Principal Component Analysis on Moving Objects

Quinn Luo

February 2019

Abstract

This report shows an application of principal component analysis; this example employs data mining to infer the dimensions of movement. Specifically, I will analyze the dimension of motion from video files shot from different angels. First, I will begin with theoretical background, followed by a detailed description of algorithm implementation. Finally, I will display the results from my analysis and validate them.

Introduction

When analyzing a system, whether it's mechanical, electrical, or biological, scientists often do not have information on its mechanism. Questions such as "what are the determining factors?" or "which two values are related?" are raised but not answered because governing equations are unknown. However, thanks to modern data science and machine learning, we are able to perform data-driven modeling on the system. And we should see the results of analysis as the indication of the underlying dimensions.

Theoretical Background

Singular Value Decomposition(SVD)

A matrix in $\mathbb{C}^{m\times n}$ can be decomposed in the following way:

$$A = U\Sigma V^*$$

where $U \in \mathbb{C}^{m\times n}$, $V \in \mathbb{R}^{n\times n}$, and $V^* \in \mathbb{C}^{n\times n}$.

This decomposition comes from the idea of seeing matrix operation as a process of rotating and stretching. In detail, $A\vec{x} = \vec{b}$ can be visualized as rotating the vector $\vec{x}$ and stretching it to get to $\vec{b}$. SVD implements this idea of rotating and stretching by decomposing the matrix A into two unitary matrices U, V^* as rotation matrices and one diagonal matrix S as stretch matrix. The diagonal values of Σ are lined up in descending order and they correspond to the vector in U and V^*.

Principal Component Analysis(PCA)

A matrix often contains redundant information. Multiple vectors probably are displaying the same piece of data, so we are supposed to reduce the redundancy by analyzing the dominant component of this matrix.

Principal component analysis can be accomplished by performing singular value decomposition. Consider the matrix in $\mathbb{C}^{4\times n}$

$$A = \begin{bmatrix} x_a \\ y_a \\ x_b \\ y_b \end{bmatrix}$$

with all rows being a vector in $\mathbb{C}^n$.

The variance of each vectors can be computed by taking the inner product. For example, $\sigma^2_{x_a} = \frac{1}{n-1} x_a x_a^T$ indicates the energy of x_a in the matrix. Large value of σ^2_a implies that x_a is important in matrix A.

The covariance between these vector, say, $\sigma^2_{x_a x_b}$ is defined as $\frac{1}{n-1} x_a x_b^T$. Similarly, a large covariance indicates the redundancy between these two vectors while small value implies difference.

By calculating $C_A = \frac{1}{n-1} U^* A (U^* A)^T$, I obtain the variance of each vectors and covariance between each two. In fact, Σ matrix in the SVD displays the energy of vectors.

Implementation and results

Tapes of three cameras that record the movement of an object are put together and analyzed. And some of the tapes are just repeating others. Hence, I employ PCA to find out the dominant and major directions of movement. To note that, in this report I work on four sets of data, each of which contains tapes from three cameras.

- Masking on frames of video
  After loading video files into MATLAB, I look into the dimension of these files. They are 4-D arrays that contain RGB values of each frames of size 480×640 spanned in time.

  I transform the RGB values into gray scale first then watch the clips. I discover that the moving object only appears in one particular region out of the whole frame. In order to more accurately detect the movement, I mask the region where the object moves through by setting the gray scale of the rest of the frame to zero. Figure 1 shows an example of masking on one tape.

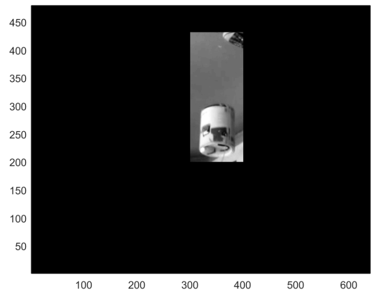

Figure 1: Masked frame in gray scale cam_1_1

- Tracking the object
  After cropping the frames, I start to track the moving object by looking into every frame. The moving paint can seems brighter than the background so I track the brightest pixel of the frame as the movement.

