix}1.5\\0.3\end{bmatrix}$	$\begin{bmatrix}1.480\\0.399\end{bmatrix}$	$\begin{bmatrix}1.472\\0.430\end{bmatrix}$	$\begin{bmatrix}1.469\\0.440\end{bmatrix}$	$\begin{bmatrix}1.469\\0.440\end{bmatrix}$	$\begin{bmatrix}1.468\\0.443\end{bmatrix}$
$\boldsymbol{g_1}$	$\begin{bmatrix}0.667\\1.333\end{bmatrix}$	$\begin{bmatrix}0.513\\1.410\end{bmatrix}$	$\begin{bmatrix}0.460\\1.429\end{bmatrix}$	$\begin{bmatrix}0.443\\1.435\end{bmatrix}$	$\begin{bmatrix}0.438\\1.437\end{bmatrix}$	$\begin{bmatrix}0.436\\1.437\end{bmatrix}$	$\begin{bmatrix}0.436\\1.437\end{bmatrix}$
$C^{f}_{1(i)}$		1.020	0.002	0.0002	$2.32e^{-5}$	$2.39e^{-6}$	$2.44e^{-7}$
$C^{g}_{1(i)}$		1.007	0.0007	0.0001	$7.35e^{-6}$	$7.53e^{-7}$	$7.71e^{-8}$

Step (5) Update the approximation with the computed components:

$$\boldsymbol{A}^1 = \boldsymbol{A}^0 + \boldsymbol{f_1}\boldsymbol{g_1}^T = \begin{bmatrix}0 & 0\\0 & 0\end{bmatrix} + \begin{bmatrix}1.468\\0.443\end{bmatrix}[0.436\ 1.437] = \begin{bmatrix}0.639 & 2.110\\0.193 & 0.638\end{bmatrix} \quad (5.63)$$

Step (6) Update residual $\boldsymbol{R_1}$ and check the matrix convergence factor C_1

$$\boldsymbol{R_1} = \boldsymbol{A} - \boldsymbol{A}^1 = \begin{bmatrix}1 & 2\\-1 & 1\end{bmatrix} - \begin{bmatrix}0.639 & 2.110\\0.193 & 0.638\end{bmatrix} = \begin{bmatrix}0.361 & -0.110\\-1.193 & 0.362\end{bmatrix} \quad (5.64)$$

$$C_1 = \frac{\text{norm}(\boldsymbol{R_1})}{\text{norm}(\boldsymbol{A})} = \frac{1.303}{2.303} = 0.566 \quad (5.65)$$

Since the matrix convergence factor (typically 10^{-6} indicates a good convergence) is too large meaning the 1st order approximation is not good enough for this case.

Thus, steps (4)-(6) are repeated to compute the next components $\boldsymbol{f_2}$ and $\boldsymbol{g_2}$ (and associated second order approximation $\boldsymbol{A^2}$). After the steps (4)-(6) are repeated, the values of $\boldsymbol{f_2}$ and $\boldsymbol{g_2}$ are

$$\boldsymbol{f_2} = \begin{bmatrix}0.126\\-0.416\end{bmatrix} \quad (5.66)$$

$$\boldsymbol{g_2} = \begin{bmatrix}2.871\\-0.871\end{bmatrix} \quad (5.67)$$

The second order approximation $\boldsymbol{A^2}$ can be computed by

$$\boldsymbol{A}^2 = \boldsymbol{A}^1 + \boldsymbol{f_2}\boldsymbol{g_2}^T = \begin{bmatrix}0.639 & 2.110\\0.193 & 0.638\end{bmatrix} + \begin{bmatrix}0.126\\-0.416\end{bmatrix}[2.871\ \ -0.871] = \begin{bmatrix}1.000 & 2.000\\-1.000 & 1.000\end{bmatrix} \quad (5.68)$$

This result is very accurate as compared with the original matrix $\boldsymbol{A}$, *and the* matrix convergence factor $C_2 = 2.16 \times 10^{-16}$, which means two modes are good enough for achieving a convergence (indicated by a small enough convergence factor).

