Principal Component Analysis

Ron Wilson

February 21, 2020

Abstract

The usefulness and practicality of Principal Component Analysis (PCA) is explored in this write-up through four different tests. The tests are experiments filmed from three different cameras, observing the motion of a paint can.

1 Introduction and Overview

Three cameras are observing the motion of a paint can through four different tests. In test 1 (the ideal case), there is a small displacement of the mass of the can in the z direction, causing the entire oscillating motion to be in the z direction, allowing for simple harmonic motion to be observed. Test 2 (the noisy case) is the same as test 1 except that there is some camera shake in the video recordings, making it more difficult to extract the simple harmonic motion. In test 3 (horizontal displacement), the mass is released off-center so as to produce motion in the x-y plane as well as in the z direction. Thus, there is both pendulum motion and simple harmonic oscillations. Finally, test 4 (horizontal displacement and rotation) is similar to test 3 except that the mass also rotates, which effects the motion in the x-y plane and rotation in the z direction. In this write-up, I will analyze the behavior of the motion of the can in each of the four tests using the PCA method.

2 Theoretical Background

In this section, I will provide brief background information relevant to the algorithms of PCA.

2.1 Singular Value Decomposition (SVD)

The SVD of an mxn matrix A is defined as:

$$A = U\Sigma V^*. \tag{1}$$

where the matrices U, Σ, and V are the following:

$$\begin{aligned} U &\in \mathbb{C}^{mxm} \text{ is unitary.} \\ V &\in \mathbb{C}^{nxn} \text{ is unitary.} \\ \Sigma &\in \mathbb{R}^{mxn} \text{ is diagonal.} \end{aligned} \tag{2}$$

Note that every matrix has a corresponding SVD, which can be computed in MATLAB using the function $[U, S, V] = \text{svd}(A)$ (see Appendix A). By computing the SVD of a matrix A, we can easily diagonalize the matrix A. By diagonalizing a system such as in the four tests, we can understand the behavior of the motion of the system. In MATLAB, we can use the function $x = \text{diag}(A)$ to diagonalize the matrix A.

Now, using the fact that the matrices U and V are unitary, we know that they are also orthonormal bases in $\mathbb{C}^{mxm}$ and $\mathbb{C}^{nxn}$, respectively. Thus, any vector in these spaces can be expanded in their bases. Consider the vectors $b \in \mathbb{C}^{mxm}$ and $x \in \mathbb{C}^{nxn}$. So, these vectors can be expanded as the following:

$$b = U\hat{b},\ x = V\hat{x} \tag{3}$$

Using the vector expansions in 3, we can consider the equation $Ax = b$ or the equation $U^*Ax = U^*b$. Using the SVD of the matrix A, notice that the equation becomes $U^*U\Sigma V^*x = U^*b$. Therefore, by equation 3 and the properties of unitary matrices, we have:

$$\Sigma\hat{x} = \hat{b}. \tag{4}$$

Equation 4 shows that A reduces to the diagonal matrix Σ when the range is expressed in terms of the basis vectors of U and the domain is expressed in terms of the basis vectors of V. In practice, we can use the fact that all matrices have an SVD to find the diagonal matrix Σ to understand the behavior of the four tests.

2.2 Principal Component Analysis (PCA)

In each of the four tests, we have motion described by a mass and spring system, which is recorded by three different cameras. If we denote the data from the three cameras with subscripts a, b, and c, then the data collected is represented by the following:

$$\begin{aligned} &\text{camera 1: } (x_a, y_a) \\ &\text{camera 2: } (x_b, y_b) \\ &\text{camera 3: } (x_c, y_c) \end{aligned} \tag{5}$$

where each set (x_j, y_j) is data collected over time of the position in the x-y plane of the camera. All of the data collected can then be gathered into a single matrix:

$$X = \begin{bmatrix} x_a \\ y_a \\ x_b \\ y_b \\ x_c \\ y_c \end{bmatrix} \tag{6}$$