  However, because the lighting condition is changing and the paint can is rotating, the brightest pixel cannot accurately track the movement of the whole can. As a result, I modify this algorithm to track the average location of several multiple brightest pixels. I test different values to find the best percentage at which tracking is accurate enough, ending up using the 5% - 10% brightest pixels. With this algorithm, six vectors of location are generated.
- Aligning tapes from different cameras
  After plotting the location vectors from different cameras, I find that the recordings do not start at the same time. So I must shift the vectors so the time is aligned. Figure 2 shows the how I shift the vector generated by camera 3 to align with other cameras in time.

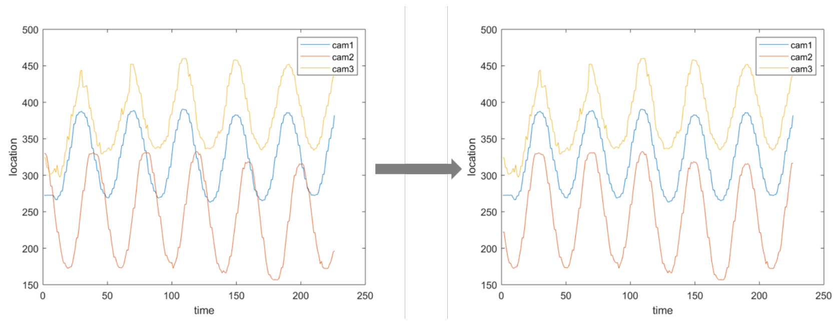

Figure 2: Shifting location vector of camera 3 for time alignment

- Extracting the principal component
  Placing six vectors into a matrix and subtracting the mean value from every entry so the value are centered at zero, I subsequently perform singular value decomposition of this matrix and plot the energy matrix Σ, normalized by dividing each by the total energy. This plot should give us a sense of how many principal there are and how dominant they are by showing their energy percentage. Besides, in order to see the

location of components in the coordinate system on base U, I also plot $V \times \Sigma$. Figure 3 shows the energy of each component and its location in time of the first dataset. There is only one dominant component because the object is only moving on vertical direction.

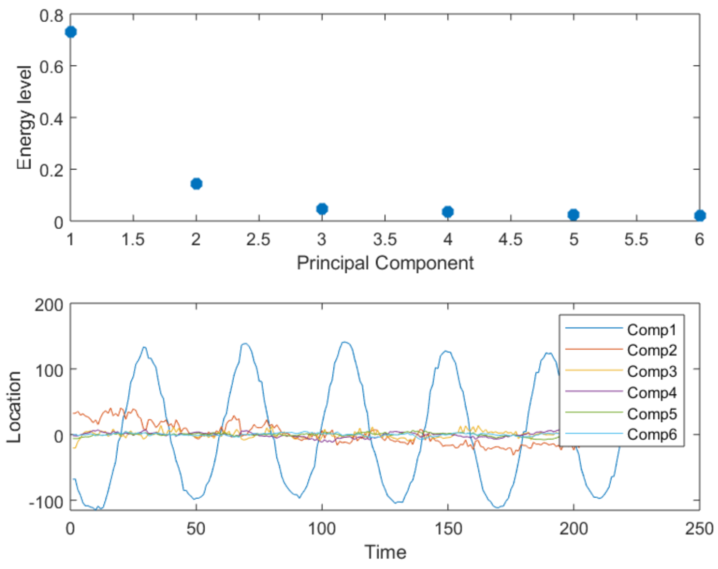

Figure 3: Principal components of Dataset 1

In case 2, because the noise level is much higher, the principal components are not as dominant. Figure 4 shows the energy of principal components and their location changing in time in this new system of base U.

