AMATH 582 Homework 3: PCA

Brady Griffith

February 24, 2021

Abstract

A mass on a spring is recorded with 3 cameras each at different views. This problem has a known physical behavior, so it's an excellent candidate to preform PCA on and compare the results. Four cases are considered: purely vertical motion, the same with camera shake, vertical and horizontal motion, and vertical and horizontal motion with rotation of the mass. The spring mass motion and pendulum motion are all identifiable in the principle components in the cases without camera shake and rotation. The camera renders the image almost completely useless while the rotation just mixes the modes.

Introduction and Overview

My undergraduate degree was in physics, so I have a particular fondness for the spring mass oscillator. This project takes a backwards look, as if the problem were novel physics to explain. Using videos taken at three different perspectives, I use principal component analysis to infer the physics of the mass/spring system. This particular problem has the advantage of a theoretical solution to compare results against.

In the simplest version of this problem, the spring is constrained to only move vertically, i.e. without pendulum motion. This is the problem solved in the first week of any mechanics class, and for the mass's vertical displacement from equilibrium $z(t)$,

$$z(t) = z_0 \sin(\omega_Z t - \phi_0) \tag{1}$$

where z_0 and ϕ_0 are the initial displacement and phase, and $\omega_Z^2 = \frac{k}{m}$ is defined as the resonant frequency for the spring constant k and mass m. When behaving in this limit, you would expect principal mode to be a sinusoid with that frequency.

A more complicated version of this problem displaces the mass laterally as well, inducing a pendulum motion. The exact dynamics of this system are complicated with precession of the pendulum motion and can involve coupling between modes [1]. Exact motion is not important to this analysis and only short time scales are considered. As long as the frequencies stay away from being in resonance, the

two can be treated separately. We would then expect second and third modes asscociated with the elliptical pendulum motion.

A similar problem is the example used in the textbook when explaining PCA [2]. The difference here being that instead of each perspective reporting a pendulum position, this projects looks at the unprocessed camera data. This probably isn't the best way to extract information about the motion, but does serve as a better exploration of PCA's limitations.

Theoretical Background

Principal component analysis in use for time series, as is the case in this problem, begins by collecting a vectors of each measurement's time series, $\boldsymbol{x_i}$, into rows of a single large matrix.

$$\mathbf{X} = \begin{bmatrix} \boldsymbol{x_0} \\ \boldsymbol{x_1} \\ \vdots \\ \boldsymbol{x_n} \end{bmatrix} \tag{2}$$

The covariance of between all measurements is given by the matrix elements of

$$\mathbf{C_X} = \frac{1}{n-1}\mathbf{X}\mathbf{X}^T \tag{3}$$

It would be informative to switch into a basis where the each row has no covariance with any other rows. This is the same as having a diagaonal covariance matrix. This can be done by diagonalizing $\mathbf{X}$. Suppose we use SVD to determine $\mathbf{U}$, $\mathbf{\Sigma}$, and $\mathbf{V}^*$. $X = \mathbf{U\Sigma V}^*$. If we define a transformed measurement matrix $\mathbf{Y} = \mathbf{U}^*\mathbf{X}$,

$$\begin{aligned} C_Y &= \frac{1}{n-1}\mathbf{Y}\mathbf{Y}^T \\ &\vdots \quad \text{See Kutz (2013) §15.4} \\ C_Y &= \frac{1}{n-1}\mathbf{\Sigma}^2 \end{aligned}$$

This means that the output of the SVD is the new basis without covariance.

Specifically applied to the problem at hand, the measurements in each row are the intensity of each pixel from each camera over the recording time. In the optimal case of purely vertical motion of the mass you would expect that the first component of Y would be the unchanging scene elements like the walls and floor. The second would be the sinusoidal motion of the mass and the rest would all be 0. The deviations from this are the subject of the discussion in the results section.