This matrix deposition problem can be done by SVD. To compare the results from SVD and PGD, the SVD result of the same matrix $\boldsymbol{A} = \begin{bmatrix}1 & 2\\-1 & 1\end{bmatrix}$ can be expressed as

$$\boldsymbol{A}_{SVD} = [\boldsymbol{u_1} \quad \boldsymbol{u_2}]\begin{bmatrix}\sigma_1 & 0\\0 & \sigma_2\end{bmatrix}\begin{bmatrix}\boldsymbol{v_1}^T\\\boldsymbol{v_2}^T\end{bmatrix} = \begin{bmatrix}0.957 & 0.290\\0.290 & -0.957\end{bmatrix}\begin{bmatrix}2.30 & 0\\0 & 1.30\end{bmatrix}\begin{bmatrix}0.290 & 0.957\\0.957 & -0.290\end{bmatrix} \quad (5.69)$$

PGD result of the matrix $\boldsymbol{A}$ can be expressed as

$$\begin{aligned}\boldsymbol{A}_{PGD} &= \begin{bmatrix}1.468\\0.443\end{bmatrix}[0.436\ 1.437] + \begin{bmatrix}0.126\\-0.416\end{bmatrix}[2.871\ \ -0.871]\\ &= \begin{bmatrix}1.468 & 0.126\\0.443 & -0.416\end{bmatrix}\begin{bmatrix}0.436 & 1.437\\2.871 & -0.871\end{bmatrix}\end{aligned} \tag{5.70}$$

It is noted that the vectors in PGD matrices are not normal. They can be normalized so that the PGD result can be compared with SVD result

$$\begin{aligned}\boldsymbol{A}_{PGD} &= \left[1.53*\begin{bmatrix}0.957\\0.290\end{bmatrix}\quad 0.43*\begin{bmatrix}0.290\\-0.957\end{bmatrix}\right]\begin{bmatrix}1.5*[0.290\quad 0.957]\\3.0*[0.957\quad -0.290]\end{bmatrix}\\ &= [1.53\boldsymbol{u_1}\quad 0.43\boldsymbol{u_2}]\begin{bmatrix}1.5\boldsymbol{v_1}^T\\3.0\boldsymbol{v_2}^T\end{bmatrix} = [\boldsymbol{u_1}\quad \boldsymbol{u_2}]\begin{bmatrix}2.30 & 0\\0 & 1.30\end{bmatrix}\begin{bmatrix}\boldsymbol{v_1}^T\\\boldsymbol{v_2}^T\end{bmatrix}\end{aligned} \tag{5.71}$$

Thus, PGD can find the same principal components as SVD for the matrix decomposition problem. In practice, the order of matrix can be higher, e.g., a n-by-n matrix.

A PGD example using modal superposition

In the previous example using incremental PGD, the first mode (i.e., $\boldsymbol{f_1 g_1}^T$) is completed solved first, and then the next components $\boldsymbol{f_2}$ and $\boldsymbol{g_2}$ are computed based on the known first mode. In contrast, the modal superposition PGD updates all the modes back and forth until all the modes are unchanged. In general, the modal superposition is expected to obtain the optimal decomposition, which includes a smaller number of modes than incremental PGD. But the modal superposition is typically more computationally expensive for high-dimensional problems and sometimes encounters convergence difficulties because of its global iterative feature. Consider the same two-by-two matrix

$$\boldsymbol{A} = \begin{bmatrix}1 & 2\\-1 & 1\end{bmatrix} \tag{5.72}$$

In modal superposition PGD, the number of modes is predetermined (two modes in this case):

$$\boldsymbol{A} = \boldsymbol{f_1 g_1}^T + \boldsymbol{f_2 g_2}^T \tag{5.73}$$

Step (1) The initial $\boldsymbol{f_2}$ and $\boldsymbol{g_2}$ are assumed to be

$$\boldsymbol{f_2} = \begin{bmatrix}1\\1\end{bmatrix} \tag{5.74}$$

$$\boldsymbol{g_2} = \begin{bmatrix}1\\1\end{bmatrix} \tag{5.75}$$

Step (2) The residual $\boldsymbol{R}$ is defined as the difference of $\boldsymbol{A}$ and $\boldsymbol{f_2 g_2}^T$