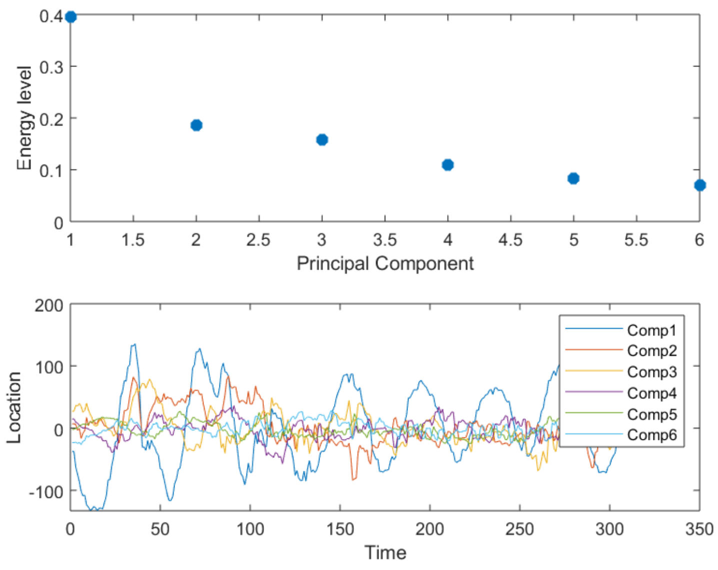

Figure 4: Principal components of Dataset 2

The moving object is moving not only up and down, but also horizontally. As a result, case 3 has two principal components because there are two dimensions of movement. Figure 5 displays the PCA for test 3.

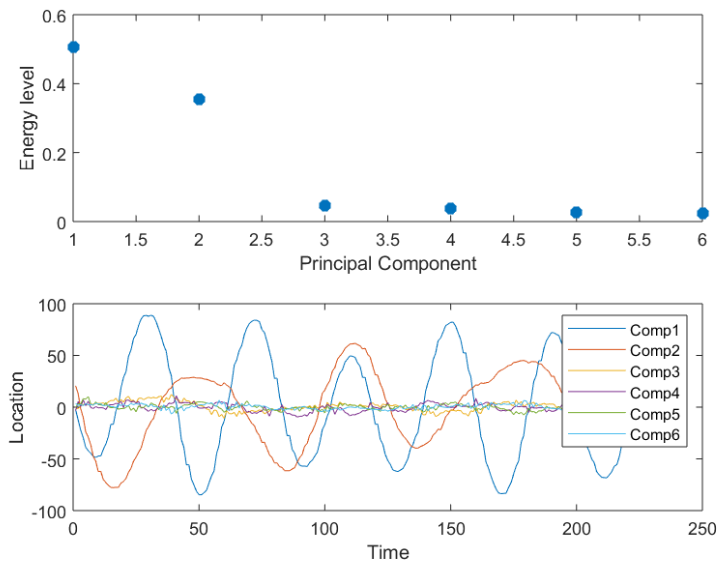

Figure 5: Principal components of Dataset 3

In case 4, the paint can is also moving front and back, so a third dimension is added to the system. However, due to the high noise and fading motion in some directions, the principal components are not clearly dominant. Figure 6 shows the result of analysis for dataset 4.