Now that the data has been arranged, we need to address two important issues: noise and redundancy. We can identify redundancy by considering the covariance between two data sets. The covariance matrix appropriate for this problem is the following:

$$C_X = \frac{1}{n-1}XX^T. \tag{7}$$

The diagonal terms of equation 7 are the variances for particular measurements, while the off-diagonal terms are the covariance between measurements. Through the process of diagonalization we can remove the off-diagonal terms and thus remove the redundancies in the data. This is where the SVD comes in, as it naturally diagonalizes the system and orders the largest variances of the particular measurements. This means that the system has been written in terms of its principal components. In the principal component basis, we can diagonalize the system:

$$Y = U^*X \tag{8}$$

The covariance matrix corresponding to this new basis can be written as the following:

$$\begin{aligned} C_Y &= \frac{1}{n-1}YY^T \\ &= \frac{1}{n-1}(U^*X)(U^*X)^T \\ &= \frac{1}{n-1}(U^*U\Sigma V^*)(U^*U\Sigma V^*)^T \\ &= \frac{1}{n-1}\Sigma V^*V\Sigma^T \\ &= \frac{1}{n-1}\Sigma^2 \end{aligned} \tag{9}$$

Since, Σ is designed to be a diagonal matrix, we have made sure to remove all redundancies in the data when using the principal component basis. Note that the issue of noise will be discussed along with limitations of PCA in the results & summary sections (sections 4-5) of this write-up.

3 Algorithm Implementation and Development

Here I will explain the algorithms used for the four tests. Note that the algorithms are the same for each test. Please refer to Appendix B for the MATLAB code corresponding to these algorithms.

1. For each test, I will first load the three cameras and find the initial position of the flashlight on top of the paint can manually. I do this by viewing the first frame from each camera with the MATLAB function `imshow`, then clicking on the flashlight manually. The code reads my input from the function `ginput` and stores the position as the first column in a matrix of indices.

2. Now that I have the initial position, I can place a window around that location and track the paint can through time. For each iteration through time, I can find the exact location of the flashlight within the window by finding the maximum pixel within the window. This corresponds to the pixel intensity of the flashlight. Once again, I will store the x and y values in the corresponding columns of the indexing matrix.

   - It is important to note that the window could extend beyond the boundary of the frame. To avoid this from happening, I am comparing the size of the window to the size of the frame and truncating the window if it is too large.

3. Now that I have a matrix of indices corresponding to the position of the flashlight through time, I can generate the position matrix X as seen in equation 6.

   (a) First, I need to adjust the indices to align the data from all three cameras. I want to normalize the indexing matrix for all three cameras to get the maximum and minimum values in the data to occur at the same time for all three cameras. In figure 1, you can see the normalized indices from test 1 and how the peaks and valleys in the data occur at the same time.

   (b) Next, I want to set the three cameras to start at the same time. So, I truncate the unnecessary data at the end of each indexing matrix to achieve this result. Note that these values are different for each camera and for each of the four tests.

   (c) Now, I can create the matrix X from the values within the indexing matrices over time. However, I will also subtract the mean of the data to make sure the data is centered for all measurements when I compute the variance and covariance.

4. Finally, I can compute the SVD of the matrix X. As explained in section 2, I will diagonalize the resulting Σ matrix to remove redundancies in the data. I can then use the diagonalized Σ to calculate the variance of the data from all of the measurements. As in equation 8, I will find the principal component basis to study the projection of the principal components through time.

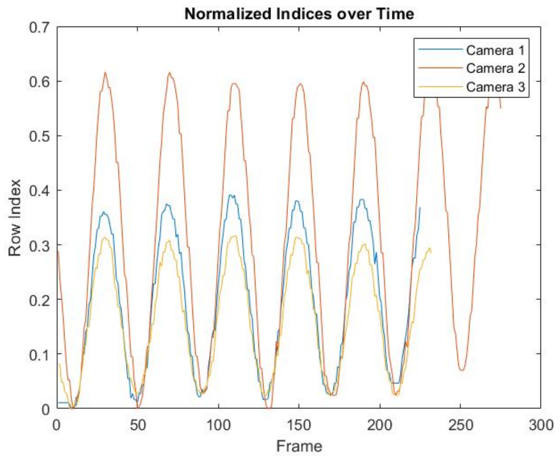

Figure 1: Alignment of the 3 Cameras in Test 1

- I will produce plots of the variance and time evolution behavior over the modes of the PCA to understand what is going on in each of the four tests.

In the next section of this write-up, I will discuss the results of the plots produced by these algorithms.