$$\boldsymbol{R} = \boldsymbol{A} - \boldsymbol{f_2 g_2}^T = \begin{bmatrix}1 & 2\\-1 & 1\end{bmatrix} - \begin{bmatrix}1 & 1\\1 & 1\end{bmatrix} = \begin{bmatrix}0 & 1\\-2 & 0\end{bmatrix} \tag{5.76}$$

Step (3) The first components $\boldsymbol{f_1}$ and $\boldsymbol{g_1}$ are computed by

$$\boldsymbol{f_1} = \boldsymbol{R g_1}(\boldsymbol{g_1}^T\boldsymbol{g_1})^{-1} \tag{5.77}$$

$$\boldsymbol{g_1} = \boldsymbol{R^T f_1}(\boldsymbol{f_1}^T\boldsymbol{f_1})^{-1} \tag{5.78}$$

A random initial of $\boldsymbol{g_1} = [1 \quad 1]^T$ is set and then the above two equations are used to compute $\boldsymbol{f_1}$ and updated $\boldsymbol{g_1}$. The $\boldsymbol{f_1}$ and updated $\boldsymbol{g_1}$ can be computed as

$$\boldsymbol{f_1} = \begin{bmatrix} 0 & 1 \\ -2 & 0 \end{bmatrix} \begin{bmatrix} 1 \\ 1 \end{bmatrix} \left([1\ 1] \begin{bmatrix} 1 \\ 1 \end{bmatrix}\right)^{-1} = \begin{bmatrix} 0.5 \\ -1 \end{bmatrix} \tag{5.79}$$

$$\boldsymbol{g_1} = \begin{bmatrix} 0 & -2 \\ 1 & 0 \end{bmatrix} \begin{bmatrix} 0.5 \\ -1 \end{bmatrix} \left([0.5\ -1] \begin{bmatrix} 0.5 \\ -1 \end{bmatrix}\right)^{-1} = \begin{bmatrix} 1.6 \\ 0.4 \end{bmatrix} \tag{5.80}$$

Step (4) The residual $\boldsymbol{R}$ is updated as the difference of $\boldsymbol{A}$ and $\boldsymbol{f_1}\boldsymbol{g_1}^T$

$$\boldsymbol{R} = \boldsymbol{A} - \boldsymbol{f_1}\boldsymbol{g_1}^T = \begin{bmatrix} 1 & 2 \\ -1 & 1 \end{bmatrix} - \begin{bmatrix} 0.8 & 0.2 \\ -1.6 & -0.4 \end{bmatrix} = \begin{bmatrix} 0.2 & 1.8 \\ 0.6 & 0.4 \end{bmatrix} \tag{5.81}$$

Step (5) The first components $\boldsymbol{f_2}$ and $\boldsymbol{g_2}$ are computed by

$$\boldsymbol{f_2} = \boldsymbol{R}\boldsymbol{g_2}(\boldsymbol{g_2}^T\boldsymbol{g_2})^{-1} \tag{5.82}$$

$$\boldsymbol{g_2} = \boldsymbol{R^T}\boldsymbol{f_2}\left(\boldsymbol{f_2}^T\boldsymbol{f_2}\right)^{-1} \tag{5.83}$$

Step (6) The matrix convergence factor C_1 is computed to check convergence. The matrix convergence factor C_i at the i^{th} iteration is difeinde as

$$C_i = \frac{\text{norm}(\boldsymbol{A} - \boldsymbol{f_1}\boldsymbol{g_1}^T - \boldsymbol{f_2}\boldsymbol{g_2}^T)}{\text{norm}(\boldsymbol{A})} \tag{5.84}$$

Thus, steps (2)-(6) are repeated for each iteration until the matrix convergence factor C_i is less than a predetermined criterion (e.g., 10^{-6}).