Algorithm Implementation and Development

The provided videos are in Matlab M-Files, which I load into numpy arrays using the scipy io.loadmat function. Preprocessing is needed before the video can be loaded into the X matrix. The videos start and end at different times, and this method requires them to synced. I manually determine the offset needed to align each segment, and then crop the longest segment all 3 videos contain.

The video files are large and long, which poses a computational limit to I will be able to process. SVD is preformed on the $\mathbf{X}^T\mathbf{X}$ which is a square matrix of width (# of measurements). If I kept the entire 640x480 frame with all 3 color channels, $\mathbf{X}^T\mathbf{X}$ would have 8×10^{12} elements. Using a double precision floating pint number, this matrix would need 61 TB of memory, and this is well beyond the 16 GB installed in my computer.

To make the computation tractable, I preform 3 simplifications to the data. First, I ignore color, averaging over the 3 channels to produce a monochrome image. Second, I crop the frames to the region that contains the pendulum. The bounds were manually determined for each clip. Both of these steps still yield to a matrix that is too large, so I lower the resolution by taking combining 3 pixel by 3 pixel groups.

The numpy linalg.svd function preforms the SVD on the measurement matrix. From the output the, the covariances of each component are plotted to show their contribution. I then plot the first six components.

Computational Results

With the procedure described in the last section, for the four tests, the principal components are found. The variance of the first 100 terms is plotted. Below that, the first 6 components are shown. I will attempt to give the physical interpretation of these when possible.

Test 1 is the ideal case. (See figure 1) The mass is only offset z direction. The oscillation small, so behavior should be ideal. The second largest component is periodic as would be expected. The period is around 45 frames. which at a 30 FPS would be a 1.5 s period. The 4th and 5th principal component display a periodicity that would be consistent with the 2rd and 3rd harmonic.

Test 2 is the noisy case. (See figure 2) Camera shake was introduced. The results suffer because of this. It is not possible to find the harmonic motion over the whole time series. For the first 100 frames, some of the motion can be seen in the second component, but gets lost as the motion blur gets worse.

Test 3 is released with horizontal displacement. (See figure 3) The 3rd component corresponds to the spring oscillation motion and has the same 45 frame period. The 4th component is periodic with a longer period. This is the pendulum motion.

Test 4 is also released with horizontal displacement, but now also with rotation. (See figure 4) This rotation acts to confuse the PCA. The fundamental spring mass mode gets lots, although the 2nd and 3rd harmonics are still found as the 2nd and 5th component. The pendulum motion is also lost. The 1st and 3rd components appear to be blends of both modes.

Summary and Conclusions

PCA was preformed on four different cases of a spring mass system recorded with cameras in different perspectives. The cases where the camera remains fixed and the mass moves without rotating have quality representation in the principal components. Camera shake rendered the data almost useless, as no part of the mass motion could be identified. Rotation of the mass confused the results but not so badly as to obscure all the physics.

There are several additions that could improve the results of this particular problem. First, I could have used a wavelet transform to identify edges or compress the data before applying PCA. Instead of using monochrome images, I could have used a channel that maximized the mass's contrast against the wall. If data could be recollected, longer videos would also improve identifying components of the spring mass system.

References

[1] P. Lynch, "Resonant motions of the three-dimensional elastic pendulum," *International Journal of Non-Linear Mechanics*, vol. 37, pp. 345–367, Mar. 2002.

[2] J. N. Kutz, *Data-driven modeling & scientific computation: methods for complex systems & big data*. Oxford: Oxford University Press, first edition ed., 2013. OCLC: ocn858608087.

