AMATH 582 Homework 3: Principal Component Analysis

Eric A. Silk

***Abstract*—Principal Component Analysis (PCA) is a highly useful technique for exploratory data analysis which decomposes a series of observations along axes of the most variance, which then allows for discarding of the less important variables to project into a lower dimensional subspace. This report explores its use in tracking the motion of paint can.**

***Index Terms*—Singular Value Decomposition, Principal Component Analysis, Object Tracking, OpenCV**

I. Introduction

II. Theoretical Background

The technique of Principal Component Analysis works by computing the eigenvalues and eigenvectors of the covariance matrix of the data matrix X, which is defined as XX^T. Calculating this directly can be numerically problematic – instead, the Singular Value Decomposition can be use. For full details of the computation, see Dr. Kutz's Book, "Data-Driven Modeling and Scientific Computation."

A. Properties

One of the most important properties of this technique is a relative independence from the bases used in the initial observation of the data. For instance, in this homework, we are given three perspectives that are clearly non-orthogonal. Because of our a priori knowledge of the system, we could conceivably transform the data to force a better perspective. In many cases, this isn't possible. Use of SVD/PCA will allow us to do this more generally. More importantly, PCA will tell us which of the modes are most critical in describing the phenomena. This can be used in a number of ways:

1) Data Exploration, to find the "important" modes to investigate further
2) Feature Engineering, to feed a more limited subset of data to a machine learning algorithm
3) Visualization, to project a high dimension dataset into a visualizable subspace
4) Lossy compression, by discarding the lower energy modes

III. Algorithm Implementation and Development

A. Object Tracking

Object tracking proved to be a large portion of the project. In the end, OpenCV with Python bindings was used to identify and track an object of interest.

Eric Silk is a Masters Student in Applied Mathematics at the University of Washington, and a Research Engineer for Schweitzer Engineering Laboratories, Pullman, WA 99163 (email: esilk16@uw.edu, eric.silk@ericsilk.com)

1) Thresholding: In order to facilitate object isolation, and given that the paint can had a lit flashlight on top of it, a simple thresholding of the grayscale image was optionally applied. Specifically, a "to zero" threshold was used. This works by identifying regions of luminance less than the threshold and setting them to zero. Comparatively, a binary threshold sets areas below the threshold to zero and areas above to the maximum luminance.

2) Bounding Box Selection: The first frame of the video is held on screen until the user draws a bounding box around the object of interest and presses either the space or enter key. This works quite well in most cases, except in instances in which the primary point of interest (the lit side of the flashlight) is not immediately visible.

3) Tracking: Once the bounding box is selected, a tracker provided by OpenCV was used. Specifically, the `CSRT` tracker, or "Discriminative Filter with Channel and Spatial Reliability" tracker (I'm unsure how the acronym matches, but alas). This was the best choice in a rather non-rigorous series of tests, and is touted as a fairly accurate method (albeit more computationally expensive than some other objects).

One major issue that was noted was, in the event of object occlusion or other loss of tracking, there was very poor recovery behavior. If this was to be in production, some other method of object detection would be required, or manual re-selection of the object upon failure.

4) Saving of Data: Upon conversion completion, any failures to track would be recorded as `NaN`, and the total number of these were reported. If the user felt they were acceptable, the data was then saved as `.npy` file for easy import.

B. SVD/PCA

1) SVD: Fortunately, per instructor permission, the SVD itself was able to be done using a pre-built implementation. Specifically, `numpy.linalg.svd`, which wraps `LAPACK`'s `_gesdd` routine, allowing for high numerical precision and speed.

The observation vectors were stacked before calculating the SVD, truncating to the length of the shortest for a given case.

2) Plotting and Exploration: Following the example set in Dr. Kutz's book, a series of four plots were made for each case:

1) Modal Energy
2) Modal Energy, on SemiLog axes
3) "Spatial" behavior of the 2 most significant modes
4) Temporal behavior of the 2 most significant modes

Additionally, reconstruction of the SVD with only N modes was performed to observe the effects of discarding lower energy modes.