The values of $\boldsymbol{f_1}, \boldsymbol{g_1}, \boldsymbol{f_2}, \boldsymbol{g_2}$and convergence factor for five iterations are shown in Table *5*. The calculation reaches a convergence after five iterations so that the addition of the two modes is equal to the original matrix

$$\boldsymbol{f_1}\boldsymbol{g_1}^T + \boldsymbol{f_2}\boldsymbol{g_2}^T = \begin{bmatrix} 1 & 2 \\ -1 & 1 \end{bmatrix} = \boldsymbol{A} \tag{5.85}$$

Table 5 The values *of* $\boldsymbol{f_1}, \boldsymbol{g_1}, \boldsymbol{f_2}, \boldsymbol{g_2}$ and convergence factors for seven iterations

i	1	2	3	4	5
$\boldsymbol{f_1}$	$\begin{bmatrix} 0.5 \\ -1 \end{bmatrix}$	$\begin{bmatrix} 0.412 \\ -0.912 \end{bmatrix}$	$\begin{bmatrix} 0.406 \\ -0.905 \end{bmatrix}$	$\begin{bmatrix} 0.405 \\ -0.905 \end{bmatrix}$	$\begin{bmatrix} 0.405 \\ -0.905 \end{bmatrix}$
$\boldsymbol{g_1}$	$\begin{bmatrix} 1.6 \\ 0.4 \end{bmatrix}$	$\begin{bmatrix} 1.522 \\ 0.711 \end{bmatrix}$	$\begin{bmatrix} 1.517 \\ 0.722 \end{bmatrix}$	$\begin{bmatrix} 1.517 \\ 0.722 \end{bmatrix}$	$\begin{bmatrix} 1.517 \\ 0.722 \end{bmatrix}$
$\boldsymbol{f}_2$	$\begin{bmatrix} 1 \\ 1 \end{bmatrix}$	$\begin{bmatrix} 1.059 \\ 1.027 \end{bmatrix}$	$\begin{bmatrix} 1.061 \\ 1.028 \end{bmatrix}$	$\begin{bmatrix} 1.061 \\ 1.028 \end{bmatrix}$	$\begin{bmatrix} 1.061 \\ 1.028 \end{bmatrix}$
$\boldsymbol{g}_2$	$\begin{bmatrix} 0.4 \\ 1.6 \end{bmatrix}$	$\begin{bmatrix} 0.365 \\ 1.609 \end{bmatrix}$	$\begin{bmatrix} 0.363 \\ 1.609 \end{bmatrix}$	$\begin{bmatrix} 0.363 \\ 1.609 \end{bmatrix}$	$\begin{bmatrix} 0.363 \\ 1.609 \end{bmatrix}$
C_i	0.151	0.0073	$3.33e^{-4}$	$1.47e^{-5}$	$6.52e^{-7}$

PGD for high-dimensional tensor decomposition

One benefit of PGD is that it can be used on high-dimensional tensor decomposition rather than the matrix decomposition that can be handled by SVD/PCA. A tensor is typically represented as a (potentially multidimensional) array, just as a vector (i.e., one-dimensional tensor) is represented by a one-dimensional array. The numbers in the array are denoted by indices giving their position in the array as subscripts following the symbolic name of the tensor. For example, a matrix can be treated as a two-dimensional tensor represented by a two-dimensional array. The values of this two-dimensional tensor (i.e., matrix) $\boldsymbol{A}$ could be denoted A_{ij}, where i and j are position indices.

An example is shown here: a four-dimensional tensor A_{ijkl} can be decomposed by PGD as

$$A_{ijkl} \approx \sum_{m=1}^{n} \boldsymbol{f_m} \otimes \boldsymbol{g_m} \otimes \boldsymbol{h_m} \otimes \boldsymbol{p_m} \tag{5.86}$$

where $\boldsymbol{f_m}$, $\boldsymbol{g_m}$, $\boldsymbol{h_m}$, and $\boldsymbol{p_m}$ are vectors in an index of m, and $\otimes$ is the operation of the outer product which produces a tensor by inputting multiple vectors. For example, $(\boldsymbol{f} \otimes \boldsymbol{g} \otimes \boldsymbol{h})_{ijk} = f_i g_j h_k$. The number of modes is given as n.

