Amath 482 Homework 3

Oscar Zhang
Feb.23rd 2021

Abstract

In this assignment, we are going to apply the SVD(single value decomposition) and PCA(principal component analysis) to 12 videos(3 cameras from different angles as one case, there are four different cases with different conditions). Our purpose is to apply those two methods so that we can filter out the position we are interested in and remove all the information we don't need. Furthermore, the PCA can also calculate the energy content for each single component when the objects are in complex movement and noisy measurements.

1. Introduction

So the background of this assignment is that people usually observe the same objects from different position. To be more specific, there are 12 videos divided into 4 different conditions. Test 1 is the ideal condition – the entire motion is in the z direction with simple harmonic motion. Test 2 is under noisy condition – the camera is shaking. Test 3 is under horizontal displacement – there are both harmonic and pendulum motions. Test 4 is under horizontal displacement and rotation – the mass is released off the center and rotated.
One problem is that there are lots of objects in the image so that we really need to filter out the part we need and the part we don't. So our first goal is to get the part that we want and rebuild a new matrix. Then we apply SVD to the new matrix and PCA to analyze the energy content for each situation.

2. Theoretical Background

So the main two methods and tools we are going to use in this assignment is the SVD and PCA.
SVD, known as Single Value Decomposition, is to transform one matrix, any dimensions n*m, into orthogonal directions. To be more specific,

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

where A is a n*m matrix, the column of U is the left singular vector of A, the column of V is the right singular vector of A. Σ is a diagonal matrix with the diagonal as the singular values of A, from large to small and all are non-negative. Also, the singular value is just the square of the eigenvalues of A^*A, which is the same eigenvalue of AA^*.
PCA, known as Principal Component Analysis, is to project the original data to the principal component basis by diagonalizing the matrix through the SVD method. PCA is a method to calculate the redundancy of two sets of data through calculating the covariance: 1

$$C_X = \frac{1}{n-1}XX^T$$

where C_x is a square matrix, known as covariance matrix.
The main idea here is to change the basis as the following:

$$C_X = \frac{1}{n-1}XX^T = AA^T = U\Sigma^2U^T$$

Multiplying both side by $U^{-1} = U^T$:

$$Y = U^TX$$

Then the covariance of Y is:

$$C_Y = \frac{1}{n-1}YY^T = \frac{1}{n-1}U^TXX^TU = U^TAA^TU = U^TU\Sigma^2U^TU = \Sigma^2$$

Since the off diagonal elements of Σ is zero, then the variables in Y are not correlated.

3. Algorithm Implementation and Development

So since there are four tests and every test has three different positions, we can apply the same code to four tests by just load the different video file. The first step is to load the video file and based on the video, we decide the part we want to keep from the image and set a filter with proper width and length for it. Since we get the frame we want, we need to transfer the image to grayscale so that we can analyze the image more clear by rgb2gray() command and apply double() command to the frame to convert it into double type so that we can analyze the data and apply operation to it. Then we make another filter to track the movement. Since we have apply rgb2gray() and double(), we get a new matrix, we need to find max values of it and also find a point that is pretty close to this max values. To make this approachable, I set the percentage as 95%. Then we need to find the index of those points and use the index of them to find the real coordinate by ind2sub() command. After getting all the coordinates, we average the x and y and store them in a new matrix. After all the steps, we get the data we need for the SVD and PCA.
After we apply the same steps to the other 2 position under the same test, we have three different matrices with different length, we need to find the minimum length among them so that we can keep them in the same length. Then we combined those three new matrices together and get a new matrix with 6 * min_length size. We calculate the mean of every single row and subtract the mean from the matrix so that we can center the data. We apply [u, s, v]command to perform the SVD and square the s to get the eigenvalue, which is also the variance of principal components. Then we get all the data we need to plot.

