Extracting Harmonic Dynamics from Noisy Data Using PCA

Ben Francis,

Febryary 24, 2021

Abstract

Principle components analysis (PCA) is used to determine the dynamics of interest for a variety of spring and pendulum experiments. Data pre-processing methods are explored to go from video-captured data to a data matrix in the form needed for PCA. Singular value decomposition and variance are seen to be key concepts and tools toward this purpose.

1 Introduction and Overview

The purpose of this report is to determine the most important underlying dynamics of a measured physical system. The system being examined here contains physically realized spring and pendulum motion. Newtonian physics has shown that spring motion and the small angle approximation of pendulum motion should both correspond to sinusoidal motion. However, real systems are not equivalent to the theoretical constructs which they are expected to act like. Additional physical effects due to the imperfect realization of a frictionless, point-mass spring and non-linear effects not incorporated into certain approximations result in additional dynamics and noise. In order to characterize how a specific system acts, data must be collected upon it. This has been done here for a suspended, weighted spring with a variety of different initial conditions. The weight at the end of the spring is a bucket with a flashlight attached to aid in data pre-processing. The steps between pre-processing the collected data and a comparison of the theoretical and observed behavior will be described here.

1.1 Dataset

In order to examine the spring motion under question, an experiment was performed with a particular data collection process. The experiment consisted of pushing a bucket attached to a spring over four different trials. First, the bucket was pushed relatively straight downward, to provide a baseline for idealized, simple spring motion. Second, camera shaking was introduced to show how non-ideal data collection scenarios can be handled. Third, the weight was released off-center to introduce two-dimensional dynamics, including pendulum motion. Fourth, the weight was released with a rotation of the bucket about its vertical axis to introduce additional interesting dynamics to the system, potentially coupling to the rotational properties of the spring and its relation to the spring constant.[2] Three cameras were used to record the system. This both adds additional information to the dataset and introduces redundancy. If only one camera was used, it would be difficult to determine how the three-dimensional system was operating. In using three cameras, there is much more information about the motion of the bucket in three-dimensions. However, there is also additional redundancy since some of the information from one camera is not contained in the video of the others. Principal component analysis (PCA) will be used to help remove this redundancy and determine the most important dynamics of the system. This also helps remove noise from the signal. In order to use PCA, a single coordinate path for the bucket is needed for each experiment. Approximating this is taken care of in the data pre-processing step.

2 Theoretical Background

2.1 Data Pre-Processing

The first step of data pre-processing is considering in what form the data is given. While the system of interest is the physical instantiation of a spring, data is collected in a way that introduces non-trivial complexities into the data analysis process. One of the easiest methods of collecting data on physical systems is by use of video cameras, which collect electromagnetic light from the nearby environment of the system in question. However, these light waves come in a continuum, and the process of segmenting out discrete objects is a complex process in its own right. Ideally, the coordinates of the spring mass would be known at each time point of the video. These would play the role of the point mass concept associated with simple Newtonian springs and pendulums. In the dataset used, only continuous video is recorded, so a method is needed to identify the bucket with a point at each timeframe throughout the video in a reliable way. In preparation for this difficulty, a flashlight was put on top of the bucket to make it identifiable by having a high brightness.

2.2 Covariance

When data is collected about a system, there is a high chance of statistical dependence between observations. Data vectors with higher statistical dependence should have a larger covariance. To determine the covariance of two data vectors, their dot product is taken and they are normalized by dividing by their length minus 1. This projects one onto the other, determining their similarity in the high-dimensional data space. Independence corresponds to a covariance of zero, which is akin to orthogonal vectors in the data vector's vector space. The vector's covariance with itself is just its variance, which is not related to independence. If there are multiple observations, arranged in a matrix such that each row is a data vector, the covariance of each can be found by multiplying the data matrix by its transpose and dividing by the number of data points in each data vector minus one. This gives the covariance matrix $C_X = \frac{1}{n-1}XX^T$. The diagonal elements of this square matrix will then be the variances, and the off-diagonal elements will be the covariances. Then a more redundant dataset will have larger off-diagonal terms in its covariance matrix, and a less redundant dataset will have smaller off-diagonal terms. Additionally, data vectors with higher variance are considered to correspond to dynamics of interest, since they fluctuate more. However, this is not true in all systems. If the data matrix could be transformed such that the covariances were zero, then this would represent the ideal basis for considering the dynamics of the system, since there would be no redundancy. Additionally, if the variances were ordered then the degree to which the bases of the transformed data contribute to the dynamics of interest would be known. Only considering the bases corresponding to high variance is one method of eliminating noise in the data. Each of these are accomplished by diagonalizing the covariance matrix, which can be done using the singular value decomposition of the data matrix. [3]

