Lecture 16
Image Separation and MATLAB

Jason J. Bramburger

Having now discussed the theory behind Independent Component Analysis, particularly as it applies to image separation, we will now put this method into practice. We will begin with synthetically blended images to highlight some important aspects of the problem. Then, we will turn to our original motivation which was removing reflections from an image.

Example 1: Synthetic Blending

We will begin by loading two images taken in Sicily into MATLAB. The MATLAB commands are as follows:

```
S1 = imread('sicily1','jpeg');
S2 = imread('sicily2','jpeg');

subplot(1,2,1), imshow(S1);
subplot(1,2,2), imshow(S2);
```

These two images are very different. One overlooks the Mediterranean from the medieval village of Erice and the second is of the Capella Palatina in Palermo. Recall that we required statistical independence of the two images. Thus, the pixel strengths and colours should be largely de-corrolated throughout the image.

As mentioned above, we are going to synthetically mix the images. We will do this with the following matrix

$$\mathbf{A} = \begin{bmatrix} 4/5 & \alpha \\ 1/2 & 2/3 \end{bmatrix}.$$

In what follows we will consider $\alpha = 1/5$ and $\alpha = 3/5$. We can create the synthetically mixed images in MATLAB with the following code, starting with $\alpha = 1/5$.

```
A = [4/5 1/5; 1/2 1/3];
X1 = double(A(1,1)*S1 + A(1,2)*S2);
X2 = double(A(2,1)*S1 + A(2,2)*S2);

subplot(1,2,1), imshow(uint8(X1));
subplot(1,2,2), imshow(uint8(X2));
```

We can following the three steps from the previous lecture to attempt to reconstruct the independent components.

Step 1:
Recall that in this step we are required to find the angle, θ_0, that defines the direction of maximal variance. We already calculated θ_0 and we obtained the formula

$$\theta_0 = \frac{1}{2}\tan^{-1}\left(\frac{-2\sum_{j=1}^{N} x_1(j)x_2(j)}{\sum_{j=1}^{N}[x_2^2(j) - x_1^2(j)]}\right).$$

With this angle we can obtain the rotation matrix $\mathbf{U}^*$ by

$$\mathbf{U}^* = \begin{bmatrix} \cos(\theta_0) & \sin(\theta_0) \\ -\sin(\theta_0) & \cos(\theta_0) \end{bmatrix}.$$

The following MATLAB code finds θ_0 and creates $\mathbf{U}^*$.

```
% Reshape data
[m,n] = size(X1);
x1 = reshape(X1,m*n,1);
x2 = reshape(X2,m*n,1);

% Remove mean
x1 = x1 - mean(x1); x2 = x2 - mean(x2);

% U^* matrix
theta0 = 0.5*atan(-2*sum(x1.*x2)/sum(x1.^2 - x2.^2));
Us = [cos(theta0) sin(theta0); -sin(theta0) cos(theta0)];
```

Step 2:
We now need to rescale the rotated parallelogram using the singular values along the principal component directions θ_0 and $\theta_0 - \pi/2$. The rescaling matrix is computed with the following MATLAB code.

```
sig1 = sum( (x1*cos(theta0) + x2*sin(theta0)).^2);
sig2 = sum( (x1*cos(theta0 - pi/2) + x2*sin(theta0 - pi/2)).^2);
Sigma = [1/sqrt(sig1) 0; 0 1/sqrt(sig2)];
```