4. Computational Results

Test 1: Ideal Test
So for the first test, my code doesn't give me the expected output – the first principal component should be extremely high and it also should capture the majority of the energy. However, my projection onto the first principal component is pretty similar to

the actual data. So I have no idea about what is going on. Theoretically, it should only need one principal component since in this test, it's only oscillating in one direction.
<u>percentage: 0.507, 0.470, 0.018, 0.002, 0.001, 0</u>

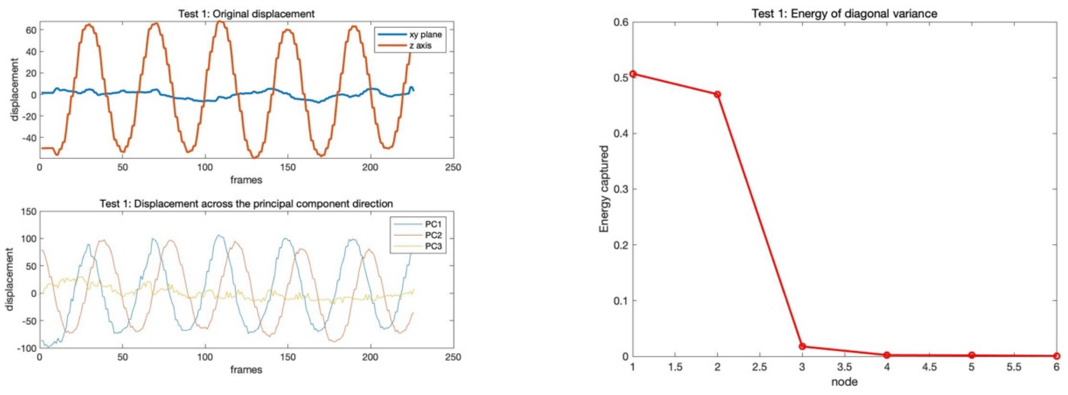

Figure 1: plots for ideal test

Test 2: Noisy case
So for this case, we can see that the first two principal components added up to around 80% of the total energy, which should not be considered as high. Since there is noise in this test, we also need to consider the remaining principal components. However, the projection onto the first principal components is pretty similar to the actual data. Since there are noise, PCA helps us to get rid of some calculation and therefore we can still see a clear oscillatory behavior.
<u>percentage: 0.647, 0.150, 0.111, 0.043, 0.027, 0.022</u>

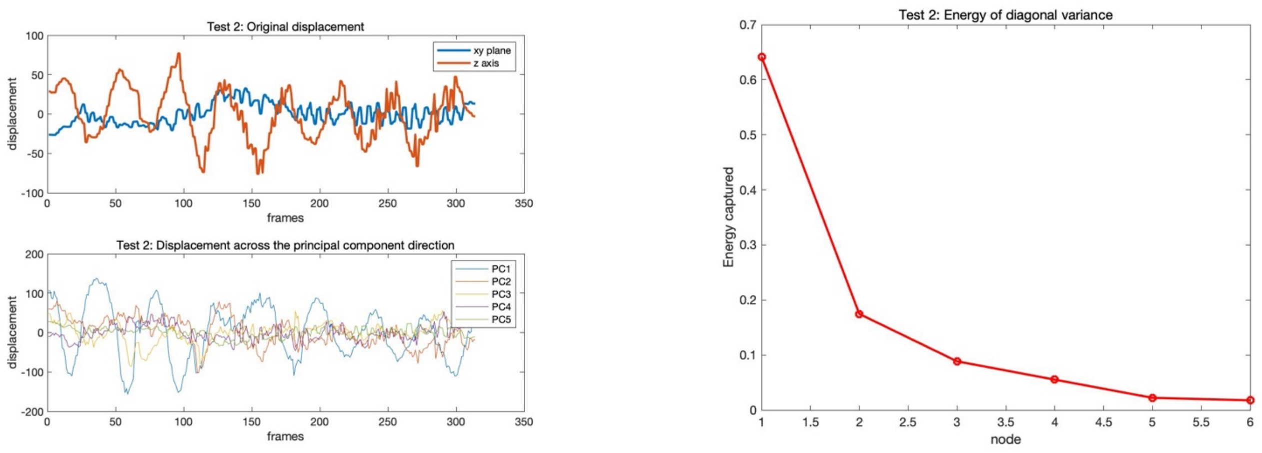

figure 2: plots for noisy case

Test 3: Horizontal displacement

For this test, we see that the sum of the first three principal components sum up to around 94% of the total energy, which is very high, however, the rest 3 have very low percentage. Therefore, the result fits the data – since the can is moving in both x, y, and z direction, it should have three different principal components. In addition, the projection onto the first principal components is pretty similar to the actual data. In addition, we do see a tremendous drop from 1st principal components to 2nd and to 3rd, which also makes the result reasonable.