The high-dimensional PGD can be solved using the same procedure demonstrated above. The Matlab implementation of higher-dimensional PGD can be found at a website created by Northwestern post-doctoral fellow Dr. Ye Lu[30]. This code can automatically determine the optimal number of modes given an approximation accuracy. In summary, PGD decomposes raw data into a series of 1D vectors (functions) and performs reduced order modelling for high-dimensional tensors. For 2D tensor (i.e., matrix) decomposition provides the same outcome (under appropriate normalization) as the SVD/PCA. PGD can be applied to a dataset that is described by a 3D, 4D, or even higher dimensional array (tensor). Therefore, PGD can be seen as a higher dimensional extension of SVD/PCA. Moreover, the PGD might provide more efficient solution for high-dimensional complex problems due to its iterative feature during the solution.

5.4. Eigenvalues and eigenvectors [Advanced topic]

Consider a real n-by-p matrix $\boldsymbol{B}$, if the covariances between the rows of the matrix are of interest the covariance matrix of $\boldsymbol{B}$ can be computed by

$$\boldsymbol{C}_B = \frac{1}{n-1} \boldsymbol{B}^T \boldsymbol{B} \tag{5.87}$$

where n is the number of columns of the matrix $\boldsymbol{B}$. Superscribe T denotes the matrix transpose.

Since the matrix $\boldsymbol{C}_B$ is a real symmetric p-by-p matrix, i.e., ${\boldsymbol{C}_B}^T = \boldsymbol{C}_B$, it has real eigenvalues and eigenvalues defined by[31]

[30] https://yelu-git.github.io/hopgd/

[31] Hawkins, T. (1975), "Cauchy and the spectral theory of matrices", Historia Mathematica, 2: 1–29

$$\boldsymbol{C}_B = \boldsymbol{G\Lambda G}^T = [\boldsymbol{g_1} \quad \boldsymbol{g_2} \quad \cdots \quad \boldsymbol{g_p}] \begin{bmatrix} \lambda_1 & 0 & \dots & 0 \\ 0 & \lambda_2 & \dots & 0 \\ \dots & \dots & \dots & 0 \\ 0 & 0 & 0 & \lambda_p \end{bmatrix} \begin{bmatrix} \boldsymbol{g_1^T} \\ \boldsymbol{g_2^T} \\ \dots \\ \boldsymbol{g_p^T} \end{bmatrix} \tag{5.88}$$

where $\boldsymbol{g_1}$, $\boldsymbol{g_2}$, ..., $\boldsymbol{g_p}$ are eigenvectors and they are orthonormal, and λ_1 , λ_2 , ..., λ_p are eigenvalues. The eigenvalues and eigenvectors can be obtained by solving the following equations

$$|\boldsymbol{C}_B - \lambda\boldsymbol{I}| = 0 \tag{5.89}$$

$$(\boldsymbol{C}_B - \lambda\boldsymbol{I})\boldsymbol{g} = \boldsymbol{0} \tag{5.90}$$

where $|\cdot|$ is the determinant of the matrix, and $\boldsymbol{I}$ is the unit matrix.

5.5. Mathematical relation between SVD and PCA [Advanced topic]

The principal components can be obtained by solving the eigenvectors and eigenvalues problem (eigenvectors are indeed the principal components). However, it is sometimes computationally expensive to solve the eigenvectors and eigenvalues for a covariance matrix $\boldsymbol{C}_B$ because of the calculation of the determinant. It is more efficient to solve the SVD for the mean-subtracted data matrix directly and obtain the principal components (eigenvectors) because SVD and PCA are mathematically consistent.

