Lecture 17
Advanced Discussion of ICA

Jason J. Bramburger

In the previous two lectures we introduced Independent Component Analysis (ICA) and saw how when we have only two independent components it can be related to the SVD. There are of course many scenarios that one could conjure up where we would have more than two measurements/observations/signals that we would like to decompose into independent components. Unfortunately, the SVD method doesn't scale to more than two components, so in this lecture we will discuss some more advanced methods for obtaining the independent components.

Recall that the general setting for ICA is that we are given measurements $\mathbf{x}$ and we seek to identify a mixing matrix $\mathbf{A}$ and the independent components $\mathbf{s}$ such that

$$\mathbf{x} = \mathbf{A}\mathbf{s}.$$

That is, the measurements are just linear combinations of the independent components. If we know what the mixing matrix $\mathbf{A}$ is, we could then simply invert it (if it is invertible) to solve for the independent components

$$\mathbf{s} = \mathbf{A}^{-1}\mathbf{x}.$$

Unfortunately, we don't have $\mathbf{A}$, so the entire goal going forward will be to estimate it knowing only the measurements $\mathbf{x}$. Let's first start with how we can preprocess the data $\mathbf{x}$ to make the problem more tractable.

Pre-processing for ICA

Before applying an ICA algorithm to the measurement data $\mathbf{x}$, it is typically advantageous to do some preprocessing first. There are two main things that should be done. The second of which has major repercussions for simplifying estimating the inverse of the matrix $\mathbf{A}$ from the measurement data.

Centring: We already saw this one in the previous lectures. The first thing you should do with any measurement data is subtract the mean so that the data has mean zero. This will also give that the independent components $\mathbf{s}$ will have mean zero.

Whitening: Anther useful preprocessing strategy in ICA is to first whiten the measurements. Whitening means that we apply an invertible linear transformation to the measurements $\mathbf{x}$ to obtain a new set of measurements $\tilde{\mathbf{x}}$ whose components are uncorrelated and their variances are equal to unity. Mathematically, this means that the covariance matrix satisfies

$$\mathbf{C}_{\tilde{\mathbf{x}}} := \frac{1}{n-1}\tilde{\mathbf{x}}\tilde{\mathbf{x}}^T = \mathbf{I}.$$

Recall from our lecture on principal component analysis, the eigenvalue decomposition of

$$\mathbf{C}_{\mathbf{x}} := \frac{1}{n-1}\mathbf{x}\mathbf{x}^T$$

provides an orthogonal matrix $\mathbf{V}$ (the principal components) such that

$$\mathbf{C}_{\mathbf{x}} = \mathbf{V}\Lambda\mathbf{V}^T,$$

since $\mathbf{V}^T = \mathbf{V}^{-1}$. We can then produce the whitened measurements by

$$\tilde{\mathbf{x}} = \mathbf{V}\Lambda^{-1/2}\mathbf{V}^T\mathbf{x},$$

where $\Lambda^{-1/2}$ is just the diagonal matrix formed by raising each diagonal element of Λ to the power of $-1/2$.

Applying the whitening transform gives the new mixing equation

$$\tilde{\mathbf{x}} = \mathbf{V}\Lambda^{-1/2}\mathbf{V}^T\mathbf{A}\mathbf{s} =: \tilde{\mathbf{A}}\mathbf{s}.$$

What was the point? Well, the new matrix $\tilde{\mathbf{A}}$ is now orthogonal! That means that if we have N measurements, we have N^2 unknowns that make up the matrix $\mathbf{A}$, while the orthogonal matrix $\tilde{\mathbf{A}}$ has only $N(N-1)/2$ unknowns. This dimensionality reduction comes from the orthogonality constraint $\tilde{\mathbf{A}}^{-1} = \tilde{\mathbf{A}}^T$. How do we know $\tilde{\mathbf{A}}$ is orthogonal? Well,

$$\mathbf{I} = \frac{1}{n-1}\tilde{\mathbf{x}}\tilde{\mathbf{x}}^T = \frac{1}{n-1}\tilde{\mathbf{A}}\mathbf{s}\mathbf{s}^T\tilde{\mathbf{A}}^T,$$

and since the independent components are uncorrelated, it follows that

$$\frac{1}{n-1}\mathbf{s}\mathbf{s}^T = \mathbf{I},$$

resulting in the orthogonality condition for $\tilde{\mathbf{A}}$:

$$\mathbf{I} = \tilde{\mathbf{A}}\left(\frac{1}{n-1}\mathbf{s}\mathbf{s}^T\right)\tilde{\mathbf{A}}^T = \tilde{\mathbf{A}}\tilde{\mathbf{A}}^T.$$