<u>percentage: 0.429, 0.327, 0.180, 0.053, 0.001, 0.0002</u>

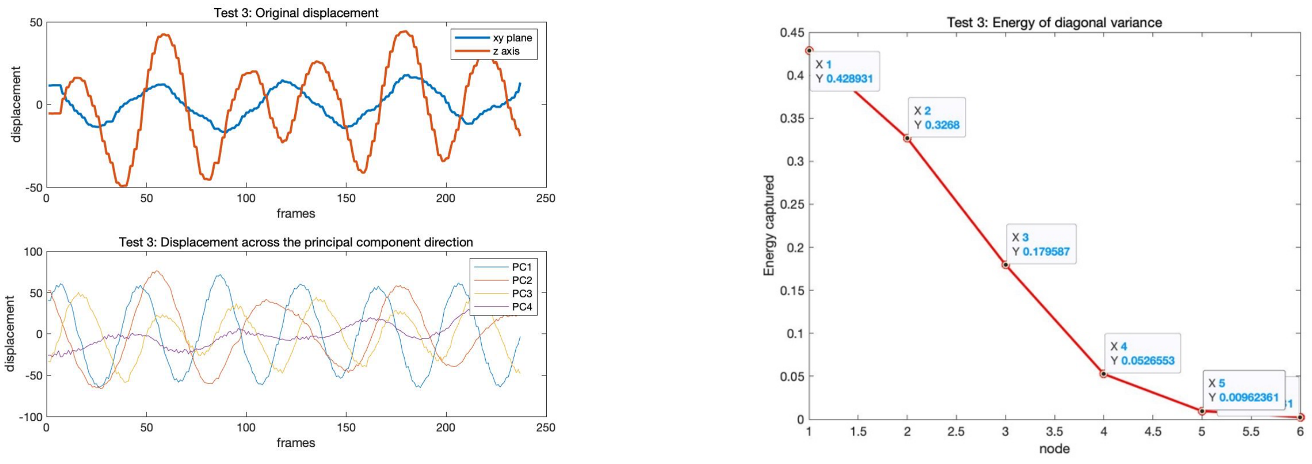

Figure 3: plots for horizontal displacement

Test 4: Horizontal displacement and rotation:

For this case, we also see that the first three principal components added up to around 94% of the total energy, which is also very high and the rest is pretty low. Then So the result fits the test. Since in test 4, it's released off center and also rotates, there are motions in all x, y, and z directions. In addition, the projection onto the first principal components is pretty similar to the actual data. On top of that, we do see a tremendous drop from 1st principal components to 2nd and to 3rd, which also makes the result reasonable.

<u>percentage: 0.592, 0.272, 0.077, 0.048, 0.007, 0.004</u>

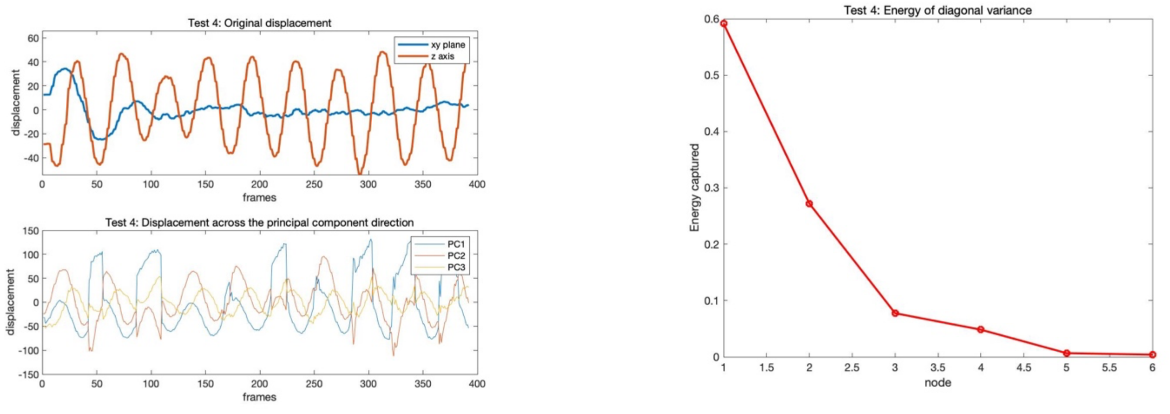

Figure 4: plots for horizontal displacement and rotation

5. Summary and Conclusion

From the above results and plots, we find that PCA is very useful when given some data to find the components when at the same motion but different position. Also, PCA can work as well when there exists some noise but noise can influence PCA method as well(consider case 2)

To sum up, when we are going to analyze the motion of the same object but from different position of viewing it, PCA is always a powerful method since it can always give us the principal components required for the motion and it also determine the one that dominates the motion and it can also be applied when there is noise but noise will certain weaken PCA.

