Principal Component Analysis:
A Picture Paints a Thousand Data Points

Tim Ferrin

February 21, 2020

Abstract

Given are four different scenarios of a paint can bouncing up and down on a spring, with three cameras at different angles recording each scenario. First the can simply oscillates up and down, then this is repeated with noise introduced in the form of camera shake, then the can is released to the side and down to produce vertical oscillation and pendulum motion, and finally the can is pulled to the side and down and is given a spin to produce vertical oscillation, pendulum motion, and rotation about its internal axis. The goal of this report is to use Principal Component Analysis to analyze the motion of the paint can, disregarding prior knowledge of physical systems.

I Introduction and Overview

Recorded prior to the analysis of this report, four different scenarios of a paint can bouncing up and down on a spring are captured using three cameras from different angles. The four scenarios are as follows:

1. Simple, vertical harmonic motion
2. Simple, vertical harmonic motion with the addition of noise in the form of camera shake
3. Simple, vertical harmonic motion and pendulum motion
4. Simple, vertical harmonic motion, pendulum motion, and internal rotation

Figure 1 shows an image taken from a camera recording the fourth scenario.

Using Principal Component Analysis (PCA), as described in the Theoretical Background section, the motion of the paint can is observed. This is particularly interesting, as we don't know how many cameras are required to fully capture the motion of the paint can. (This is assuming we know nothing about physics and differential equations.) PCA allows us to determine what combination of the x and y data captured from each of the cameras is necessary and how to use that data to form a complete picture of the observed motion.

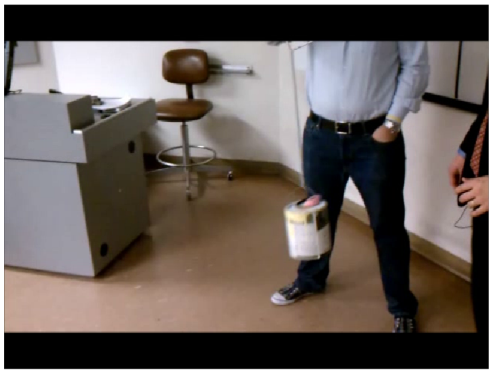

Figure 1: Camera 1 of the fourth scenario involving vertical harmonic motion, pendulum motion, and internal rotation.

II Theoretical Background

In order to discuss PCA, we must first talk about the Singular Value Decomposition (SVD) of a matrix. For any general matrix $A \in \mathbb{R}^{m \times n}$,

$$A = \begin{bmatrix} A_{11} & A_{12} & ... & A_{1n} \\ A_{21} & A_{22} & ... & A_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ A_{m1} & A_{m2} & ... & A_{mn} \end{bmatrix} \tag{1}$$

A can be thought of as a series of transformations, and this can be visualized when thinking how A transforms a collection of n n-dimensional orthogonal vectors on an n-dimensional hypersphere. If we call these vectors $\vec{v}_1$, $\vec{v}_2$, ... , $\vec{v}_n$. We define these vectors to be the specific vectors such that when these vectors are multiplied by A, they transform into the semi-axes of an n-dimensional hyperellipse in $\mathbb{R}^m$. We will call these new, transformed vectors $\sigma_1\vec{u}_1$, $\sigma_2\vec{u}_2$, ... , $\sigma_n\vec{u}_n$, where $\sigma_1, \sigma_2, ..., \sigma_n$ are scalars that make $\vec{u}_1$, $\vec{u}_2$, ... , $\vec{u}_n$ normal. Thus in matrix form the above process can be represented as

$$\begin{bmatrix} A_{11} & A_{12} & ... & A_{1n} \\ A_{21} & A_{22} & ... & A_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ A_{m1} & A_{m2} & ... & A_{mn} \end{bmatrix} \begin{bmatrix} | & | & & | \\ \vec{v}_1 & \vec{v}_2 & ... & \vec{v}_n \\ | & | & & | \end{bmatrix} = \begin{bmatrix} | & | & & | \\ \vec{u}_1 & \vec{u}_2 & ... & \vec{u}_n \\ | & | & & | \end{bmatrix} \begin{bmatrix} \sigma_1 & 0 & ... & 0 \\ 0 & \sigma_2 & ... & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & ... & \sigma_n \end{bmatrix} \tag{2}$$