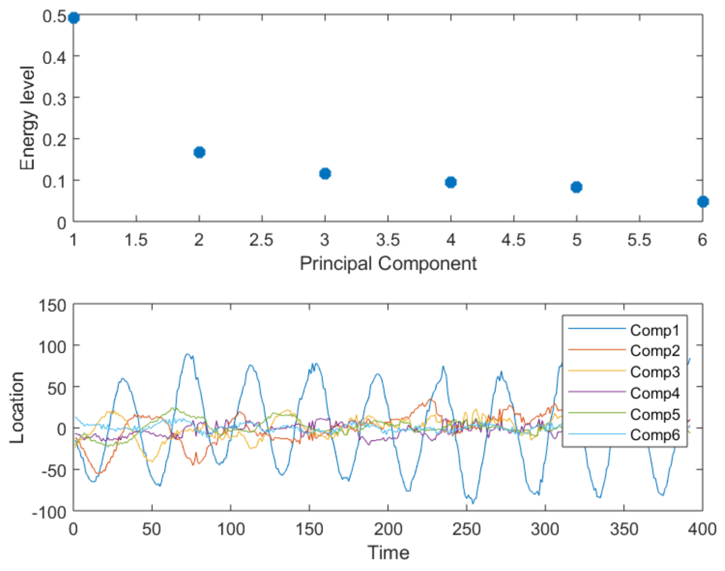

Figure 6: Principal components of Dataset 4

- Reconstructing the matrix from principal component
  To validate my findings, I re-create the matrix to test if it restores the value. For example, if there is one dominant principal component, I am able to approximate the original matrix by computing $U_1 \cdot \Sigma_1 \cdot V_1^*$. The principal components can recreate the original matrix because they store the most part of information in that matrix. Table 1 shows the norm of the original matrix and that of the difference between the recreated one and the original one. Because these components take up the most percentage of energy, the norm of difference is small, relatively.

Table 1: Norm of matrices: how principal components recreate the original matrix

	Test 1	**Test 2**	**Test 3**	**Test 4**
Norm of the original matrix	1257	1134	780	1038
Norm of difference matrix - one component	246	532	547	346
Norm of difference matrix - two component			73	244
Norm of difference matrix - three component				196

Conclusion

PCA enables me to find out the dominant component(s) of the system. In details,

1. PCA shows the energy of each component and determines the dominant one(s). This technique can also produce the change in time of each component.
2. Higher noise level reduces the strength of PCA: the principal component is not as dominant as it really is.
3. Through validation, I can confirm that the principal components contain a large portion of information in the original matrix.

Appendix I: MATLAB commands explained

`rgb2gray(I)` turns RGB values to gray scale

`double(X)` turns integers to double

`max(x(:))` returns the largest value of the matrix

`[a, b] = find(statement)` returns the indices of points that satisfy the statement

`mean(X, k)` returns the mean value of a k-dimension matrix

`repmat(A, M, N)` produces a matrix of size M by N with all entries being A

Appendix II: MATLAB code

Loading files

```
clear all;
load('tapes/cam1_1.mat');
load('tapes/cam2_1.mat');
load('tapes/cam3_1.mat');

load('tapes/cam1_2.mat');
load('tapes/cam2_2.mat');
load('tapes/cam3_2.mat');

load('tapes/cam1_3.mat');
load('tapes/cam2_3.mat');
load('tapes/cam3_3.mat');

load('tapes/cam1_4.mat');
load('tapes/cam2_4.mat');
load('tapes/cam3_4.mat');
```

Ideal case

```
close all; clc
time = length(vidFrames1_1(1, 1, 1, :));
a_1 = zeros(1, time); b_1 = zeros(1, time);
filter = 0.92;
for i = 1:time
    %cleaning the frames
    x = double(rgb2gray(vidFrames1_1(:, :, :, i)));
    x(1:200, :) = 0;
    x(:, 1:300) = 0;
    x(:, 401:end) = 0;

    M = max(x(:));
    [maxa, maxb] = find(x >= M*filter);
    a_1(i) = mean(maxa);
    b_1(i) = mean(maxb);
end
a_2 = zeros(1, time); b_2 = zeros(1, time);
for i = 1:time
    x = double(rgb2gray(vidFrames2_1(:, :, :, i+10)));
    x(1:100, :) = 0;
    x(380:end, :) = 0;
    x(:, 1:250) = 0;
    x(:, 351:end) = 0;

    M = max(x(:));
    [maxa, maxb] = find(x >= M*filter);
    a_2(i) = mean(maxa);
    b_2(i) = mean(maxb);
end
a_3 = zeros(1, time); b_3 = zeros(1, time);
for i = 1:time
```

```
    x = double(rgb2gray(vidFrames3_1(:, :, :, i)));
    x(1:250, :) = 0;
    x(351:end, :) = 0;
    x(:, 1:280) = 0;
    x(:, 501:end) = 0;

    M= max(x(:));
    [maxa, maxb] = find(x >= M*filter);
    a_3(i) = mean(maxa);
    b_3(i) = mean(maxb);
end
%%
%PCA
A = [a_1; b_1; a_2; b_2; a_3; b_3];

[m, n] = size(A);
mn = mean(A, 2);
A = A - repmat(mn, 1, n);

[u, s, v] = svd(A, 'econ');

figure(2)
subplot(2, 1, 1)
plot(diag(s)/sum(diag(s)), '*', 'LineWidth', 3)
xlabel('Principal Component'); ylabel('Energy level')
subplot(2, 1, 2)
plot(v*s)
xlabel('Time'); ylabel('Location')
legend('Comp1', 'Comp2', 'Comp3', 'Comp4', 'Comp5', 'Comp6')
%%
%validating
B = u(:, 1) * s(1, 1) * v(:, 1)';
C = A - B;
```