Implementing ICA Numerically

Now that we have preprocessed the data, we can turn to actually finding the independent components. Recall that we have reduced ourselves to finding the orthogonal matrix $\tilde{\mathbf{A}}$, thus giving that

$$\mathbf{s} = \tilde{\mathbf{A}}^{-1}\tilde{\mathbf{x}}.$$

Notice that we haven't altered the independent components, so we don't have to transform anything back once we are done. Since we are actually interested in obtaining $\tilde{\mathbf{A}}^{-1}$, instead of $\tilde{\mathbf{A}}$, we will keep with convention in the literature and define $\mathbf{W} := \mathbf{A}^{-1}$. The matrix $\mathbf{W}$ makes up the *weights* of each measurement that go into building each independent component. At the highest level, algorithms that seek to perform ICA come down to variations on the following minimization process:

$$\text{minimize } \|\mathbf{W}\mathbf{x}\|_1 \quad \text{s.t. } \mathbf{W}\mathbf{W}^T = \mathbf{I}.$$

First, the constraint $\mathbf{W}\mathbf{W}^T = \mathbf{I}$ is exactly the constraint that $\mathbf{W}$ is an orthogonal matrix. There are lots of orthogonal matrices, especially for large N, so the quantity that we are minimizing helps us to specify which one we would like. It should be noted that every orthogonal matrix $\mathbf{W}$ provides uncorrelated independent components since $\tilde{\mathbf{x}}$ is uncorrelated (try and check this fact yourself). Then, we are minimizing a 1-norm: the sum of the absolute value of all of the components of $\mathbf{W}\mathbf{x}$. This minimum seeks to impose a *sparsity* constraint on $\mathbf{W}$. Loosely, it seeks to obtain a matrix $\mathbf{W}$ with as many zero values in it as possible. This keeps things simple, while also adhering to what has been observed in many applications.

Reconstruction ICA (RICA)

Searching for a matrix $\mathbf{W}$ that satisfies the above minimization procedure can be very difficult from a numerical standpoint. The reason for this is that we have hard constraints, coming from the equality condition $\mathbf{W}\mathbf{W}^T = \mathbf{I}$. Equality conditions like this are notoriously difficult in optimization problems. This has caused researchers to look for ways to relax this condition.

We are going to present one of these relaxations: the reconstruction ICA (RICA). This is the one that MATLAB uses for to preform ICA, so it's worth talking about. The idea is quite simple. We just need to replace the equality constraint with something slightly looser:

$$\text{minimize } \|\mathbf{W}\mathbf{x}\|_1 + \lambda\|\mathbf{W}\mathbf{W}^T\mathbf{x} - \mathbf{x}\|_2^2.$$

We can see that the sparsity constraint is still in there, but now we've added another piece. The second piece aims to get $\mathbf{W}\mathbf{W}^T$ as close to the identity (in the 2-norm) as possible, without imposing that constraint outright. This

leaves a little room for numerical error, but greatly improves the performance and brings down the run-time. The value $\lambda > 0$ is the *regularization parameter*, and can be used to preference one aspect of the minimization process over the other. With $\lambda > 1$ you preference the orthogonality constraint, and with $\lambda < 1$ you preference sparsity.