Or more compactly,

$$AV = \hat{U}\hat{\Sigma} \tag{3}$$

Since we defined the vectors of V to be orthonormal, V is unitary and thus

$$AVV^* = \hat{U}\hat{\Sigma}V^* \quad \implies \quad A = \hat{U}\hat{\Sigma}V^T \qquad \text{if A is real, and} \tag{4}$$

$$AVV^* = \hat{U}\hat{\Sigma}V^* \qquad \Longrightarrow \qquad A = \hat{U}\hat{\Sigma}V^* \qquad \text{if A is complex} \tag{5}$$

If A is full rank, then $\hat{\Sigma}$ is an $n \times n$ positive, diagonal matrix. Such a decomposition of matrix A is called the reduced SVD. The full SVD is obtained by augmenting $\hat{U}$ with an additional $m - n$ columns that are orthonormal and orthogonal to the pre-existing columns of $\hat{U}$. This makes U an $m \times m$ unitary matrix and requires an additional $m - n$ rows of zeros concatenated on the bottom of the $\hat{\Sigma}$ matrix. If A is not full rank (say it is rank r), this simply means that there are simply $m - r$ instead of $m - n$ 0 rows and columns. So now the new matrix Σ has r positive diagonal entries, with the remaining $n - r$ being 0. The full SVD is then given by

$$A = \begin{bmatrix} | & | & & | & & | \\ \vec{u}_1 & \vec{u}_2 & \ldots & \vec{u}_n & \ldots & \vec{u}_m \\ | & | & & | & & | \end{bmatrix} \begin{bmatrix} \sigma_1 & 0 & \ldots & 0 \\ 0 & \sigma_2 & \ldots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \ldots & \sigma_n \\ 0 & 0 & \ldots & 0 \\ \vdots & \vdots & & \vdots \\ 0 & 0 & \ldots & 0 \end{bmatrix} \begin{bmatrix} — & \vec{v}_1 & — \\ — & \vec{v}_2 & — \\ & \vdots & \\ — & \vec{v}_n & — \end{bmatrix} \tag{6}$$

Or more compactly

$$A = U\Sigma V^* \tag{7}$$

Now for some general information about the SVD:

1. The values of σ_j are called the singular values of A
2. The vectors $\vec{u}_j$ are called the left singular vectors of A
3. The vectors $\vec{v}_j$ are called the right singular vectors of A
4. The singular values are always non-negative
5. By convention, the singular values are ordered from largest to smallest
6. The number of nonzero singular values is the rank of A
7. $\vec{u}_1$, $\vec{u}_2$, ... , $\vec{u}_r$ forms a basis for the range of A and $\vec{u}_{r+1}$, ... , $\vec{u}_n$ forms a basis for the nullspace of A
8. Every matrix has an SVD

The last point is an important distinction to make, in comparison to an eigendecomposition. If the reader will recall, the compact form of the full SVD above appears symbolically similar to the eigendecomposition, and indeed the two are related. With some manipulation, it can be shown that the singular values of A are the square roots of the eigenvalues of A^TA, the right singular vectors of A are the eigenvectors of A^TA, and the left singular vectors of A are the eigenvectors of AA^T. This leads to a potential method of computation of the SVD.

Having covered the SVD, we can now move on to the concept of principal component analysis. The covariance of two vectors $\vec{a}$ and $\vec{b}$ of length n gives a value to how related the two vectors are to each other and is defined to be

$$\sigma_{ab}^2 = \frac{1}{n-1}\vec{a}\vec{b}^T = \frac{1}{n-1}\vec{b}\vec{a}^T \tag{8}$$