Appendix I: Matlab Commands

1.[U, S, V] = svd(A): perform SVD to matrix A and give us the result in the order of U, S, V
2. (A) = rgb2gray(): convert a colorful image to a grayscale image.
3. A = double(B): convert B into a double precision data A
4. max(A(:)): find the maximum value of matrix/vector A
5. m = mean(A(a)): find the mean of the elements in row/column a of A

Appendix II: Matlab Code

```
%% set up
load('cam1_1.mat')
load('cam2_1.mat')
load('cam3_1.mat')
load('cam1_2.mat')
load('cam2_2.mat')
load('cam3_2.mat')
load('cam1_3.mat')
load('cam2_3.mat')
load('cam3_3.mat')
load('cam1_4.mat')
load('cam2_4.mat')
load('cam3_4.mat')
%% case 1
camera11 = []; %% camera 1
numFrames1 = size(vidFrames1_1, 4);

for i = 1:numFrames1
    filter1 = zeros(480,640);
    filter1(170:430, 300:420) = 1;

    x1 = vidFrames1_1(:, :, :, i);
    x1 = rgb2gray(x1);
    x1 = double(x1);
    x1 = x1 .* filter1;

    indeces = find(x1 > max(x1(:)) * 0.95);
    [y, x] = ind2sub(size(x1), indeces);

    camera11 = [camera11; mean(x), mean(y)];
end

camera21 = []; %% camera 2
```

```
numFrames2 = size(vidFrames2_1, 4);

for i = 1:numFrames2
    filter2 = zeros(480, 640);
    filter2(100:400, 250:350) = 1;

    x2 = vidFrames2_1(:, :, :, i);
    x2 = rgb2gray(x2);
    x2 = double(x2);
    x2 = x2 .* filter2;

    indeces = find(x2 > max(x2(:)) * 0.95);
    [y, x] = ind2sub(size(x2), indeces);

    camera21 = [camera21; mean(x), mean(y)];
end


camera31 = []; %% camera 3
numFrames3 = size(vidFrames3_1, 4);

for i = 1:numFrames3
    filter3 = zeros(480, 640);
    filter3(240:340, 275:475) = 1;

    x3 = vidFrames3_1(:, :, :, i);
    x3 = rgb2gray(x3);
    x3 = double(x3);
    x3 = x3 .* filter3;

    indeces = find(x3 > max(x3(:)) * 0.95);
    [y, x] = ind2sub(size(x3), indeces);

    camera31 = [camera31; mean(x), mean(y)];
end


min_length = min([length(camera11); length(camera21);
length(camera31)]);

camera11 = camera11(1:min_length, :);
camera21 = camera21(1:min_length, :);
camera31 = camera31(1:min_length, :);
data_total = [camera11'; camera21'; camera31'];
```

```
ave = mean(data_total, 2);

data_total = data_total - ave;

A = data_total' / sqrt(min_length - 1);
[U, S, V] = svd(A);
lamda = diag(S) .^ 2;
V = data_total' * V;

figure()
plot(lamda / sum(lamda), 'ro-', 'Linewidth', 2);
title('Test 1: Energy of diagonal variance');
xlabel('node');
ylabel('Energy captured');

figure()
subplot(2, 1, 1)
plot((1:min_length), data_total(1, :), (1:min_length),
data_total(2, :), 'Linewidth', 2)
title('Test 1: Original displacement')
xlabel('frames')
ylabel('displacement')
legend('xy plane', 'z axis')

subplot(2, 1, 2)
plot((1:min_length), V(:, 1), (1:min_length), V(:, 2),
(1:min_length), V(:, 3))
title('Test 1: Displacement across the principal component
direction')
xlabel('frames')
ylabel('displacement')
legend('PC1', 'PC2', 'PC3')

%% case 2
camera12 = []; %% camera 1
numFrames1 = size(vidFrames1_2, 4);

for i = 1:numFrames1
    filter1 = zeros(480,640);
    filter1(200:420, 310:430) = 1;

    x1 = vidFrames1_2(:, :, :, i);
    x1 = rgb2gray(x1);
    x1 = double(x1);
    x1 = x1 .* filter1;

    indeces = find(x1 > max(x1(:)) * 0.95);
    [y, x] = ind2sub(size(x1), indeces);