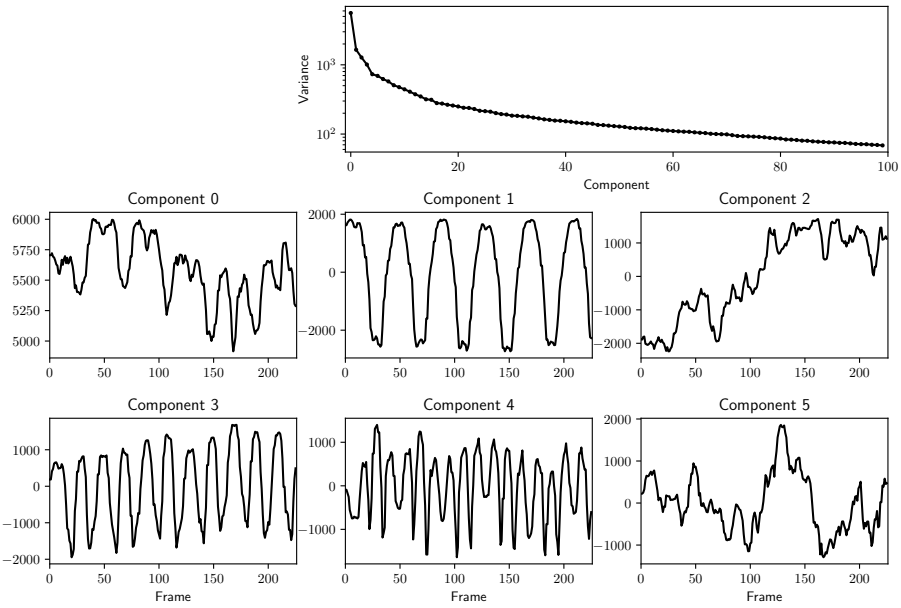

Figure 1: Test 1. Ideal cade. The covariance of the first 100 principal components along with the first 6 components.

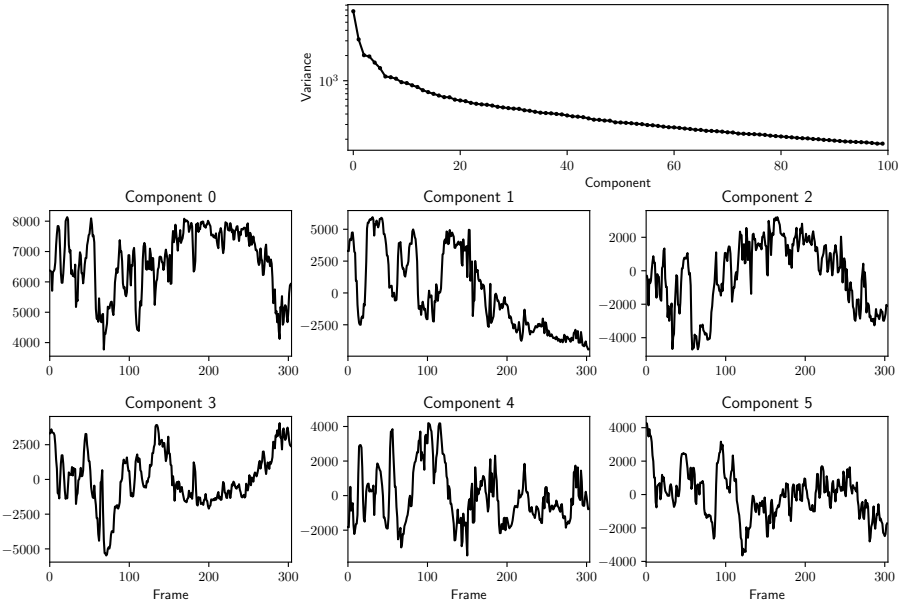

Figure 2: Test 2. Noisy cade. The covariance of the first 100 principal components along with the first 6 components.