If $\vec{a} = \vec{b}$, then this quantity is called the variance of $\vec{a}$. Given a data matrix X whose rows represent variables about the same subject and whose columns represent different snapshots in time or different subjects, its covariance matrix is computed with the following formula

$$X = \begin{bmatrix} — & \vec{a} & — \\ — & \vec{b} & — \\ — & \vec{c} & — \\ — & \vec{d} & — \end{bmatrix} \qquad C_X = \frac{1}{n-1}XX^T = \begin{bmatrix} \sigma_a^2 & \sigma_{ab}^2 & \sigma_{ac}^2 & \sigma_{ad}^2 \\ \sigma_{ba}^2 & \sigma_b^2 & \sigma_{bc}^2 & \sigma_{bd}^2 \\ \sigma_{ca}^2 & \sigma_{cb}^2 & \sigma_c^2 & \sigma_{cd}^2 \\ \sigma_{da}^2 & \sigma_{db}^2 & \sigma_{dc}^2 & \sigma_d^2 \end{bmatrix} \tag{9}$$

Diagonalizing this covariance matrix will give you the principal components (column vectors of U) and singular values of the data matrix, as described above. Finally, you can determine which of the principal components most contribute to the formation of the data matrix, ie what weighting of the variables is actually statistically significant. The relative importance of a principal component or group of principal components is given by the energy of its corresponding singular value(s) with the following formula. Taking the first N principal components and zeroing out all the rest gives you the best rank-N approximation of the original data matrix.

$$E_N = \frac{\sigma_1^2 + \sigma_2^2 + ... + \sigma_N^2}{\sigma_1^2 + \sigma_2^2 + ... + \sigma_r^2} \tag{10}$$

III Algorithm Implementation and Development

We are given 3 camera videos for each scenario described in the the introduction section. First we will need to collect data on the location of a single point on the paint can and track it throughout each videos' frames. The code in Appendix B contains a function script called "getData" that does exactly this. Essentially, it takes in the frames of the video, uses a discrete wavelet transform (similar to the Fourier Transform) of the grayscale image to detect vertical and/or horizontal edges. It then looks in the window of x and y values given by the bound parameters for groupings of pixels that are lit up in the shape specified by the checks parameter. In the case of the first scenario, the bottom rim of the paint can was used. If there is a match of the shape within the given window, it will then store the position of the origin defined by the check points in a result vector. It does this for each frame of the video and returns the position and frame data. For the scenarios where the paint can is swinging, a specified rectangle is analyzed for a certain percentage of the pixels being activated, as the signature of the paint can. The upper-left corner of this rectangle is used to to determine position data.

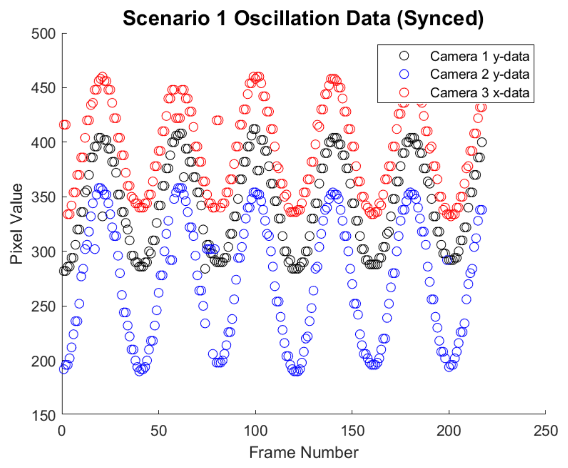

Figure 2: The synced up pixel data of the bottom left corner of the paint can from scenario 1 is shown. Each camera shows similar behavior.

After data collection, any missing frame position information due to bluriness or otherwise failure of the algorithm is filled in manually to complete the picture. In the case of scenario 2, this data is is also obtained for an easily identifiable anchor point, ie the v shape formed by the meeting of the man's legs. The shifting in this anchor point for each frame is then subtracted from the paint can position data in order to denoise it. Finally, as the videos are not of the same length, the position data from each camera is cropped in length and shifted so that the cycles of the paint can match up, as in Figure 2.

The x and y data of the paint can from each recording is then layed out in the rows of our data matrix, as described in the Theoretical Background section. The mean of each row vector is then subtracted from its respective row and PCA is done on the matrix using the MATLAB svd() command.