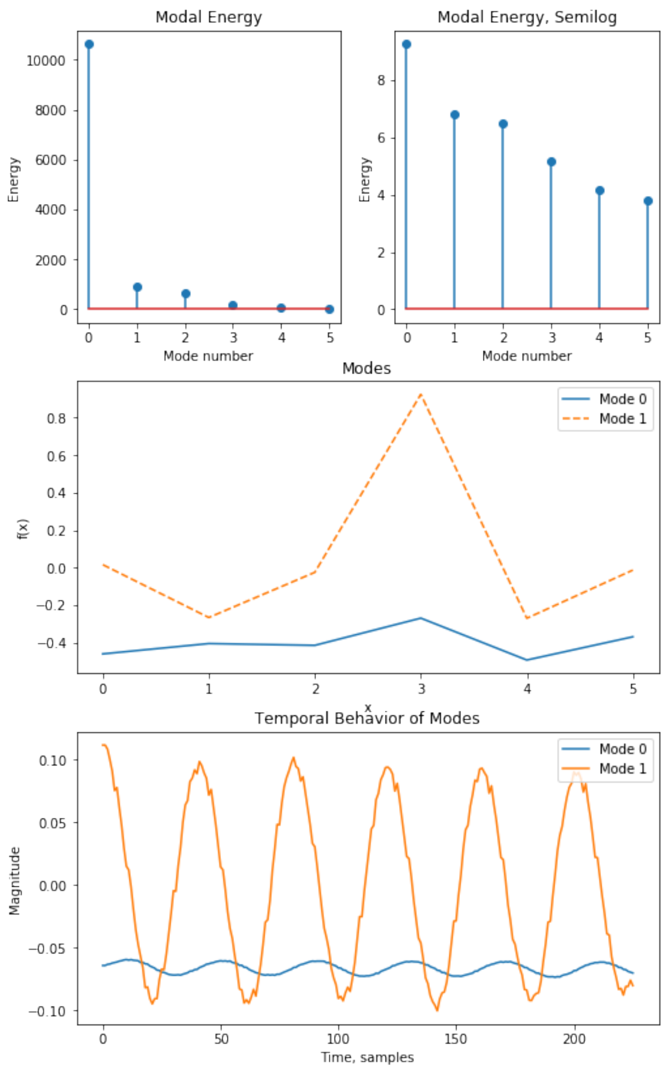

Fig. 1. Kutz Plot, Case 1

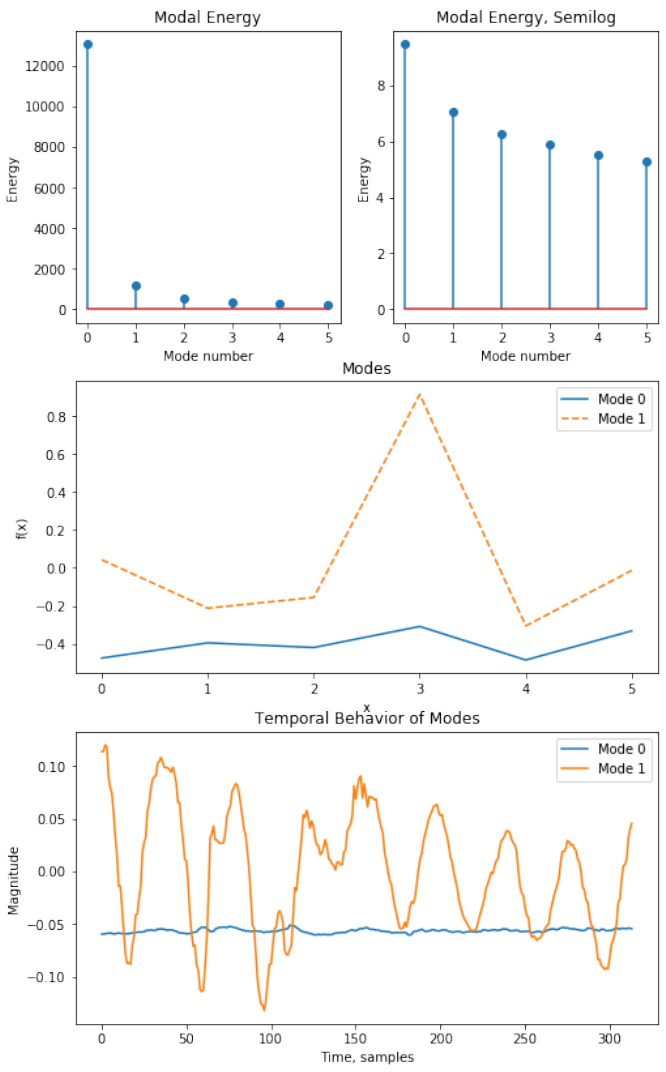

Fig. 2. Kutz Plot, Case 2

IV. Computational Results

A. Kutz Plots

The results as “Kutz Plots” (described in “Plotting and Exploration”) for each of the cases can be see in Figures 1, 2, 3, and 4.

The results are a bit confusing. The temporal data indicates that mode zero (the highest energy mode) demonstrates little change. My best guess is that mode 0 is some offset, since the data hasn’t been normalized to have a mean of zero. Counter to this, however, is that the highest energy mode is supposed to be the highest variance and thus best at capturing the movement of the paint can; a roughly constant offset has, by definition, roughly zero variance. Exacerbating the matter, the second mode appears to demonstrate temporal behavior that matches what is expected – strongly oscillatory movement.

The spatial data also presents a challenge to parse. Based upon the examples in Dr. Kutz’s book, I’m led to believe this is akin to the basis function for a transform, as evidenced by the example in figure 160. But, the resulting data doesn’t present such a clear function. My assumption is either that the underlying function is a sinusoid or a somthing like a dirac impulse, which when changed in time produces the oscillatory behavior of the system.

Encouragingly, both the first and second cases (Fig. 1, 2) produce VERY similar modal results while producing different temporal results. This is indicative of the ability of the method to ignore the noise introduced by the camera shake.

B. Reconstruction/Projection

Given that a primary use of PCA is to project the data into a lower subspace, this was explored in several key plots. For one, the effect of increasing the number of modes was demonstrated in Fig. 5. As expected, more modes show greater complexity of behavior of each time series. In this case, much beyond two or three modes doesn’t appear to improve or change the resulting plots.

Based upon this threshold, the remaining cases were reconstructed with 3 modes (Figs. 6, 7, 8).