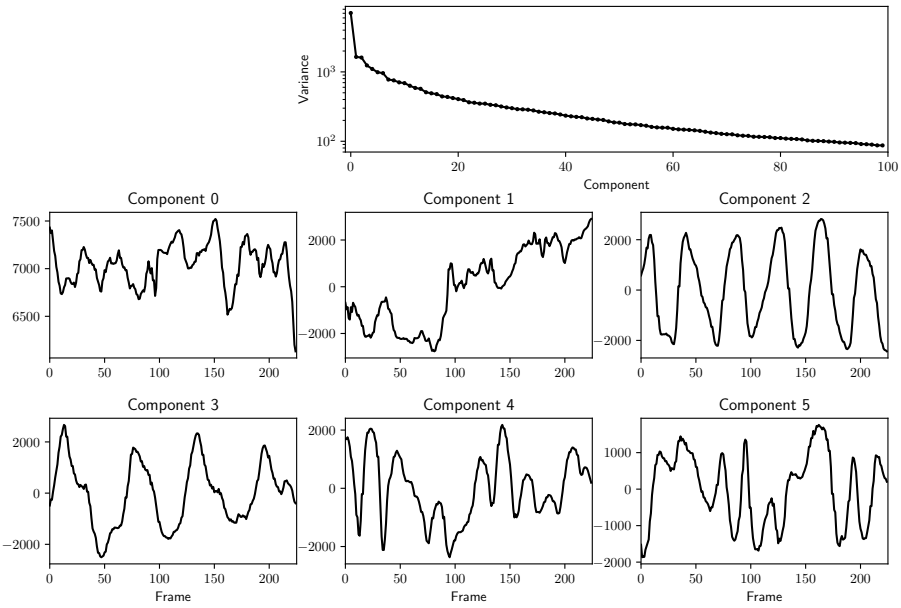

Figure 3: Test 3. Horizontal displacement. The covariance of the first 100 principal components along with the first 6 components.

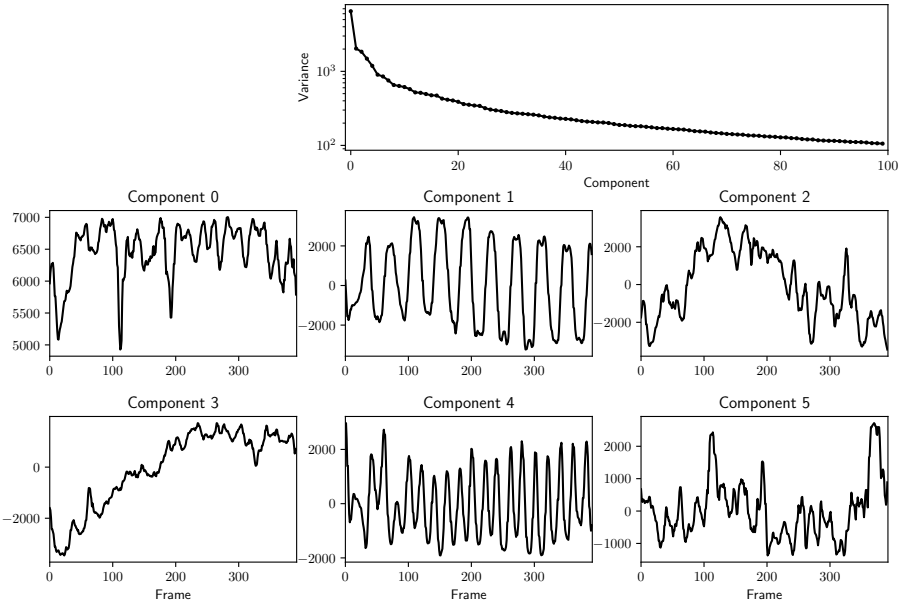

Figure 4: Test 4. Horizontal displacement and rotation. The covariance of the first 100 principal components along with the first 6 components.

Here is a link to the Github repository for this project.

Python Functions

Video Objects

```
class Video()
```

Parameters of to load a single video matrix.

This is to load the video form the matlab matrix file and then crop it appropriately.

Attributes:

- `filename` *str* - The path to the matlab matrix file for the video.
- `start` *int* - The frame to start loading from.
- `end` *int* - Load frames before this number.
- `left` *int* - The left edge to crop from.
- `right` *int* - The right edge to crop from.
- `top` *int* - The top edge to crop from.
- `bottom` *int* - The bottom edge to crop from.