Consider a mean-subtracted data n-by-p matrix $\boldsymbol{B}$, the SVD of the matrix $\boldsymbol{B}$ is expressed as

$$\boldsymbol{B} = \boldsymbol{U\Sigma V}^T = [\boldsymbol{u_1} \quad \boldsymbol{u_2} \quad \cdots \quad \boldsymbol{u_p}] \begin{bmatrix} \sigma_1 & 0 & \dots & 0 \\ 0 & \sigma_2 & \dots & 0 \\ \dots & \dots & \dots & 0 \\ 0 & 0 & 0 & \sigma_p \end{bmatrix} \begin{bmatrix} \boldsymbol{v_1^T} \\ \boldsymbol{v_2^T} \\ \dots \\ \boldsymbol{v_p^T} \end{bmatrix} \tag{5.91}$$

where $\boldsymbol{u_1}$, $\boldsymbol{u_2}$, ..., $\boldsymbol{u_p}$ and $\boldsymbol{v_1}$, $\boldsymbol{v_2}$, ..., $\boldsymbol{v_p}$ are singular vectors, and σ_1, σ_2, ..., σ_p are singular values. It can be proven that there are the following relations between singular values/vectors and eigenvalues/eigenvectors:

$$\boldsymbol{G} = \boldsymbol{V} \;\; or \;\; \boldsymbol{g_i} = \boldsymbol{v_i} \tag{5.92}$$

$$\boldsymbol{\Lambda} = \frac{\boldsymbol{\Sigma^2}}{n-1} \;\; or \;\; \lambda_i = \frac{\sigma_i^2}{n-1} \tag{5.93}$$

Where $\boldsymbol{G}$ is the matrix including the eigenvectors $\boldsymbol{g_1}$, $\boldsymbol{g_2}$, ..., $\boldsymbol{g_p}$, $\boldsymbol{\Lambda}$ is the matrix including the eigenvalues λ_1, λ_2, ..., λ_p, and n is the number of columns of the matrix $\boldsymbol{B}$.

Proof:

Given that $\boldsymbol{U}^T\boldsymbol{U} = \boldsymbol{I}$ and $\boldsymbol{V}^T\boldsymbol{V} = \boldsymbol{VV}^T = \boldsymbol{I}$, and $\boldsymbol{\Sigma}$ is a diagonal matrix, the covariance matrix of the matrix $\boldsymbol{B}$ can be expressed as

$$\boldsymbol{C}_B = \frac{1}{n-1}\boldsymbol{B}^T\boldsymbol{B} = \frac{1}{n-1}(\boldsymbol{U\Sigma V}^T)^T\boldsymbol{U\Sigma V}^T = \frac{1}{n-1}\boldsymbol{V\Sigma U}^T\boldsymbol{U\Sigma V}^T$$
$$= \boldsymbol{V}\left(\frac{1}{n-1}\boldsymbol{\Sigma}^2\right)\boldsymbol{V}^T \tag{5.94}$$

Comparing with the form of eigenvectors and eigenvalues in PCA, i.e., $\boldsymbol{C}_B = \boldsymbol{G\Lambda G}^T$, two relations can be obtained:

$$\boldsymbol{G} = \boldsymbol{V} \tag{5.95}$$

$$\boldsymbol{\Lambda} = \frac{\boldsymbol{\Sigma}^2}{n-1} \tag{5.96}$$

Q.E.D

In PCA, a transform is sought so that the covariance matrix of the transformed matrix is diagonal. It can be proven that the covariance matrix of $\boldsymbol{BV}$ is diagonal.

Proof:

$$\boldsymbol{C}_{BV} = \frac{1}{n-1}(\boldsymbol{BV})^T\boldsymbol{BV} = \frac{1}{n-1}(\boldsymbol{U\Sigma V}^T\boldsymbol{V})^T\boldsymbol{U\Sigma V}^T\boldsymbol{V} = \frac{1}{n-1}\boldsymbol{\Sigma U}^T\boldsymbol{U\Sigma}$$
$$= \frac{1}{n-1}\boldsymbol{\Sigma}^2 \tag{5.97}$$

Q.E.D