4 Computational Results

4.1 Test 1: Ideal Case

See figure 2 for the plots of the results of test 1. The singular values from the six PCA modes, which are found from Σ, are in the plot on the top left, while the percentage of the variance corresponding to the singular values are in the plot on the top right. The projections onto the principal component basis (from equation 8) are in the middle plot. The time evolution behavior of the system is found in the bottom plot. This structure for the plots is the same for all four tests.

Because there was little noise in the first test, the three cameras captured mostly the same data, causing a lot of redundancies. We can see this in the percentage of the variance. The percent of the variance for the first singular value is 74.62%, and the second singular value has a variance of approximately 20%. Thus, the first two PCA modes account for about 95% of the total variance in the system. The remaining singular values captured very little variance, and thus can be mostly ignored as noise in the system. In the PCA projections plot, we can see that the first mode accounted for most of the behavior of the system, which corresponds to its large variance. This is backed up by the time evolution behavior, where the first mode is has a smooth, oscillating behavior, while the rest are essentially noisy data. We can see that the oscillations of the first mode appears to slow down over time, which indicates the mass is slowing down, likely due to air resistance.

4.2 Test 2: Noisy Case

PCA was very effective in analyzing the system in the first test due to the low amount of noise in the system. Now, we'll look at a noisier version of test 1. Here, the same test is run, but the three cameras are shaking, creating more noticeable noise in the data. See figure 3 for the results of test 2.