2.3 Singular Value Decomposition

The basic idea of the singular value decomposition (SVD) is to describe any matrix as a combination of a rotation, scaling, and rotation in the vector space. Each of these steps itself corresponds to a matrix, in the geometric interpretation of linear algebra where matrices are linear transformations in a vector space. This factorization of a matrix X is given by:

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

U and V^* are unitary matrices, and so their columns are orthonormal bases. Σ is a diagonal matrix with all off-diagonal terms of 0 and with diagonal elements that are non-negative and ordered, called the matrice's *singular values*. Singular values σ_i are related to eigenvalues in that σ_i^2 gives the non-negative eigenvalues associated with the self-adjoint matrix X^TX. Among the important properties of SVD is that it works on any matrix, unlike eigendecomposition. Additionally, and critically for PCA, if a data matrix X is projected onto the left singular matrix U, such that the projected matrix is $Y = U^*X$, the covariance of Y is: [3]

$$C_Y = \frac{1}{n-1}YY^T = \frac{1}{n-1}U\Sigma V^*(U\Sigma V^*)^T = \frac{1}{n-1}U\Sigma V^*V\Sigma U^* = \frac{1}{n-1}\Sigma^2$$

$$C_Y = \frac{1}{n-1}\Sigma^2 \tag{2}$$

2.4 Principal Components Analysis

Since Σ is diagonal, all of the covariance terms are zero. Since the diagonal of Σ is ordered, the rows, from left to right, of Y represent the vectors with the highest variance, and so presumably represent the dynamics of interest. These are the *principal components* of the data matrix X. Since U is orthonormal, these provide a new orthonormal coordinate system, with each axis ranked in terms of its contribution of variance into the dynamics. If the first few diagonal elements of Σ represent a large portion of the sum of the trace of Σ, which is the total *energy* of the data vector, then the corresponding principal components are considered to capture an amount of this energy proportional to their diagonal element, which are the singular values squared. If a subset of principal components contain a large percentage of the total energy, the rest of the principal values can be discarded, resulting in dimensional-reduction of the system. The Nth partial sum of the projections onto the new basis captures as much information of the underlying matrix X as possible. Ignoring principal components with low energy also removes low-variance noise from the system. The basis vectors of U provide a *proper orthogonal decomposition* of the system, a type of function expansion. Here each successive basis function is orthonormal in whatever direction maximizes variance.

3 Algorithm Implementation and Development

Algorithm 1: Data Pre-Processing

```
Data: Video files
Result: Data matrices representing recording of bucket from three cameras
Load 12 video files;
Turn video frames to grayscale, integer values to double;
for each video do
    coordinate_list_x, coordinate_list_y ← empty list
    Set brightness threshold and plausible spatial boundary for bucket coordinates;
for each video time frame do
    for each video do
        coordinate_pool_x, coordinate_pool_y ← empty list
        for each pixel do
            Apply logic mask:
            if pixel brightness above threshold and within spatial boundaries then
                Add x and y values to coordinate pool;
            else
                set pixel brightness to 0;
        add mean of coordinate values in coordinate pools to coordinate lists;
smooth coordinate lists
for each of 4 experiments do
    create data matrix X, with each row the smoothed coordinate lists corresponding to the
      experiment;
slice off ends of each row of each data matrix so all 3 cameras are synchronized;
```

```
Algorithm 2: PCA
  Data: Data coordinate matrices
  Result: Principal Component Projections of Data
  for each data matrix row do
      subtract mean of row, divide by $\sqrt{\text{number of data points - 1}}$
  for each data matrix do
      [u, s, v] ← Take SVD of each data matrix
      Y ← Project data matrix onto principal components matrix, u' *data matrix
  plot first 6 projections of principal components, the rows of Y;
  plot 6 largest eigenvalues $\sigma_i^2$ to determine percentage of variance contained in each projection;
```