IV Computational Results

As a note, scenario 1 has and will be the main plotting data used, as it is more difficult to visualize and interpret the other scenarios. As described in the Theoretical Background section, an approximation's energy is computed to determine that principal components statistical significance. In the case of the first scenario, the log of the different energy weights are shown in Figure 3. As determined by the energy calculation, 99.4% of the energy of the data matrix for scenario one is captured by the first singular value. A similar value of 97.6% is obtained for the first singular value for scenario two. For scenario 3, the second singular value energy is still quite high, relative to the the other singular values, and an energy level of 98.5% is obtained when keeping the first two singular values. Finally for scenario 4, the first 3 singular values all contain significant energy (together a computed 96.3%), and thus are statistically significant.

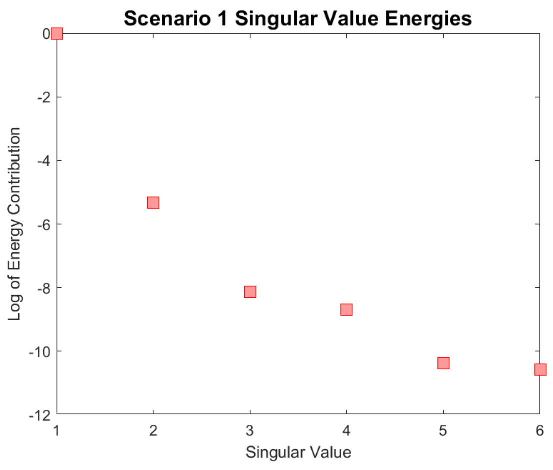

Figure 3: Log of the individual energies of each of the singular values from Scenario 1. Drop off after value one is noticeably large.

V Summary and Conclusions

We started off not knowing anything about the motion of the paint can. Through edge detection with the discrete wavelet transform, we were able to detect the changes in position of a paint can over relatively large periods of time. PCA has informed us that only one principal component is required to describe the motion of scenarios 1 and 2, that 2 principal components are required for scenario 3, and 3 principal components for scenario 4. This essentially tells us the number of degrees of freedom of the motion of the paint can in each scenario, which matches up with what our intuition/experience tells us. Although we cannot plot in or interpret six dimensions, we still have taken raw video footage and extracted useful information of the redundancy of our recordings. This analysis is extremely powerful and has much greater applications than this simple example.

Appendix A: MATLAB Functions Used and Brief Implementation Explanation

svd(data,'econ') – Used to decompose the data matrix into its SVD matrices.

dwt2(data,'haar') – Uses a Haar wavelet basis to extract high frequency data from a picture and effectively detect horizontal and vertcal edges.

rgb2gray(image) – Converts RGB images to gray images so that they are easier to work with.