Noisy case

```
close all; clc
time = length(vidFrames1_2(1, 1, 1, :));
a_1 = zeros(1, time); b_1 = zeros(1, time);
filter = 0.95;

for i = 1:time
    x = double(rgb2gray(vidFrames1_2(:, :, :, i)));
    x(1:200, :) = zeros(200, 640);
    x(:, 1:300) = zeros(480, 300);
    x(:, 401:640) = zeros(480, 240);

    M = max(x(:));
    [maxa, maxb] = find(x >= M*filter);
    a_1(i) = mean(maxa);
    b_1(i) = mean(maxb);
    [a, b] = ind2sub([480, 640], I);
end
a_2 = zeros(1, time); b_2 = zeros(1, time);
for i = 1:time
    x = double(rgb2gray(vidFrames2_2(:, :, :, i+25)));
    x(1:50, :) = 0;
    x(:, 1:150) = 0;
    x(:, 401:640) = 0;

    M = max(x(:));
    [maxa, maxb] = find(x >= M*filter);
    a_2(i) = mean(maxa);
    b_2(i) = mean(maxb);
end
a_3 = zeros(1, time); b_3 = zeros(1, time);
for i = 1:time
```

```
    x = double(rgb2gray(vidFrames3_2(:, :, :, i)));
    x(1:200, :) = 0;
    x(321:480, :) = 0;
    x(:, 1:280) = 0;
    x(:, 501:640) = 0;

    M = max(x(:));
    [maxa, maxb] = find(x >= M*filter);
    a_3(i) = mean(maxa);
    b_3(i) = mean(maxb);
end

%PCA
A = [a_1; b_1; a_2; b_2; a_3; b_3];

[m, n] = size(A);
mn = mean(A, 2);
A = A - repmat(mn, 1, n);

[u, s, v] = svd(A, 'econ');
figure(2)
subplot(2, 1, 1)
plot(diag(s)/sum(diag(s)), '*', 'LineWidth', 3)
xlabel('Principal Component'); ylabel('Energy level')
subplot(2, 1, 2)
plot(v*s)
xlabel('Time'); ylabel('Location')
legend('Comp1', 'Comp2', 'Comp3', 'Comp4', 'Comp5', 'Comp6')

%%
B = u(:, 1) * s(1, 1) * v(:, 1)';
C = A - B;
```