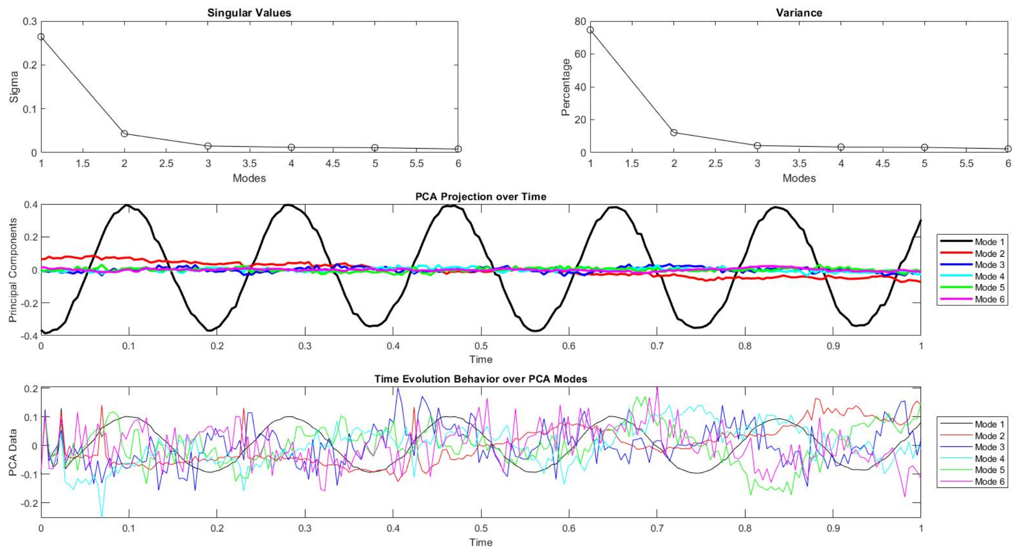

Figure 2: PCA of Test 1

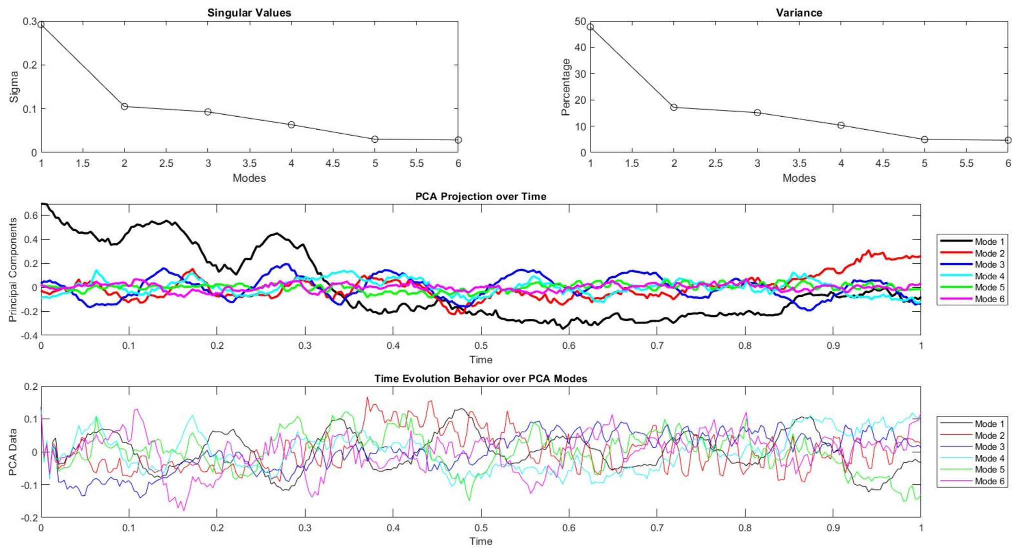

Figure 3: PCA of Test 2

Here, the variance is spread out a lot more among the six modes. The first singular value has a variance of 47.76%. Thus, it is hard to understand the behavior of the system from the PCA projections or the time evolution behavior plots because all of the modes have noisy data. We know what is happening here from the results of test 1, but here, the data is too noisy to be analyzed effectively with the PCA method.

4.3 Test 3: Horizontal Displacement

See figure 4 for the results of test 3. Here, we see that the first three modes account for the majority of the variance in the system, with the first mode containing 39.3% of the variance. Because the variance is still spread out amongst the modes, it leads to a very noisy plot of the time evolution behavior. However, we can