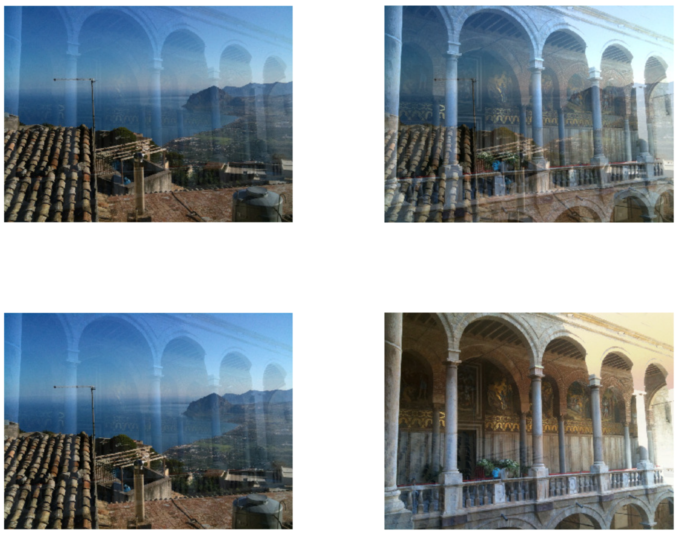
Step 3:
Finally, we apply another rotation that aims at producing, as best as possible, a separable probability distribution. Recall that to do this we require the $\bar{\mathbf{x}}$ variables, which comes from applying $\mathbf{U}^*$ and then Σ^{-1} to our $\mathbf{x}$ variables. We do this first, then determine $\mathbf{V}$.

```
% Create barred variables
```

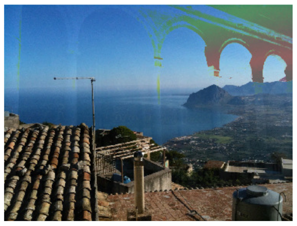
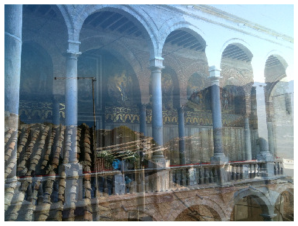
```
X1bar = Sigma(1,1)*(Us(1,1)*X1 + Us(1,2)*X2);
X2bar = Sigma(2,2)*(Us(2,1)*X1 + Us(2,2)*X2);

x1bar = reshape(X1bar,m*n,1);
x2bar = reshape(X2bar,m*n,1);

% V matrix
phi0 = 0.25*atan( -sum(2*(x1bar.^3).*x2bar - 2*x1bar.*(x2bar.^3))...
    /sum(3*(x1bar.^2).*(x2bar.^2) - 0.5*(x1bar.^4) - 0.5*(x2bar.^4)) );

V = [cos(phi0) sin(phi0); -sin(phi0) cos(phi0)];

% Recover independent components
S1bar = V(1,1)*X1bar + V(1,2)*X2bar;
S2bar = V(2,1)*X1bar + V(2,2)*X2bar;
```

The final rotation completes the three steps of computing the SVD matrix. The final thing we need to do is rescale back to the full image brilliance. That is, the data points need to be made positive and scaled back to values ranging from 0 (dim) to 255 (bright). This is achieved with the following code, in which we compare the mixed images with the identified independent components.

```
min1 = min(min(min(S1bar)));
S1bar = S1bar - min1;
max1 =  max(max(max(S1bar)));
S1bar = S1bar*(255/max1);

min2 = min(min(min(S2bar)));
S2bar = S2bar - min2;
max2 =  max(max(max(S2bar)));
S2bar = S2bar*(255/max2);

% Plot mixed images and the independent components together

subplot(2,2,1), imshow(uint8(X1));
subplot(2,2,2), imshow(uint8(X2));
subplot(2,2,3), imshow(uint8(S1bar));
subplot(2,2,4), imshow(uint8(S2bar));
```

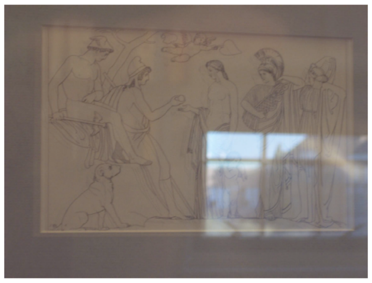
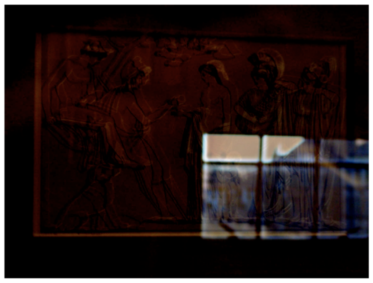


Let's compare with the case $\alpha = 3/5$ instead. (Have a little bit of fun yourself and try out different mixing matrices to see what you get!)