The first task to perform is data pre-processing. The data is given as a 4-dimensional array of integers. Three of these dimensions correspond to different color values, so `rgb2gray` is used to compress this to a 2-dimensional array in grayscale. `double` is used to convert the integers into the double-precision number data type. A single coordinate point needs to be associated with the bucket at each time point. To do this, it is noticed that in grayscale the bucket corresponds to high brightness values, which are scaled between 0 and 255, due to the flashlight. A logical mask can be applied to the brightness values of each time frame, which determines if it plausible that each pixel is part of the bucket or not.[4] This requires a set of conditions that a logic operator can test for. For each video, a brightness threshold was determined for which most pixels above the threshold correspond to the bucket. This does not need to capture the entire bucket, but only to exclude points that are not apart of the bucket. Additionally, the path of the bucket in each video never exceeds certain spatial boundaries, so all points outside of these boundaries are excluded. This results in a set of possible coordinates for each time frame that could correspond to the bucket. These are averaged to associate the bucket with one `x` and one `y` coordinate for each time frame. This results in a list of x-coordinates and y-coordinates for each video. To account for noise due to the process of determining these single coordinates from the video, the MatLab `smoothdata` function is used to smooth the coordinate paths. The videos are not synchronized and not of the same length, so the coordinate paths are examined and trimmed such that they approximately align and are of the same length. These processed coordinate lists are then compiled into data matrices for each experiment, with each matrix containing a row for: the x-coordinates list for camera 1, the y-coordinates list for camera 1, the x-coordinates list for camera 2, and so on, ending with the y-coordinates list for camera 3. Each then has 6 rows and a number of columns corresponding to the number of data points. This provides the data in the necessary format for the principal component analysis.

The first step in performing the principal component analysis is to set the mean of each data to 0 and their variance to 1. This is done by subtracting the mean of each data row from the corresponding row and dividing each element by the square root of the data points minus one. The singular value decomposition of each modified data matrix is then taken. The singular values squared give the energies for the principal components. The principal components are found by projecting X onto U^*, The principal components and their energies can then be plotted for analysis.

4 Computational Results

Figure 1 shows how the logical mask process and position averaging works. Figure 4 in Appendix A shows the smoothed paths of the X and Y coordinates associated with the movement of the bucket throughout each experiment. Beyond noticeable oscillatory behavior, not much can be determined from these initial results. Each experiment's principal components projection for the first six principal components are shown in Figure 3 and the percentage of the energy they capture are shown in Figure 2. The results of PCA are not always readily interpretable and they are partially black boxes. However, some insights can be determined by examining the results of each case.
In the first experiment, the first two modes capture 95% of the energy. The first mode is clearly oscillatory with a slight positive linear increase over time. The second mode is 90 degrees out of phase with the first and approximately half of the frequency. This could correspond to the derivative of the first mode, where

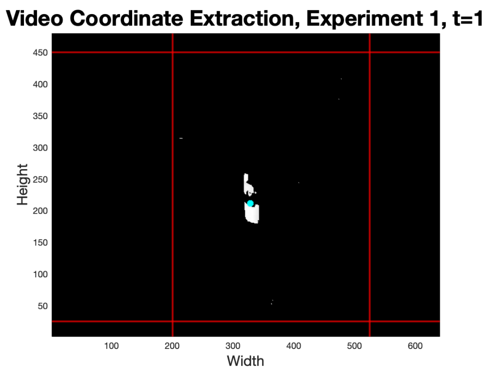

Figure 1: Example of logic mask based upon brightness and spatial location. Also shows the calculated average of remaining pixels.

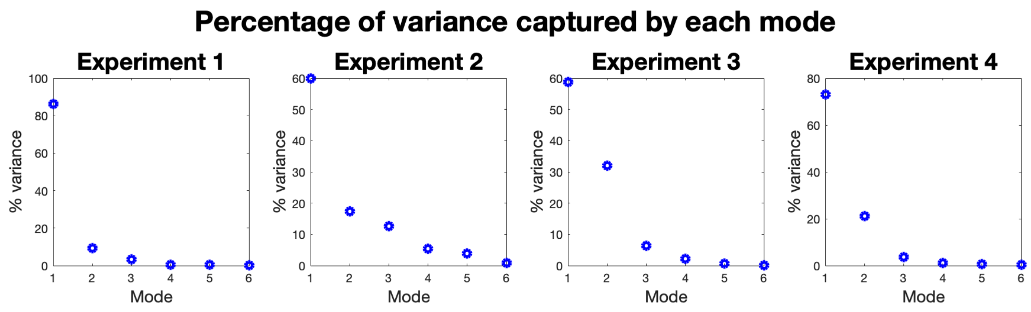

Figure 2: Percentage of variation captured by each mode of the SVD.