MATLAB Implementation

Now that we have a handle on how the theory behind implementing ICA works, let's follow an example provided by MATLAB to see how it performs. Let's first load in some sounds to blend together.

```
% Sound files to load
files = {'chirp.mat'
         'gong.mat'
         'handel.mat'
         'laughter.mat'
         'splat.mat'
         'train.mat'};

% Matrix of principal components
S = zeros(10000,6);
for i = 1:6
    test     = load(files{i});
    y        = test.y(1:10000,1);
    S(:,i)   = y;
end

% Play sound
soundsc(S(:,1))
```

Now, let's mix the signals together using a random mixing matrix and add a random offset.

```
rng default % For reproducibility
mixdata = S*randn(6) + randn(1,6);

% listen to mixed signal
soundsc(mixdata(:,1))
```

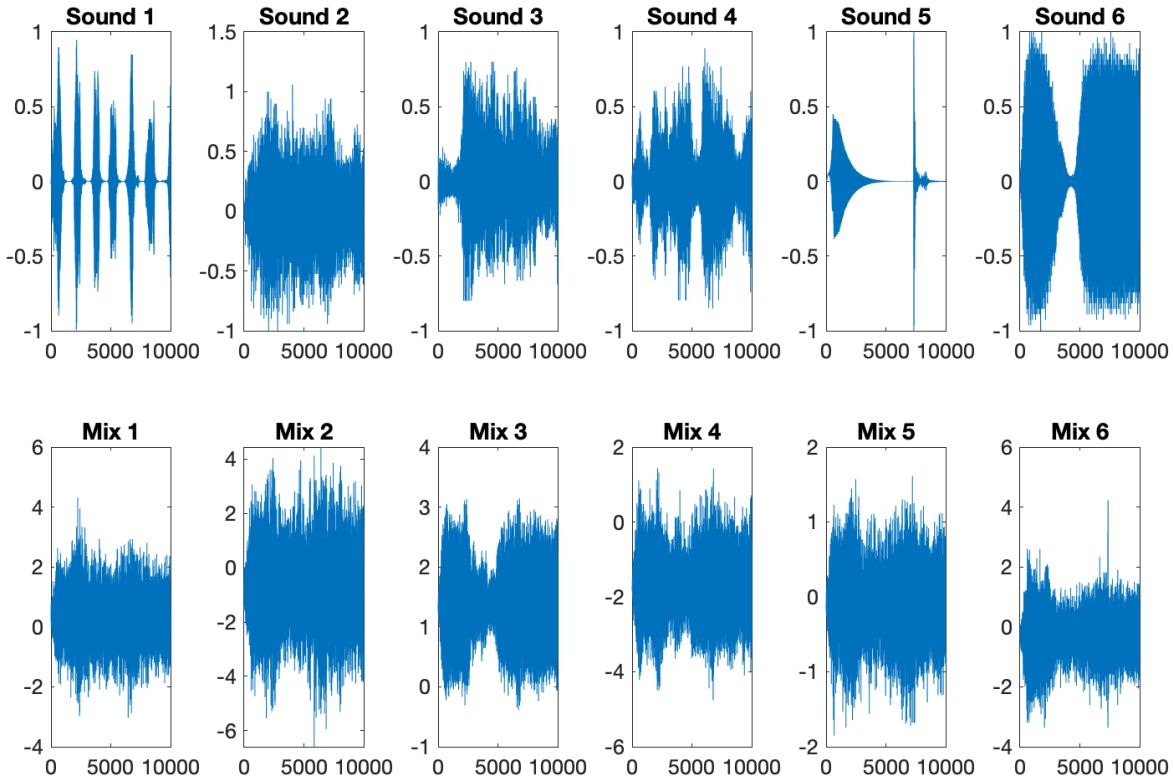
Let's plot the signals along with the mixed signals.

```
figure(1)
for i = 1:6
    subplot(2,6,i)
    plot(S(:,i))
    title(['Sound ',num2str(i)])
    set(gca,'Fontsize',12)
    subplot(2,6,i+6)
    plot(mixdata(:,i))
    title(['Mix ',num2str(i)])
    set(gca,'Fontsize',12)
end
```