imshow(image)/implay(video) – Displays/plays the given image/video

Appendix B: MATLAB Codes

Main Code

```matlab
load('cam1_1.mat')
load('cam2_1.mat')
load('cam3_1.mat')
implay(vidFrames1_1)
implay(vidFrames2_1)
implay(vidFrames3_1)
% Can do this for the other cameras/scenarios, not included in order to
% keep code concise

checkPoints1 = [5 2;6 3;7 4;7 5;7 6;7 7;7 8;8 9;8 10;8 11;
                8 12;8 13; 8 14;8 15];
positions1 = getData(vidFrames1_1,checkPoints1,308,336,280,424,0.01,0.94,0);
checkPoints2 = [0 0;1 1;2 2;3 3;3 4;3 5;3 6;3 7;3 8;3 9;
                3 10;3 11;3 12;3 13;3 14;3 15;3 16;
                3 17;3 18;3 19;3 20;3 21;3 22;3 23;
                3 24;3 25;2 26;1 27;0 28];
positions2 = getData(vidFrames2_1,checkPoints2,220,278,180,364,0.01,.87,0);

checkPoints3 = [-1 -2;0 -1;0 0;0 1;0 2;0 3;0 4;0 5;-1 6;
                -2 7;-3 7;-4 7;-5 7;-6 7;-7 7;-8 7;-9 7;
                -10 7;-11 7;-12 7;-13 7;-14 8;-15 8;-16 8;
                -17 8;-18 8;-19 8];
positions3 = getData(vidFrames3_1,checkPoints3,324,470,298,350,0.01,.96,2);

positions2 = [positions2;11 265 246;12 265 246;29 265 277;39 265 358;109 265 274;
                172 265 242;177 265 202;178 265 198;179 265 196;180 265 196;
                181 265 196;182 265 198;187 265 239;188 265 252;189 265 267;
                228 265 256;252 265 261;259 265 208;260 265 209;261 265 209];
[~,I]=sort(positions2(:,1));
positions2 = positions2(I,:);
for i=1:284
    if positions2(i,2) < 255
```

```
        positions2(i,2) = 265;
    end
end

%%
close all
cropBeg1 = 9;
cropped1 = positions1(cropBeg1+1:226,:);
cropBeg2 = cropBeg1+10;
cropEnd2 = 217 + cropBeg2;
cropped2 = positions2(cropBeg2+1:cropEnd2,:);
cropBeg3 = 9;
cropEnd3 = 217 + cropBeg3;
cropped3 = positions3(cropBeg3+1:cropEnd3,:);

figure(1)
hold on
plot(cropped1(:,1)-cropBeg1,cropped1(:,3),'ko')
plot(cropped2(:,1)-cropBeg2,cropped2(:,3),'bo')
plot(cropped3(:,1)-cropBeg3,cropped3(:,2),'ro')
xlabel('Frame Number')
ylabel('Pixel Value')
title('Scenario 1 Oscillation Data (Synced)','Fontsize',14)
legend('Camera 1 y-data','Camera 2 y-data','Camera 3 x-data')

print(gcf,'SyncedOsc.png','-dpng')

%% PCA
A = [cropped1(:,2:3) cropped2(:,2:3) cropped3(:,2:3)];
A = A-(sum(A,2)/217);
A = A.';
[U,S,V] = svd(A,'econ');
sVals = diag(S);
for i = 1:6
    sEnergy(i) = (sVals(i)^2)/sum(sVals.^2);
end
figure(2)
plot(1:6,(log(sEnergy)),'ks','MarkerSize',10,'MarkerEdgeColor','
    red','MarkerFaceColor',[1 .6 .6])
xlabel('Singular Value')
ylabel('Log of Energy Contribution')
title('Scenario 1 Singular Value Energies','Fontsize',14)

print(gcf,'Energies.png','-dpng')
```

getData function

```
function result = getData(video, checks, left, right, ceiling,
    floor, thresh1, thresh2, vhorb)

% checks is the halved y,x shape points with the NW corner as the
    origin
% left, right, and ceiling, floor are unhalved bound values
% vhorb ==0 means horizontal edge detection, ==1 means vertical,
    ==2 means both

[m,n,~,numFrames]=size(video);
positions = [];
lbound = left/2;
rbound = right/2;
ubound = ceiling/2;
bbound = floor/2;
for i=1:numFrames
    I1=rgb2gray(video(:,:,:,i));
    if vhorb == 0
        [~,cstar,~,~] = dwt2(im2double(I1),'haar');
    end
    if vhorb == 1
        [~,~,cstar,~] = dwt2(im2double(I1),'haar');
    end
    if vhorb == 2
        [~,cH,cV,~] = dwt2(im2double(I1),'haar');
        cstar = cH+cV;
    end
    points = [];
    for j=lbound:rbound % j is the halved x value
        for k=ubound:bbound % k is the halved y value
            checkPoints = [checks(:,1)+k checks(:,2)+j];
            compare = 0;
            numPoints = (size(checkPoints(:))/2);
            for p=1:numPoints
                if cstar(checkPoints(p,1),checkPoints(p,2)) >
                    thresh1
                    compare = compare + 1;
                end
            end
            if compare > numPoints*thresh2
                points = [points; j k];
            end
        end
    end
    if sum(size(points))~=0
        minX = min(points(:,1));
        minY = points(find(points(:,1)==minX,1),2);
        positions = [positions; i 2*minX 2*minY];
    end
end

result = positions;
end
```