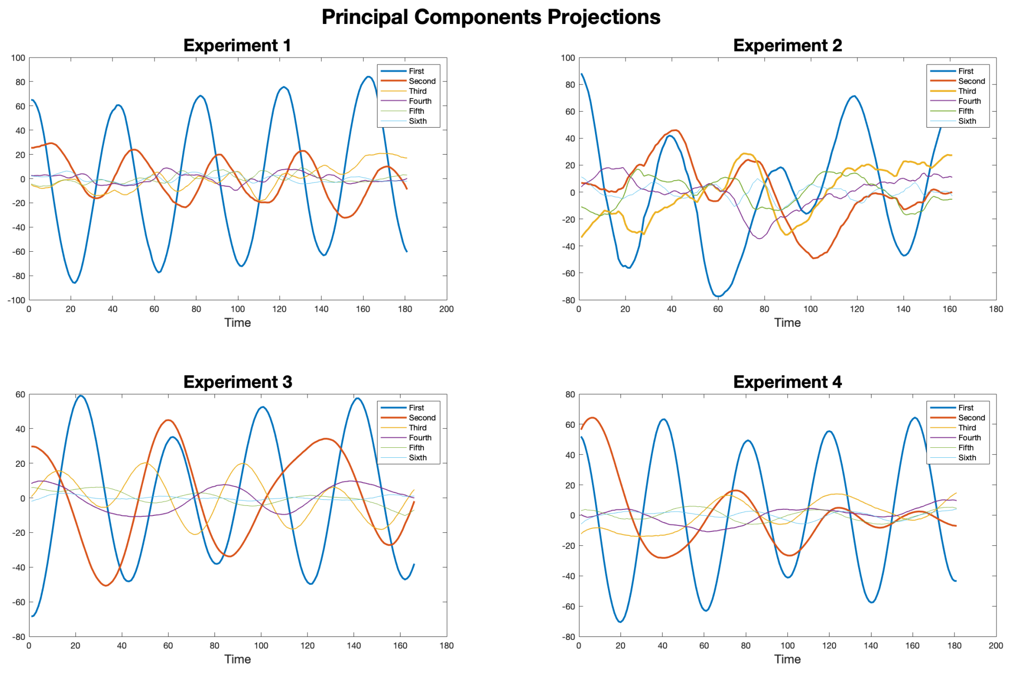

Figure 3: Projections of data onto principal components from each experiment.

the first mode would have a period of approximately 0.5. All lower modes appear to have a low variance, making them candidates for removal during dimensionality reduction. These likely correspond to the lateral motion, which was more or less random and representative of noise.

In the second experiment, the first three modes capture 90% of the energy. The first mode has the same frequency as observed in the first experiment, but multiplied by some non-trivial function, in comparison with the linear component associated with the first mode in the first experiment. The second mode is less interpretable, though it shares much of the general shape of the third mode. All lower modes are less interpretable and appear increasingly to have low signal-to-noise ratio. While the first mode corresponded to an experimental observation of a simple physical phenomena in spring motion, this experiment introduced much less modelable phenomena due to the purposeful camera shaking due to a human hand.

In the third experiment, the first two modes capture 91% of the energy. The first six modes are much more distinct than in the previous experiments, probably due to the lack of shaking and the introduction of periodic dynamics in the other dimensions. Specifically, pendulum motion is now introduced into the system, creating additional modes with high variance. In the first experiment, all horizontal movement was effectively noisy, whereas now it is driven by dynamics of interest. The first mode and third modes here appear to correspond to the first and second modes of the first experiment. The second mode likely corresponds to the introduced pendulum motion, with the fourth mode perhaps seeming like the derivative of the second. The fifth and sixth modes are harder to interpret, but they are significantly less noisy than the lower modes of the first two experiments.

In the fourth experiment, the first two modes capture 94% of the energy. The first appears to be the previous major spring motion mode, while the second is an oscillatory mode with damping, which is novel to this experiment.[1] The novel feature of this experiment was the rotation of the bucket. This perhaps resulted in energy exchange with the spring, which derives its spring motion for its rotated coils. As such, this energy may have been transferred into the spring, decreasing its stiffness and transitioning the dynamics of the system to a dampened oscillator [**hooke**]. However, this is only a hypothesis; determining the origin of these dynamics requires an additional level of analysis of the hypothesized physics of the system which is not undertaken here. The lower modes resemble those of the third experiment. Each of the principal components examined provide significantly more information than the smoothed coordinate paths prior to applying the PCA methods, in addition to being more human understandable.

5 Summary and Conclusions

The measured visual data of a spring system under various initial conditions was processed to create single data points at each time, simplifying the process so that principal component analysis can be applied to it. This was done using a logical map based upon unique characteristics of the system, namely the bucket's high brightness and confined spatial trajectory. Averaging of possible points and smoothing of the resultant trajectories then allowed for the construction of a data matrix in the form needed for PCA. SVD was used to determine the principal components, which when projected upon gives a lower-dimensional representation of the dynamics of interest. The results were explainable by the expected phenomenology, including observing one-dimensional spring motion, noise from camera shaking, one-dimensional pendulum motion, and damped harmonic motion due to a rotation applied to the spring weight.