___init___

```
| __init__(filename, start, end, left, right, top, bottom)
```

Initialize the Video class.

args: filename (str): The path to the matlab matrix file for the video. start (int): The frame to start loading from. end (int): Load frames before this number. left (int): The left edge to crop from. right (int): The right edge to crop from. top (int): The top edge to crop from. bottom (int): The bottom edge to crop from.

read

```
| read()
```

Retrieve the matrix of the video, cropped as specified.

Returns:

- `ndarray` - The matrix of the video. The shape is (vertical pixels, horizontal pixels, frames).

from_text

```
| @staticmethod
| from_text(text)
```

Creates a video class from the line of a CSV

Arguments:

- `text` *str* - Line from a csv "path, start, end, left, right, top, bottom"

Returns:

- `Video` - A `Video` class for the line provided.

make_vid_list

`make_vid_list()`

Create lists of `Video` for all 4 tests.txt

Creates the videos with the properties defined in the vid_props files.

Returns:

- `list` - List of each test's list of `Video` objects.

read_meas_matrix

`read_meas_matrix(vid_list)`

Loads the matrix with rows being the separate time measurements.

This is the X matrix expected for PCA.

Arguments:

- `vid_list` *list* - A list of `Video` class objects to load the measurements from. All of the should be the same length.

Returns:

- `ndarray` - The measurements matrix X for PCA. Shape (Number of Pixels, Number of Frames)

pca

`pca(M)`

Preforms PCA on the matrix provided.

Arguments:

- `M` *array-like* - The matrix to preform PCA on.

Returns:

- `ndarray` - The variances of the principal components.
- `ndarray` - U_T matrix to project M into principal components.

plot_dominant_mode

`plot_dominant_mode(s, U_T, X, fig_path)`

Plots the variances and the fist 4 PCA modes.

Arguments:

- `s` *array-like* - 1D arrray of the variances of the principal components.
- `U_T` *array-like* - The matrix to transform X into the principal components.
- `X` *array-like* - The measurement matrix
- `fig_path` *str* - The path to save the plot.
- `comps` *int* - Number of modes to plot