V. Summary and Conclusion

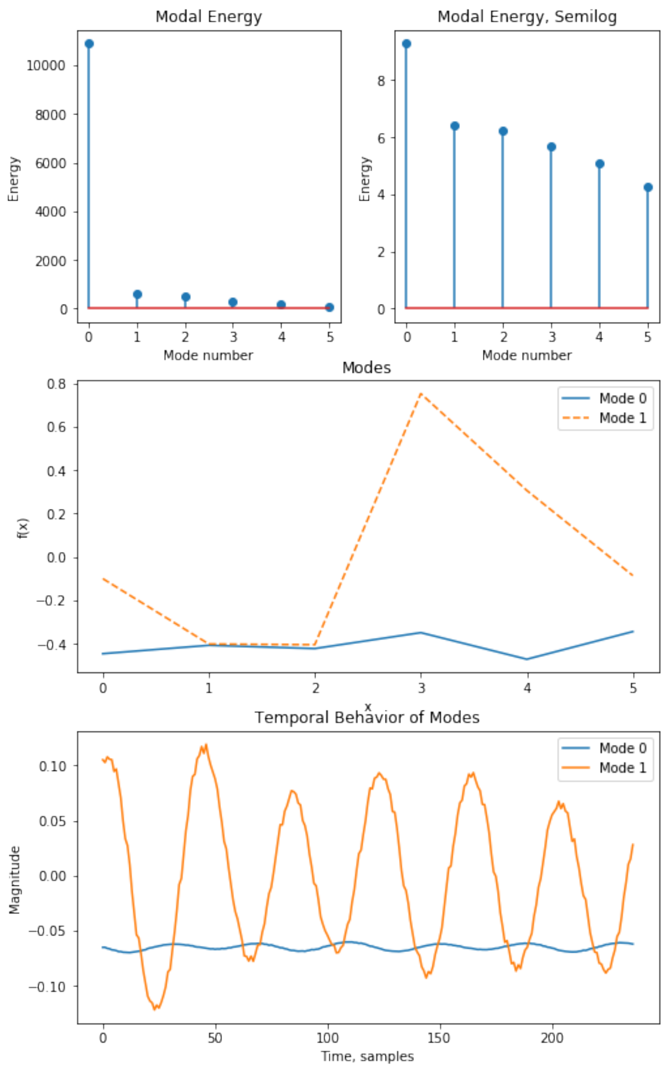

Fig. 3. Kutz Plot, Case 3

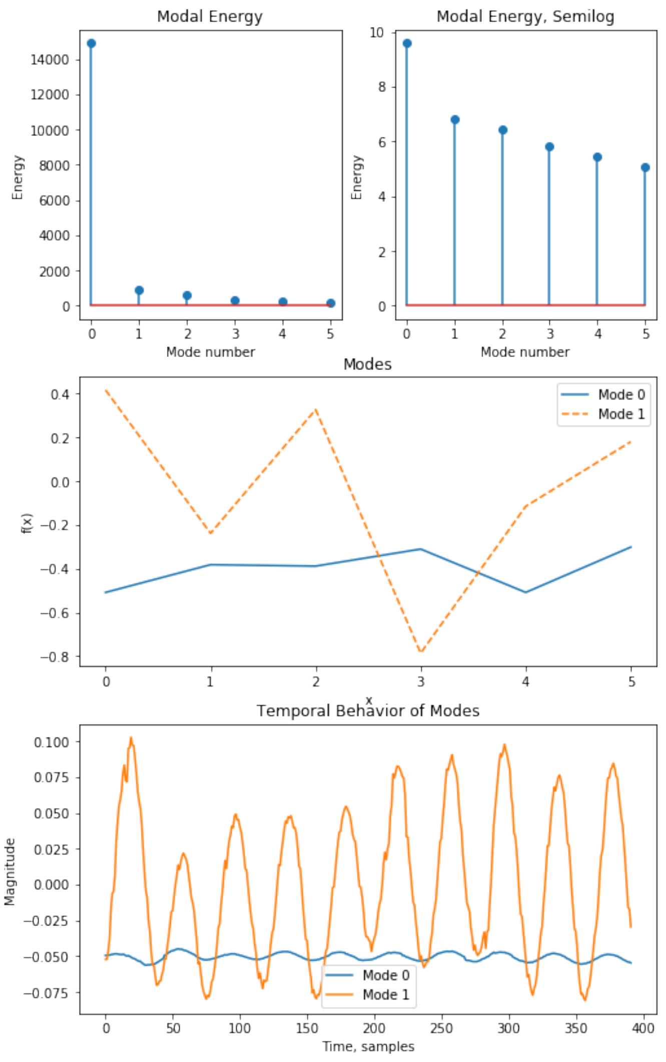

Fig. 4. Kutz Plot, Case 4

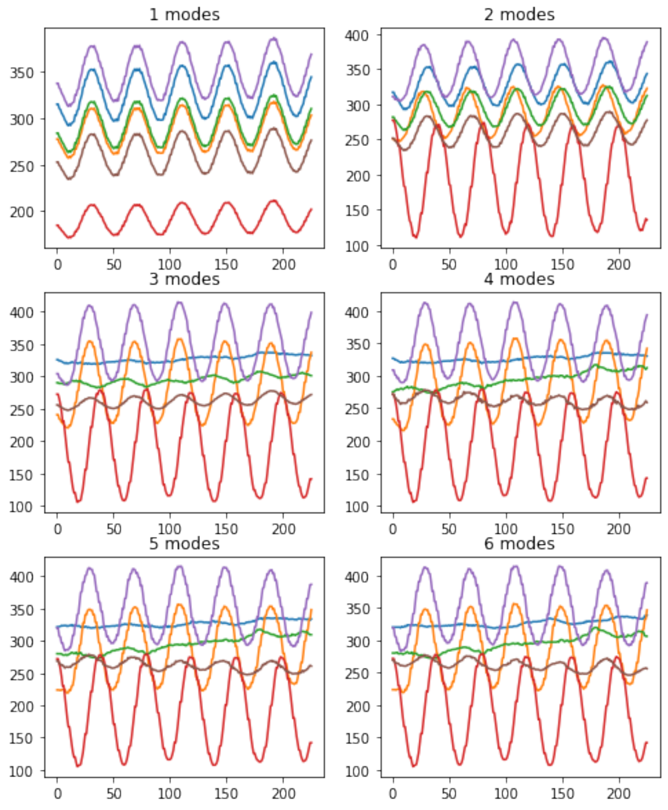

Fig. 5. Case 1 reconstructed with N modes

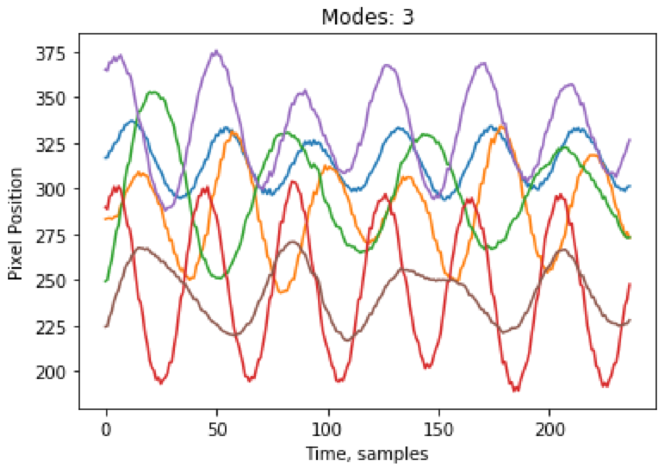

Fig. 7. Case 3, Modes=3

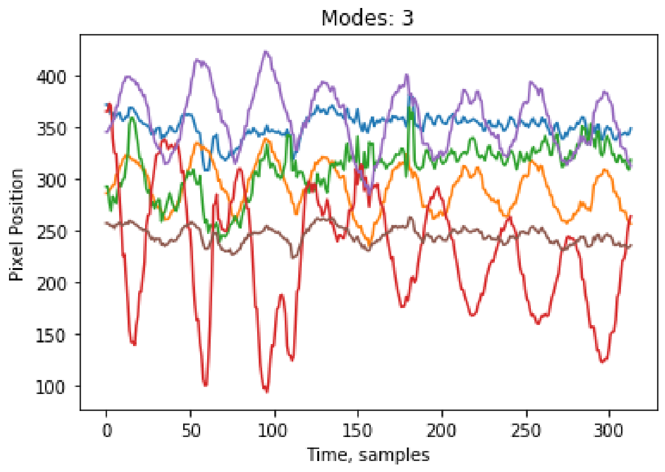

Fig. 6. Case 2, Modes=3

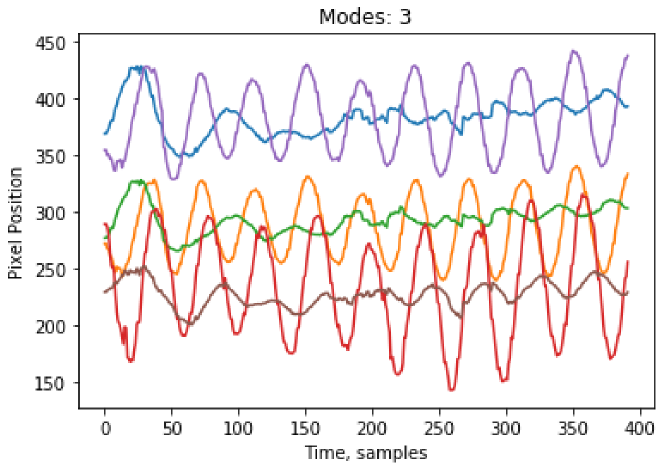

Fig. 8. Case 4, Modes=3

VI. Appendix A: Functions Used

A. `numpy.lib.stride_tricks.as_strided()`

Given a dataset (1D, in this case), a stride length, and a window width, return a “view” into the data. See here.

VII. Appendix B: Python Code

See my Github for the full repository, including this source code for this IEEE template LaTeXdocument. Code is also attached at the end of this report.