References

[1] *Damped sine wave.* URL: `https://en.wikipedia.org/wiki/Damped_sine_wave`.

[2] *Hooke's Law.* URL: `https://en.wikipedia.org/wiki/Hooke%27s_law`.

[3] Jose Nathan Kutz. *Data-driven modeling & scientific computation: methods for complex systems & big data.* Oxford University Press, 2013.

[4] *Mask (Computing).* URL: `https://en.wikipedia.org/wiki/Mask_(computing)`.

Appendix A Smoothed Coordinate Paths

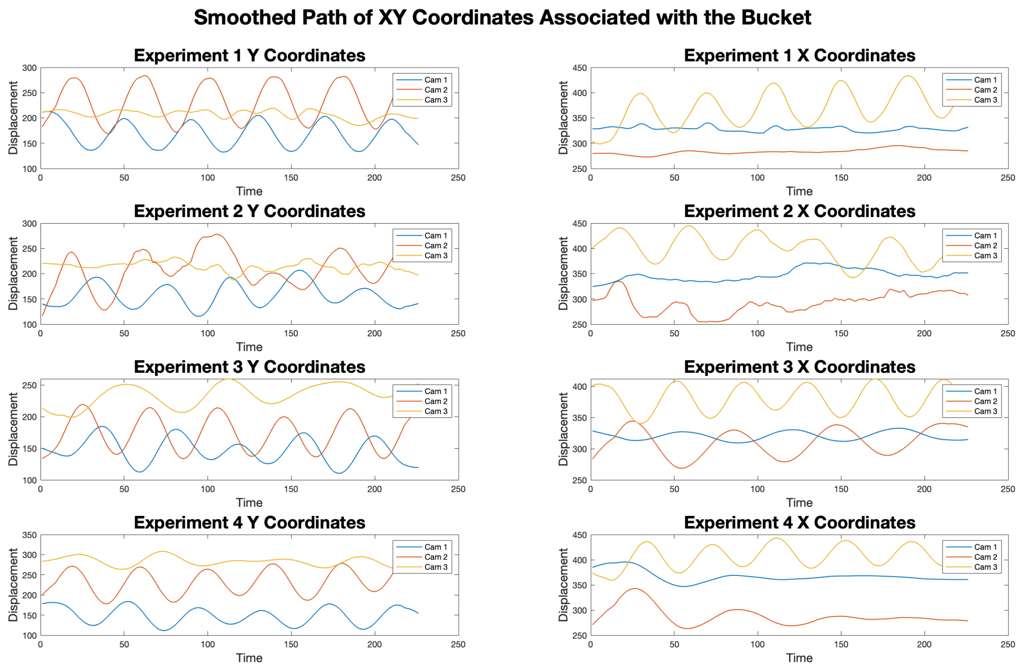

Figure 4: The paths of the x and y coordinates associated with the bucket at each time frame, for each experiment from each camera.

Appendix B MATLAB Functions

- `I = rgb2gray(RGB)`: Converts color images to grayscale by converting color values into luminance.
- `Y = double(X)`: Converts values in array X to the double-precision datatype, if possible.
- `B = flipud(A)`: Returns array A with its rows flipped in the updown direction.
- `B = smoothdata(A)`: Returns a moving average of the elements of vector A using a sliding window whose fixed length is determined using heuristics.
- `B = repmat(A,m,n)`: Repeats a copy of a vector A such that it fills a matrix of size $m * n$.
- `[U,S,V] = svd(A)`: Returns the singular value decomposition of matrix A, such that $A = USV^*$
- `x = diag(A)`: Returns a column vector of the diagonal elements of A.