    camera12 = [camera12; mean(x), mean(y)];
end

camera22 = []; %% camera 2
numFrames2 = size(vidFrames2_2, 4);

for i = 1:numFrames2
    filter2 = zeros(480, 640);
    filter2(80:420, 210:420) = 1;

    x2 = vidFrames2_2(:, :, :, i);
    x2 = rgb2gray(x2);
    x2 = double(x2);
    x2 = x2 .* filter2;

    indeces = find(x2 > max(x2(:)) * 0.95);
    [y, x] = ind2sub(size(x2), indeces);

    camera22 = [camera22; mean(x), mean(y)];
end


camera32 = []; %% camera 3
numFrames3 = size(vidFrames3_2, 4);

for i = 1:numFrames3
    filter3 = zeros(480, 640);
    filter3(200:340, 300:500) = 1;

    x3 = vidFrames3_2(:, :, :, i);
    x3 = rgb2gray(x3);
    x3 = double(x3);
    x3 = x3 .* filter3;

    indeces = find(x3 > max(x3(:)) * 0.95);
    [y, x] = ind2sub(size(x3), indeces);

    camera32 = [camera32; mean(x), mean(y)];
end
```

```
min_length = min([length(camera12); length(camera22);
length(camera32)]);

camera12 = camera12(1:min_length, :);
camera22 = camera22(1:min_length, :);
camera32 = camera32(1:min_length, :);
data_total = [camera12'; camera22'; camera32'];
ave = mean(data_total, 2);

data_total = data_total - ave;

A = data_total' / sqrt(min_length - 1);
[U, S, V] = svd(A);
lamda = diag(S) .^ 2;
V = data_total' * V;

figure()
plot(lamda / sum(lamda), 'ro-', 'Linewidth', 2);
title('Test 2: Energy of diagonal variance');
xlabel('node');
ylabel('Energy captured');

figure()
subplot(2, 1, 1)
plot((1:min_length), data_total(1, :), (1:min_length),
data_total(2, :), 'Linewidth', 2)
title('Test 2: Original displacement')
xlabel('frames')
ylabel('displacement')
legend('xy plane', 'z axis')

subplot(2, 1, 2)
plot((1:min_length), V(:, 1), (1:min_length), V(:, 2),
(1:min_length), V(:, 3),(1:min_length), V(:, 4), (1:min_length), V(:,
5))
title('Test 2: Displacement across the principal component
direction')
xlabel('frames')
ylabel('displacement')
legend('PC1', 'PC2', 'PC3', 'PC4', 'PC5')

%% case3
camera13 = []; %% camera 1
```

```
numFrames1 = size(vidFrames1_3, 4);

for i = 1:numFrames1
   filter1 = zeros(480,640);
   filter1(210:420, 280:400) = 1;

   x1 = vidFrames1_3(:, :, :, i);
   x1 = rgb2gray(x1);
   x1 = double(x1);
   x1 = x1 .* filter1;

   indeces = find(x1 > max(x1(:)) * 0.95);
   [y, x] = ind2sub(size(x1), indeces);

   camera13 = [camera13; mean(x), mean(y)];
end

camera23 = []; %% camera 2
numFrames2 = size(vidFrames2_3, 4);

for i = 1:numFrames2
   filter2 = zeros(480, 640);
   filter2(170:410, 195:420) = 1;

   x2 = vidFrames2_3(:, :, :, i);
   x2 = rgb2gray(x2);
   x2 = double(x2);
   x2 = x2 .* filter2;

   indeces = find(x2 > max(x2(:)) * 0.95);
   [y, x] = ind2sub(size(x2), indeces);

   camera23 = [camera23; mean(x), mean(y)];
end


camera33 = []; %% camera 3
numFrames3 = size(vidFrames3_3, 4);

for i = 1:numFrames3
   filter3 = zeros(480, 640);
   filter3(70:340, 270:470) = 1;

   x3 = vidFrames3_3(:, :, :, i);
```

```
    x3 = rgb2gray(x3);
    x3 = double(x3);
    x3 = x3 .* filter3;

    indeces = find(x3 > max(x3(:)) * 0.95);
    [y, x] = ind2sub(size(x3), indeces);

    camera33 = [camera33; mean(x), mean(y)];
end

min_length = min([length(camera13); length(camera23);
length(camera33)]);