As discussed above, we need to preprocess the data to simplify the optimization problem. Let's subtract off the mean and whiten the data.

```
% Compute SVD of covariance matrix
[U,Sig] = svd(cov(mixdata));
Sig     = diag(Sig);
Sig     = Sig(:)';

% Compute whitened data
mu = mean(mixdata,1);
mixdata = bsxfun(@minus,mixdata,mu);
```




```
mixdata = bsxfun(@times,mixdata*U,1./sqrt(Sig));
```

We are now ready to apply MATLAB's built-in RICA feature. We need to understand some things first. MATLAB uses gradient descent to find the minimum of a function, meaning that it needs at least an approximation of a derivative. Since the 1-norm isn't a smooth function (it is sums of absolute values), MATLAB doesn't use it. It replaces the absolute values with a smooth function that retains many of the same features. The default is

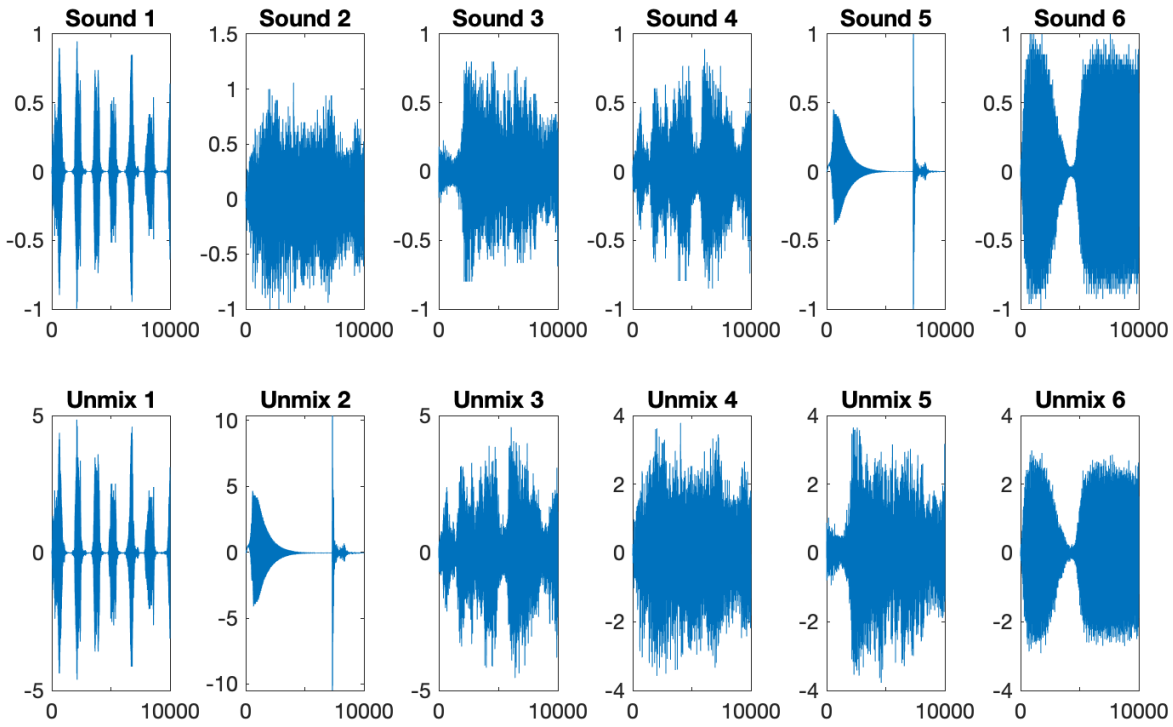
$$g(x) = \frac{1}{2}\log(\cosh(2x),$$

but there are others that can be used in practice. This is a common trick that is used in optimization to retain the same features of the 1-norm, while also using something that is smooth. The following code applies the RICA algorithm and then uses the result to un-mix the signals.

```
q = 6; % number of independent components
Mdl = rica(mixdata,q);
unmixed = transform(Mdl,mixdata);
```

Now, plot the un-mixed signals.