Appendix C MATLAB Code

See next page.

```
load cam1_1.mat; load cam1_2.mat; load cam1_3.mat; load cam1_4.mat; load cam2_1.mat; load cam2_2.mat;
load cam2_3.mat; load cam2_4.mat; load cam3_1.mat; load cam3_2.mat; load cam3_3.mat; load cam3_4.mat;
left_edge = 200; right_edge = 525; top_edge = 450; bottom_edge = 25; filter = 230; % Default
[height, width, rgb, num_frames] = size(vidFrames1_1);
coord_list_x_1_1 = []; coord_list_x_1_2 = []; coord_list_x_1_3 = []; coord_list_x_1_4 = [];
coord_list_y_1_1 = []; coord_list_y_1_2 = []; coord_list_y_1_3 = []; coord_list_y_1_4 = [];
coord_list_x_2_1 = []; coord_list_x_2_2 = []; coord_list_x_2_3 = []; coord_list_x_2_4 = [];
coord_list_y_2_1 = []; coord_list_y_2_2 = []; coord_list_y_2_3 = []; coord_list_y_2_4 = [];
coord_list_x_3_1 = []; coord_list_x_3_2 = []; coord_list_x_3_3 = []; coord_list_x_3_4 = [];
coord_list_y_3_1 = []; coord_list_y_3_2 = []; coord_list_y_3_3 = []; coord_list_y_3_4 = [];
for t=1:num_frames
    % VID 1_1
    raw_vid=vidFrames1_1(:,:,:,t); proc_vid = flipud(double(rgb2gray(raw_vid)));
    coord_pool_x = []; coord_pool_y = [];
    for h=1:height
        for w=1:width
            if (proc_vid(h,w) < filter) || (w  < left_edge) ||...
                    (w  > right_edge) || (h  > top_edge) || (h  < bottom_edge)
                proc_vid(h,w) = 0;
            else coord_pool_x = [coord_pool_x; w];coord_pool_y = [coord_pool_y; h];
            end
        end
    end
    coord_list_x_1_1 = [coord_list_x_1_1, mean(coord_pool_x)];
    coord_list_y_1_1 = [coord_list_y_1_1, mean(coord_pool_y)];
    %NOTE: REPEAT FOR ALL OTHER VIDEOS; SEE GITHUB FOR COMPLETE CODE
        pcolor(proc_vid), shading interp, colormap(gray);
    yline(top_edge,'r','LineWidth',2); yline(bottom_edge,'r','LineWidth',2);
    xline(left_edge,'r','LineWidth',2); xline(right_edge,'r','LineWidth',2);hold on;
    plot(mean(coord_pool_x),mean(coord_pool_y),'c*','LineWidth',150);
    title('Video Coordinate Extraction, Experiment 1, t=1','FontSize',24);
    xlabel('Width','FontSize',16); ylabel('Height','FontSize',16); drawnow
end
X1 = [smoothdata(coord_list_x_1_1); smoothdata(coord_list_y_1_1); smoothdata(coord_list_x_2_1); ...
        smoothdata(coord_list_y_2_1); smoothdata(coord_list_x_3_1); smoothdata(coord_list_y_3_1)];
X2 = [smoothdata(coord_list_x_1_2); smoothdata(coord_list_y_1_2); smoothdata(coord_list_x_2_2); ...
        smoothdata(coord_list_y_2_2);smoothdata(coord_list_x_3_2); smoothdata(coord_list_y_3_2)];
X3 = [smoothdata(coord_list_x_1_3); smoothdata(coord_list_y_1_3); smoothdata(coord_list_x_2_3); ...
smoothdata(coord_list_y_2_3); smoothdata(coord_list_x_3_3); smoothdata(coord_list_y_3_3)];
X4 = [smoothdata(coord_list_x_1_4); smoothdata(coord_list_y_1_4); smoothdata(coord_list_x_2_4);
smoothdata(coord_list_y_2_4);  smoothdata(coord_list_x_3_4); smoothdata(coord_list_y_3_4)];
```