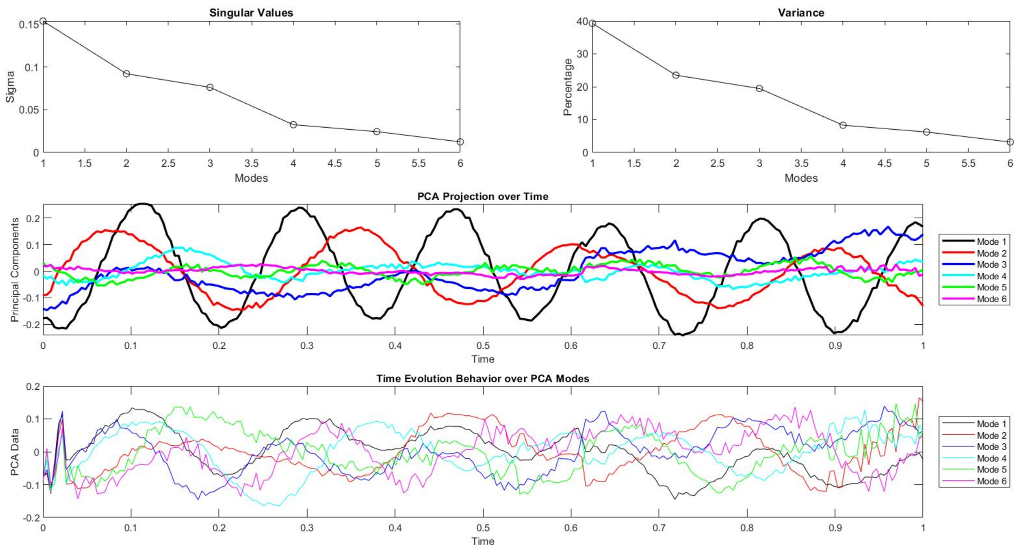

Figure 4: PCA of Test 3

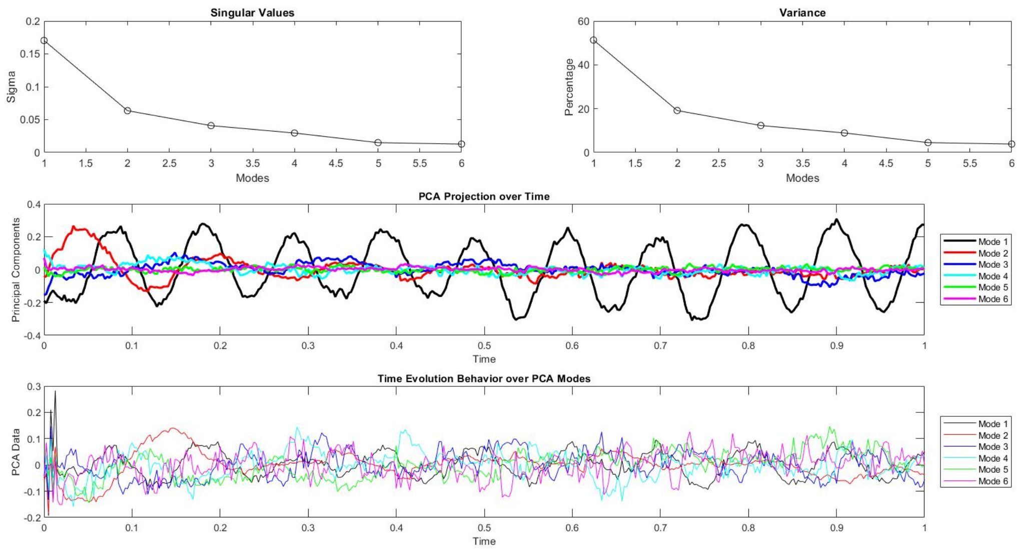

Figure 5: PCA of Test 4

see that the first three modes are displaying oscillatory behavior in the PCA projections plot. This is expected as we have a pendulum motion added into the simple harmonic motion that was present in test 1.

4.4 Test 4: Horizontal Displacement and Rotation

Finally, see figure 5 for the results of test 4. Test 4 is the same as test 3, except that there is rotation added into the motion of the paint can. Here, the data is surprisingly less noisy than it was in test 3. The variance of the first singular value is 51.32% of the total variance, and the total variance isn't as spread out across the six modes as it was in test 3. This allows us to clearly see the oscillating behavior of the first singular value in the PCA projections plot. We can also see the rotational behavior in mode 2, which begins with a large oscillation, but then quickly flattens out. However, the plot of the time evolution behavior is mostly useless. There is a sudden jump at the beginning, which we can attribute to the way the paint can was rotated at the beginning of the test, but other than that, this plot tells us almost nothing about the motion.

5 Summary and Conclusions

Using the concepts of Principal Component Analysis and Singular Value Decomposition we were able to write numerical methods to generate plots that describe the behavior of the motion of a paint can filmed from three different cameras in four different tests.

However, I can conclude that the issue of noise brought up in section 2 is very important. In the ideal (little noise) case, it was easy to understand the plots from PCA. However, the remaining tests all had more noise and were very difficult to understand when using the method of PCA. If examining a system with very noisy data, I would not recommend using PCA.

Appendix A MATLAB Functions

The following is a list of the important MATLAB functions that were used:

- `load(filename)` loads data from `filename` into the MATLAB workspace.
- `X = zeros(sz1,...,szN)` returns an `sz1-by-...-by-szN` array of zeros where `sz1,...,szN` indicate the size of each dimension.
- `imshow(I)` displays the image `I` in a figure.
- `[x,y] = ginput(n)` allows you to identify the coordinates of `n` points. To choose a point, move your cursor to the desired location and press either a mouse button or a key on the keyboard.
- `S = sum(A)` returns the sum of the elements of `A` along the first array dimension whose size does not equal 1.
- `S = sum(A,dim)` returns the sum along dimension `dim`.
- `M = max(A)` returns the maximum elements of an array.
- `M = min(A)` returns the minimum elements of an array.
- `[row,col] = find(X)` returns the row and column subscripts of each nonzero element in array `X` using any of the input arguments in previous syntaxes.
- `plot(X,Y)` creates a 2-D line plot of the data in `Y` versus the corresponding values in `X`.
- `y = linspace(x1,x2,n)` returns a row vector of `n` evenly spaced points between `x1` and `x2`.
- `M = mean(A,dim)` returns the mean along dimension `dim`.
- `B = repmat(A,r1,...,rN)` specifies a list of scalars, `r1,..,rN`, that describes how copies of `A` are arranged in each dimension. When `A` has `N` dimensions, the size of `B` is `size(A).*[r1...rN]`.
- `[U,S,V] = svd(A)` performs a singular value decomposition of matrix `A`, such that `A = U`*`S`*`V`'.
- `x = diag(A)` returns a column vector of the main diagonal elements of `A`.
- `subplot(m,n,p)` divides the current figure into an `m`-by-`n` grid and creates axes in the position specified by `p`.

Appendix B MATLAB Code

The MATLAB code used to generate the results of this write-up (`Principal_Component_Analysis.m`) can be found at the following GitHub repository: `https://github.com/ronwilson016/Data-Science-Projects/tree/master/Principal%20Component%20Analysis/MATLAB%20Code`