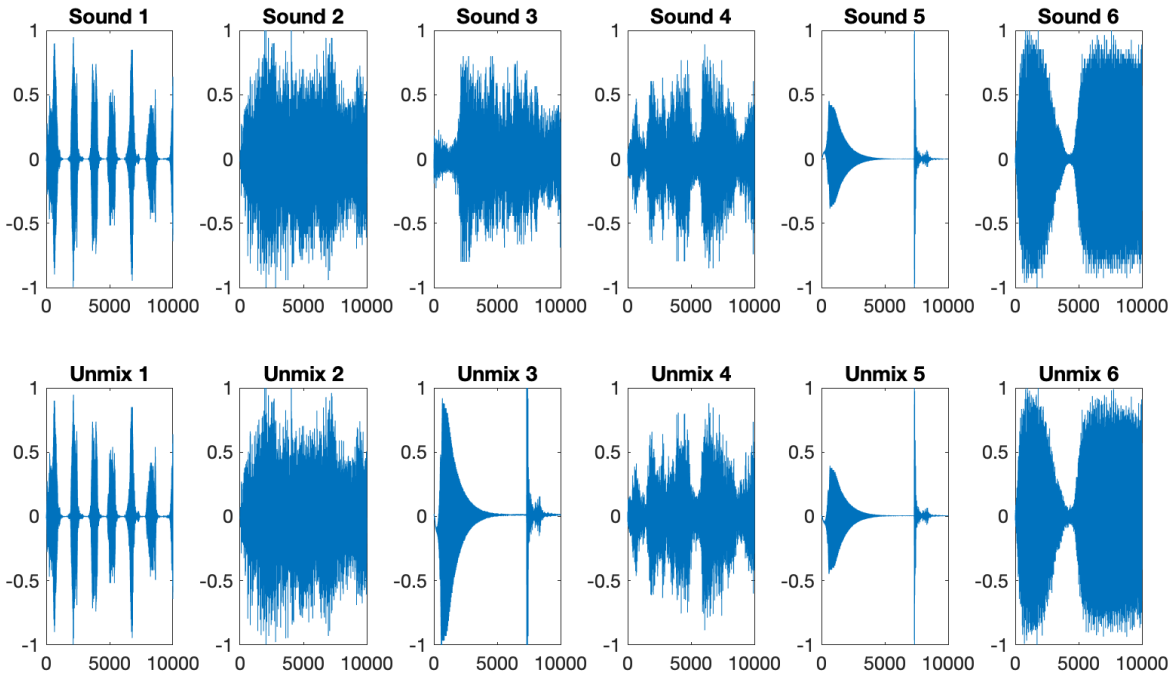
```
figure(2)
for i = 1:6
    subplot(2,6,i)
    plot(S(:,i))
    title(['Sound ',num2str(i)])
    set(gca,'Fontsize',12)
    subplot(2,6,i+6)
    plot(unmixed(:,i))
    title(['Unmix ',num2str(i)])
    set(gca,'Fontsize',12)
end
```



The order is mixed up and the scales are off, but other than that things look pretty good! This is just a feature of ICA. We can receive our independent components in any order and with any scale and still get the right answer. This comes from the fact that we need both $\mathbf{A}$ and $\mathbf{s}$, so changes in $\mathbf{s}$ can be undone with changes to $\mathbf{A}$. Let's reorder the independent components and re-scale them appropriately.

```
unmixed = unmixed(:,[2,5,3,6,3,1]);
for i = 1:6
    unmixed(:,i) = unmixed(:,i)/norm(unmixed(:,i))*norm(S(:,i));
end

figure(3)
for i = 1:6
    subplot(2,6,i)
    plot(S(:,i))
    ylim([-1,1])
    title(['Sound ',num2str(i)])
    set(gca,'Fontsize',12)
    subplot(2,6,i+6)
    plot(unmixed(:,i))
    ylim([-1,1])
    title(['Unmix ',num2str(i)])
    set(gca,'Fontsize',12)
end
```


Sound 1
Sound 2
Sound 3
Sound 4
Sound 5
Sound 6
Unmix 1
Unmix 2
Unmix 3
Unmix 4
Unmix 5
Unmix 6
1
0.5
0
-0.5
-1
0
5000
10000