Listing 1: Code to preprocess data. Github: https://github.com/benfrancis314/AMATH582

```
figure(1);
t = 1:226;

% Sync by eye
X1_sync = [X1(1,5:185);X1(2,5:185);X1(3,20:200);X1(4,20:200);X1(5,30:210);X1(6,30:210)];
X2_sync = [X2(1,55:215);X2(2,55:215);X2(3,40:200);X2(4,40:200);X2(5,40:200);X2(6,40:200)];
X3_sync = [X3(1,35:200);X3(2,35:200);X3(3,25:190);X3(4,25:190);X3(5,53:218);X3(6,53:218)];
X4_sync = [X4(1,13:193);X4(2,13:193);X4(3,20:200);X4(4,20:200);X4(5,33:213);X4(6,33:213)];

t1 = 1:length(X1_sync); t2 = 1:length(X2_sync); t3 = 1:length(X3_sync); t4 = 1:length(X4_sync);

% Normalize
X1_norm = [(X1_sync(1,:) - min(X1_sync(1,:)))/max((X1_sync(1,:) - min(X1_sync(1,:))));
    (X1_sync(2,:) - min(X1_sync(2,:)))/max((X1_sync(2,:) - min(X1_sync(2,:)))); ...
           (X1_sync(3,:) - min(X1_sync(3,:)))/max((X1_sync(3,:) - min(X1_sync(3,:))));
           (X1_sync(4,:) - min(X1_sync(4,:)))/max((X1_sync(4,:) - min(X1_sync(4,:)))); ...
           (X1_sync(5,:) - min(X1_sync(5,:)))/max((X1_sync(5,:) - min(X1_sync(5,:))));
           (X1_sync(6,:) - min(X1_sync(6,:)))/max((X1_sync(6,:) - min(X1_sync(6,:))));];
X2_norm = [(X2_sync(1,:) - min(X2_sync(1,:)))/max((X2_sync(1,:) - min(X2_sync(1,:))));
    (X2_sync(2,:) - min(X2_sync(2,:)))/max((X2_sync(2,:) - min(X2_sync(2,:)))); ...
           (X2_sync(3,:) - min(X2_sync(3,:)))/max((X2_sync(3,:) - min(X2_sync(3,:))));
           (X2_sync(4,:) - min(X2_sync(4,:)))/max((X2_sync(4,:) - min(X2_sync(4,:)))); ...
           (X2_sync(5,:) - min(X2_sync(5,:)))/max((X2_sync(5,:) - min(X2_sync(5,:))));
           (X2_sync(6,:) - min(X2_sync(6,:)))/max((X2_sync(6,:) - min(X2_sync(6,:))));];
X3_norm = [(X3_sync(1,:) - min(X3_sync(1,:)))/max((X3_sync(1,:) - min(X3_sync(1,:))));
    (X3_sync(2,:) - min(X3_sync(2,:)))/max((X3_sync(2,:) - min(X3_sync(2,:)))); ...
           (X3_sync(3,:) - min(X3_sync(3,:)))/max((X3_sync(3,:) - min(X3_sync(3,:))));
           (X3_sync(4,:) - min(X3_sync(4,:)))/max((X3_sync(4,:) - min(X3_sync(4,:)))); ...
           (X3_sync(5,:) - min(X3_sync(5,:)))/max((X3_sync(5,:) - min(X3_sync(5,:))));
           (X3_sync(6,:) - min(X3_sync(6,:)))/max((X3_sync(6,:) - min(X3_sync(6,:))));];
X4_norm = [(X4_sync(1,:) - min(X4_sync(1,:)))/max((X4_sync(1,:) - min(X4_sync(1,:))));
    (X4_sync(2,:) - min(X4_sync(2,:)))/max((X4_sync(2,:) - min(X4_sync(2,:)))); ...
           (X4_sync(3,:) - min(X4_sync(3,:)))/max((X4_sync(3,:) - min(X4_sync(3,:))));
           (X4_sync(4,:) - min(X4_sync(4,:)))/max((X4_sync(4,:) - min(X4_sync(4,:)))); ...
           (X4_sync(5,:) - min(X4_sync(5,:)))/max((X4_sync(5,:) - min(X4_sync(5,:))));
           (X4_sync(6,:) - min(X4_sync(6,:)))/max((X4_sync(6,:) - min(X4_sync(6,:))));];

% Plot
subplot(2,4,1), plot(t1,X1_norm(2,:),'r-'); hold on; plot(t1,X1_norm(4,:)); plot(t1,X1_norm(5,:),'b-');
subplot(2,4,2), plot(t2,X2_norm(2,:),'r-'); hold on; plot(t2,X2_norm(4,:)); plot(t2,X2_norm(5,:),'b-');
subplot(2,4,3), plot(t3,X3_norm(2,:),'r-'); hold on; plot(t3,X3_norm(4,:)); plot(t3,X3_norm(5,:),'b-');
subplot(2,4,4), plot(t4,X4_norm(2,:),'r-'); hold on; plot(t4,X4_norm(4,:)); plot(t4,X4_norm(5,:),'b-');
subplot(2,4,5), plot(t1,X1_norm(1,:),'r-'); hold on; plot(t1,X1_norm(3,:)); plot(t1,X1_norm(6,:),'b-');
subplot(2,4,6), plot(t2,X2_norm(1,:),'r-'); hold on; plot(t2,X2_norm(3,:)); plot(t2,X2_norm(6,:),'b-');
subplot(2,4,7), plot(t3,X3_norm(1,:),'r-'); hold on; plot(t3,X3_norm(3,:)); plot(t3,X3_norm(6,:),'b-');
subplot(2,4,8), plot(t4,X4_norm(1,:),'r-'); hold on; plot(t4,X4_norm(3,:)); plot(t4,X4_norm(6,:),'b-');
```