Python Code

pca.py

```
import numpy as np
import matplotlib.pyplot as plt
from scipy import io


class Video:
    """Parameters of to load a single video matrix.

    This is to load the video form the matlab matrix file and then crop it
    appropriately.

    Attributes:
        filename (str): The path to the matlab matrix file for the video.
        start (int): The frame to start loading from.
        end (int): Load frames before this number.
        left (int): The left edge to crop from.
        right (int): The right edge to crop from.
        top (int): The top edge to crop from.
        bottom (int): The bottom edge to crop from.
    """
    def __init__(self, filename, start, end, left, right, top, bottom):
        """Initialize the Video class.

        args:
            filename (str): The path to the matlab matrix file for the video.
            start (int): The frame to start loading from.
            end (int): Load frames before this number.
            left (int): The left edge to crop from.
            right (int): The right edge to crop from.
```

```
            top (int): The top edge to crop from.
            bottom (int): The bottom edge to crop from.
        """
        self.filename = filename

        if start > end:
            raise ValueError('`start` must be lower than `end`')
        self.start = start
        self.end = end

        if left > right:
            raise ValueError('`left` must be lower than `right`')
        self.left = left
        self.right = right

        if top > bottom:
            raise ValueError('`top` must be lower than `bottom`')
        self.top = top
        self.bottom = bottom

    def read(self):
        """Retrieve the matrix of the video, cropped as specified.

        Returns:
            ndarray: The matrix of the video. The shape is (vertical pixels,
                horizontal pixels, frames).
        """
        ml_file = io.loadmat(self.filename)

        # The file should only contain one matrix. If not, this will break
        key = list(ml_file.keys())[-1]

        ds = 3  # Down sample by this factor
        M = ml_file[key][self.top:self.bottom,  # Crop Horizontal
                         self.left:self.right,  # Crop Vertical
                         :,                     # Leave all color
                         self.start:self.end]   # Crop to start

        # Throw out anything after the last down sampled point on the
        # right and bottom
        M = M[:ds*(M.shape[0]//ds), :ds*(M.shape[1]//ds), :, :]

        # Average over the down sampled region and over all color channels
        M = M.reshape(M.shape[0]//ds,
                      ds,
                      M.shape[1]//ds,
```

```
                    ds, M.shape[-2],
                    M.shape[-1]).mean(3).mean(1).mean(2)

        return M

    @staticmethod
    def from_text(text):
        """Creates a video class from the line of a CSV

        Args:
            text (str): Line from a csv "path, start, end, left, right, top,
                bottom"

        Returns:
            Video: A `Video` class for the line provided.
        """
        elements = text.strip().split(',')

        if len(elements) != 7:
            raise ValueError('Line provided cannot form Video object')

        return Video(elements[0].strip('"'),
                     *(int(x) for x in elements[1:]))


def make_vid_list():
    """Create lists of `Video` for all 4 tests.txt

    Creates the videos with the properties defined in the vid_props files.

    Returns:
        list: List of each test's list of `Video` objects.
    """
    tests = []

    for i in range(1, 5):
        with open(f'data/vid_props/test{i}.csv') as f:
            test = [Video.from_text(l) for l in f.readlines()[1:]]
        tests.append(test)

    return tests


def read_meas_matrix(vid_list):
    """Loads the matrix with rows being the separate time measurements.
```

```
    This is the X matrix expected for PCA.

    Args:
        vid_list (list): A list of `Video` class objects to load the
            measurements from. All of the should be the same length.

    Returns:
        ndarray: The measurements matrix X for PCA. Shape (Number of Pixels,
        Number of Frames)
    """
    X = None
    for vid in vid_list:
        A = vid.read()
        B = A.reshape((-1, A.shape[-1]))
        B -= np.mean(B)
        X = B if X is None else np.append(X, B, axis=0)
    return X


def pca(M):
    """Preforms PCA on the matrix provided.

    Args:
        M (array-like): The matrix to preform PCA on.

    Returns:
        ndarray: The variances of the principal components.
        ndarray: U_T matrix to project M into principal components.

    """
    C = M / np.sqrt(np.shape(M)[-1] - 1)  # Normalize
    U, s, _ = np.linalg.svd(C)
    return s**2, U.T


def plot_dominant_mode(s, U_T, X, fig_path):
    """Plots the variances and the fist 4 PCA modes.

    Args:
        s (array-like): 1D arrray of the variances of the principal components.
        U_T (array-like): The matrix to transform X into the principal
            components.
        X (array-like): The measurement matrix
        fig_path (str): The path to save the plot.
        comps (int): Number of modes to plot
    """
```

```
    plt.rcParams.update({"text.usetex": True})
    fig = plt.figure(figsize=(9, 6))

    ax_g = fig.add_subplot(3, 3, (2, 3))

    ax_g.set_yscale('log')

    ax_g.set_xlabel('Component')
    ax_g.set_ylabel('Variance')

    ax_g.set_xlim(-1, 100)

    ax_g.plot(s[:100], c='k', marker='o', markersize=2)

    for i in range(6):
        ax = fig.add_subplot(3, 3, 4+i)
        ax.set_title(f'Component {i}')

        if i > 2:
            ax.set_xlabel('Frame')

        ax.set_xlim(0, X.shape[-1])
        ax.plot(np.matmul(U_T[i], X), c='k')

    fig.tight_layout(pad=0.05)
    # plt.show()
    fig.savefig(fig_path, bbox_inches='tight')


if __name__ == '__main__':
    for i, vid_list in enumerate(make_vid_list()):
        print(f'Loading test {i+1}')
        X = read_meas_matrix(vid_list)
        g, U_T = pca(X)

        # Save and load the PCA so that the computation
        # can be skipped when replotting
        np.save(f'output/g_{i+1}.npy', g)
        np.save(f'output/S_T_{i+1}.npy', U_T)

        g = np.load(f'output/g_{i+1}.npy')

        U_T = np.load(f'output/S_T_{i+1}.npy')
        plot_dominant_mode(g, U_T, X, f'figures/PCA_{i+1}.pdf')
```