Horizontal case

```
close all; clc
time = length(vidFrames3_3(1, 1, 1, :)) - 10;
a_1 = zeros(1, time); b_1 = zeros(1, time);
filter = 0.95;
for i = 1:time
    x = double(rgb2gray(vidFrames1_3(:, :, :, i+8)));
    x(1:200, :) = 0;
    x(:, 1:250) = 0;
    x(:, 401:end) = 0;

    M = max(x(:));
    [maxa, maxb] = find(x >= M*filter);
    a_1(i) = mean(maxa);
    b_1(i) = mean(maxb);
end
a_2 = zeros(1, time); b_2 = zeros(1, time);
for i = 1:time
    x = double(rgb2gray(vidFrames2_3(:, :, :, i+35)));
    x(1:200, :) = 0;
    x(:, 1:200) = 0;
    x(:, 401:end) = 0;

    M = max(x(:));
    [maxa, maxb] = find(x >= M*filter);
    a_2(i) = mean(maxa);
    b_2(i) = mean(maxb);
end
plot(a_2, 'r'); hold on
a_3 = zeros(1, time); b_3 = zeros(1, time);
for i = 1:time
    x = double(rgb2gray(vidFrames3_3(:, :, :, i)));
    x(1:200, :) = 0;
    x(351:end, :) = 0;
    x(:, 1:280) = 0;
    x(:, 501:end) = 0;

    M = max(x(:));
    [maxa, maxb] = find(x >= M*filter);
    a_3(i) = mean(maxa);
    b_3(i) = mean(maxb);
end
plot(b_3, 'y');

%PCA
A = [a_1; b_1; a_2; b_2; a_3; b_3];

[m, n] = size(A);
mn = mean(A, 2);
A = A - repmat(mn, 1, n);

[u, s, v] = svd(A, 'econ');
figure(2)
subplot(2, 1, 1)
plot(diag(s)/sum(diag(s)), '*', 'LineWidth', 3)
xlabel('Principal Component'); ylabel('Energy level')
subplot(2, 1, 2)
plot(v*s)
xlabel('Time'); ylabel('Location')
legend('Comp1', 'Comp2', 'Comp3', 'Comp4', 'Comp5', 'Comp6')
%%
B = u(:, 1) * s(1, 1) * v(:, 1)' + u(:, 2) * s(2, 2) * v(:, 2)';
C = A - B;
```

Rotation case

```
close all; clc
time = length(vidFrames1_4(1, 1, 1, :));
filter = 0.92;

a_1 = zeros(1, time); b_1 = zeros(1, time);
for i = 1:time
    x = double(rgb2gray(vidFrames1_4(:, :, :, i)));
    x(1:200, :) = 0;
    x(:, 1:300) = 0;
    x(:, 501:end) = 0;

    M = max(x(:));
    [maxa, maxb] = find(x >= M*filter);
    a_1(i) = mean(maxa);
    b_1(i) = mean(maxb);
end
a_2 = zeros(1, time); b_2 = zeros(1, time);
for i = 1:time
    x = double(rgb2gray(vidFrames2_4(:, :, :, i+6)));
    x(1:50, :) = 0;
    x(401:end, :) = 0;
    x(:, 1:220) = 0;
    x(:, 411:end) = 0;

    M = max(x(:));
    [maxa, maxb] = find(x >= M*filter);
    a_2(i) = mean(maxa);
    b_2(i) = mean(maxb);
end
a_3 = zeros(1, time); b_3 = zeros(1, time);
for i = 1:time
```

```
    x = double(rgb2gray(vidFrames3_4(:, :, :, i)));
    x(1:150, :) = 0;
    x(281:end, :) = 0;
    x(:, 1:300) = 0;
    x(:, 511:end) = 0;

    M = max(x(:));
    [maxa, maxb] = find(x >= M*filter);
    a_3(i) = mean(maxa);
    b_3(i) = mean(maxb);
end

%PCA
A = [a_1; b_1; a_2; b_2; a_3; b_3];

[m, n] = size(A);
mn = mean(A, 2);
A = A - repmat(mn, 1, n);

[u, s, v] = svd(A, 'econ');
figure(2)
subplot(2, 1, 1)
plot(diag(s)/sum(diag(s)), '*', 'LineWidth', 3)
xlabel('Principal Component'); ylabel('Energy level')
subplot(2, 1, 2)
plot(v*s)
xlabel('Time'); ylabel('Location')
legend('Comp1', 'Comp2', 'Comp3', 'Comp4', 'Comp5', 'Comp6')
%%
B = u(:, 1) * s(1, 1) * v(:, 1)' + u(:, 2) * s(2, 2) * v(:, 2)' \\
+ u(:, 3) * s(3, 3) * v(:, 3)';
C = A - B;
```

References

Kutz, J. N., (2013). Data-driven modeling scientific computation: Methods for complex systems big data. Oxford: Oxford University Press.