Listing 2: Code to synchronize signals. Github: https://github.com/benfrancis314/AMATH582

```
% Prep data
X1_norm = [X1(1,5:185);X1(2,5:185);X1(3,20:200);X1(4,20:200);X1(5,30:210);X1(6,30:210)];
X2_norm = [X2(1,55:215);X2(2,55:215);X2(3,40:200);X2(4,40:200);X2(5,40:200);X2(6,40:200)];
X3_norm = [X3(1,35:200);X3(2,35:200);X3(3,25:190);X3(4,25:190);X3(5,53:218);X3(6,53:218)];
X4_norm = [X4(1,13:193);X4(2,13:193);X4(3,20:200);X4(4,20:200);X4(5,33:213);X4(6,33:213)];
[m1,n1] = size(X1_norm); [m2,n2] = size(X2_norm);
[m3,n3] = size(X3_norm); [m4,n4] = size(X4_norm);
mn1 = mean(X1_norm,2); mn2 = mean(X2_norm,2);
mn3 = mean(X3_norm,2); mn4 = mean(X4_norm,2);
X1_mean=X1_norm-repmat(mn1,1,n1); X2_mean=X2_norm-repmat(mn2,1,n2);
X3_mean=X3_norm-repmat(mn3,1,n3); X4_mean=X4_norm-repmat(mn4,1,n4);
% SVD
[u1,s1,v1]=svd(X1_mean/sqrt(n1-1)); [u2,s2,v2]=svd(X2_mean/sqrt(n2-1));
[u3,s3,v3]=svd(X3_mean/sqrt(n3-1)); [u4,s4,v4]=svd(X4_mean/sqrt(n4-1));
lambda1=diag(s1).^2; lambda2=diag(s2).^2; lambda3=diag(s3).^2; lambda4=diag(s4).^2;
Y1=u1'*X1_mean; Y2=u2'*X2_mean; Y3=u3'*X3_mean; Y4=u4'*X4_mean;
% Plot
figure(1)
subplot(1,4,1), plot((lambda1 / sum(lambda1)) * 100,'bo','Linewidth',3);
title('Experiment 1','FontSize',20);
xlabel('Mode','FontSize',14); ylabel('% variance','FontSize',14)
subplot(1,4,2), plot((lambda2 / sum(lambda2)) * 100,'bo','Linewidth',3);
title('Experiment 2','FontSize',20);
xlabel('Mode','FontSize',14); ylabel('% variance','FontSize',14)
subplot(1,4,3), plot((lambda3 / sum(lambda3)) * 100,'bo','Linewidth',3);
title('Experiment 3','FontSize',20);
xlabel('Mode','FontSize',14); ylabel('% variance','FontSize',14)
subplot(1,4,4), plot((lambda4 / sum(lambda4)) * 100,'bo','Linewidth',3);
title('Experiment 4','FontSize',20);
xlabel('Mode','FontSize',14); ylabel('% variance','FontSize',14)
sgtitle('Percentage of variance captured by each mode','FontSize',24,'FontWeight','bold');
figure(2)
t1 = 1:length(Y1(1,:)); t2 = 1:length(Y2(1,:));  t3 = 1:length(Y3(1,:)); t4 = 1:length(Y4(1,:));
subplot(2,2,1), plot(t1,Y1(1,:),t1,Y1(2,:),'LineWidth',2); hold on;
plot(t1,Y1(3,:),t1,Y1(4,:),'LineWidth',1); plot(t1,Y1(5,:),t1,Y1(6,:));
title('Experiment 1','FontSize',20); xlabel('Time','FontSize',14);
legend('First','Second','Third','Fourth','Fifth','Sixth')
subplot(2,2,2), plot(t2,Y2(1,:),t2,Y2(2,:),t2,Y2(3,:),'LineWidth',2);hold on;
plot(t2,Y2(4,:),t2,Y2(5,:),'LineWidth',1);plot(t2,Y2(6,:));
title('Experiment 2','FontSize',20); xlabel('Time','FontSize',14);
legend('First','Second','Third','Fourth','Fifth','Sixth')
subplot(2,2,3), plot(t3,Y3(1,:),t3,Y3(2,:),'LineWidth',2); hold on;
plot(t3,Y3(3,:),t3,Y3(4,:),'LineWidth',1); plot(t3,Y3(5,:),t3,Y3(6,:));
title('Experiment 3','FontSize',20); xlabel('Time','FontSize',14);
legend('First','Second','Third','Fourth','Fifth','Sixth')
subplot(2,2,4), plot(t4,Y4(1,:),t4,Y4(2,:),'LineWidth',2); hold on;
plot(t4,Y4(3,:),t4,Y4(4,:),'LineWidth',1); plot(t4,Y4(5,:),t4,Y4(6,:));
legend('First','Second','Third','Fourth','Fifth','Sixth');
title('Experiment 4','FontSize',20); xlabel('Time','FontSize',14);
sgtitle('Principal Components Projections','FontSize',24,'FontWeight','bold');
```

Listing 3: Code to perform PCA. Github: https://github.com/benfrancis314/AMATH582