You can see that it isn't perfect. In fact, the more well-mixed the images are, i.e. the 'farther' **A** is from being a diagonal matrix, the harder it is to reproduce the original images.

Example 2: Image Separation from Reflection
The following example comes from the textbook and unfortunately I do not have the original image files. Nonetheless, the implementation is identical to the above implementation, so we don't really need them. Below are four images: the top row are the images with reflection and the bottom row are the independent components. The top right image is a picture of The Judgement of Paris by John Flaxman (1755-1826), which is contained in a glass case. As you can see, there is a reflection on the glass coming from a nearby window. The top left image is the same picture just taken so that the linear polarizer is rotated 90 degrees to produce a different enough image to apply ICA to. The bottom two figures show the extraction process that occurs for the two images.

You can see that the extraction process didn't do a particularly great job. This is largely due to the fact that the reflected image was so significantly brighter than the original photographed image. You can think of this as the mixing matrix **A** having large components in the part of the matrix that multiplies against the reflection independent component. Now, although this might be frustrating, this is happening when you actually look at the image! When it is very bright and sunny out, reflected images are much more difficult for us to see through to the true image.

Example 3: Separating Sounds

It's Friday night and I've spent a long week teaching. I'm looking to unwind and listen to some music, but all I can hear is laughter from the apartment next door. My neighbour must be having another party! You can imagine that there is one microphone set up in my apartment and one in my neighbours. I'm trying to listen to my music loud enough to drown the party out, but you can still hear the laughter through our paper thin walls. Similarly, underneath all the laughter at the party, you can faintly hear my music. We are going to apply ICA to separate out the two distinct sounds that can be picked up on each microphone. The following code loads in the sounds and uses our ICA algorithm to approximate the SVD of the mixing matrix.

```matlab
% Clean workspace
clear all; close all; clc

load('laughter')
S1 = y;
load('handel')
S2 = y(1:length(S1));

A = [0.85 -0.5; 0.1 0.9];
X1 = double(A(1,1)*S1 + A(1,2)*S2);
X2 = double(A(2,1)*S1 + A(2,2)*S2);

% Play the mixed signals
% soundsc(X1)
% soundsc(X2)

% Remove mean
x1 = X1 - mean(X1); x2 = X2 - mean(X2);

% U^* matrix
theta0 = 0.5*atan(-2*sum(x1.*x2)/sum(x1.^2 - x2.^2));
Us = [cos(theta0) -sin(theta0); sin(theta0) cos(theta0)];

% Sigma values
sig1 = sum( (x1*cos(theta0) + x2*sin(theta0)).^2);
sig2 = sum( (x1*cos(theta0 - pi/2) + x2*sin(theta0 - pi/2)).^2);
Sigma = [1/sqrt(sig1) 0; 0 1/sqrt(sig2)];

% Create barred variables
x1bar = Sigma(1,1)*(Us(1,1)*X1 + Us(1,2)*X2);
x2bar = Sigma(2,2)*(Us(2,1)*X1 + Us(2,2)*X2);

% V matrix
phi0 = 0.25*atan( -sum( (2*(x1bar.^3).*x2bar - 2*x1bar.*(x2bar.^3)))...
    /sum(3*(x1bar.^2).*(x2bar.^2) - 0.5*(x1bar.^4) - 0.5*(x2bar.^4)) );

V = [cos(phi0) sin(phi0); -sin(phi0) cos(phi0)];

% Recover independent components
S1bar = V(1,1)*x1bar + V(1,2)*x2bar;
S2bar = V(2,1)*x1bar + V(2,2)*x2bar;
```