camera13 = camera13(1:min_length, :);
camera23 = camera23(1:min_length, :);
camera33 = camera33(1:min_length, :);
data_total = [camera13'; camera23'; camera33'];
ave = mean(data_total, 2);

data_total = data_total - ave;

A = data_total' / sqrt(min_length - 1);
[U, S, V] = svd(A);
lamda = diag(S) .^ 2;
V = data_total' * V;

figure()
plot(lamda / sum(lamda), 'ro-', 'Linewidth', 2);
title('Test 3: Energy of diagonal variance');
xlabel('node');
ylabel('Energy captured');

figure()
subplot(2, 1, 1)
plot((1:min_length), data_total(1, :), (1:min_length),
data_total(2, :), 'Linewidth', 2)
title('Test 3: Original displacement')
xlabel('frames')
ylabel('displacement')
legend('xy plane', 'z axis')

subplot(2, 1, 2)
plot((1:min_length), V(:, 1), (1:min_length), V(:, 2),
(1:min_length), V(:, 3),(1:min_length), V(:, 4))
```

```
title('Test 3: Displacement across the principal component
direction')
xlabel('frames')
ylabel('displacement')
legend('PC1', 'PC2', 'PC3', 'PC4')

%% case 4
camera14 = []; %% camera 1
numFrames1 = size(vidFrames1_4, 4);

for i = 1:numFrames1
    filter1 = zeros(480,640);
    filter1(170:430, 300:400) = 1;

    x1 = vidFrames1_4(:, :, :, i);
    x1 = rgb2gray(x1);
    x1 = double(x1);
    x1 = x1 .* filter1;

    indeces = find(x1 > max(x1(:)) * 0.95);
    [y, x] = ind2sub(size(x1), indeces);

    camera14 = [camera14; mean(x), mean(y)];
end

camera24 = []; %% camera 2
numFrames2 = size(vidFrames2_4, 4);

for i = 1:numFrames2
    filter2 = zeros(480, 640);
    filter2(80:380, 250:350) = 1;

    x2 = vidFrames2_4(:, :, :, i);
    x2 = rgb2gray(x2);
    x2 = double(x2);
    x2 = x2 .* filter2;

    indeces = find(x2 > max(x2(:)) * 0.95);
    [y, x] = ind2sub(size(x2), indeces);

    camera24 = [camera24; mean(x), mean(y)];
end
```

```
camera34 = []; %% camera 3
numFrames3 = size(vidFrames3_4, 4);

for i = 1:numFrames3
    filter3 = zeros(480, 640);
    filter3(240:340, 265:485) = 1;

    x3 = vidFrames3_4(:, :, :, i);
    x3 = rgb2gray(x3);
    x3 = double(x3);
    x3 = x3 .* filter3;

    indeces = find(x3 > max(x3(:)) * 0.95);
    [y, x] = ind2sub(size(x3), indeces);

    camera34 = [camera34; mean(x), mean(y)];
end

min_length = min([length(camera14); length(camera24);
length(camera34)]);

camera14 = camera14(1:min_length, :);
camera24 = camera24(1:min_length, :);
camera34 = camera34(1:min_length, :);
data_total = [camera14'; camera24'; camera34'];
ave = mean(data_total, 2);

data_total = data_total - ave;

A = data_total' / sqrt(min_length - 1);
[U, S, V] = svd(A);
lamda = diag(S) .^ 2;
V = data_total' * V;

figure()
plot(lamda / sum(lamda), 'ro-', 'Linewidth', 2);
title('Test 4: Energy of diagonal variance');
xlabel('node');
ylabel('Energy captured');

figure()
subplot(2, 1, 1)
plot((1:min_length), data_total(1, :), (1:min_length),
data_total(2, :), 'Linewidth', 2)
```

```
title('Test 4: Original displacement')
xlabel('frames')
ylabel('displacement')
legend('xy plane', 'z axis')

subplot(2, 1, 2)
plot((1:min_length), V(:, 1), (1:min_length), V(:, 2),
(1:min_length), V(:, 3))
title('Test 4: Displacement across the principal component
direction')
xlabel('frames')
ylabel('displacement')
legend('PC1', 'PC2', 'PC3')
```