

Independent Component Analysis

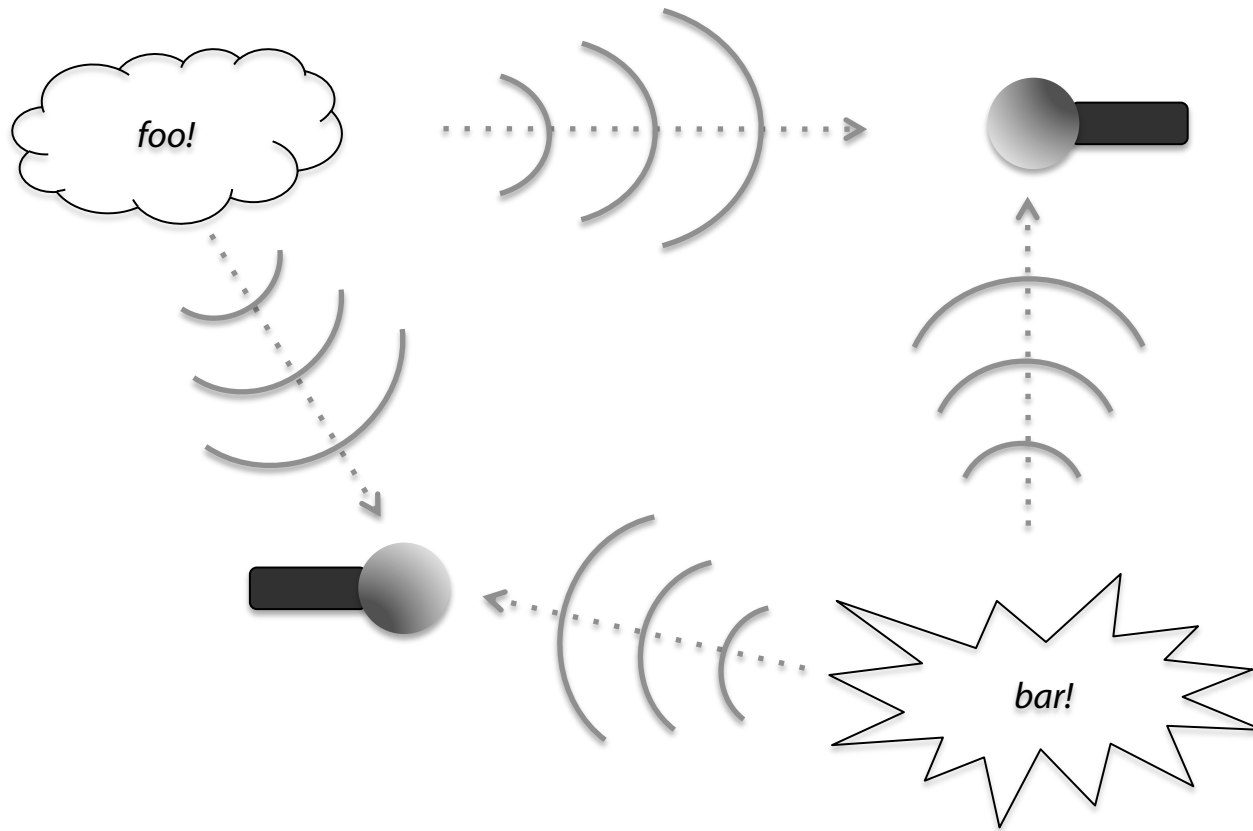
Paris Smaragdis

paris@adobe.com

This lecture's overview

- A motivating example
- The theory
 - Decorrelation
 - Independence vs decorrelation
 - Independent component analysis
- Separating sounds
 - Solving instantaneous mixtures
 - Solving convolutive mixtures
- Data exploration and independence
 - Extracting audio features
 - Extracting multimodal features

A “simple” audio problem



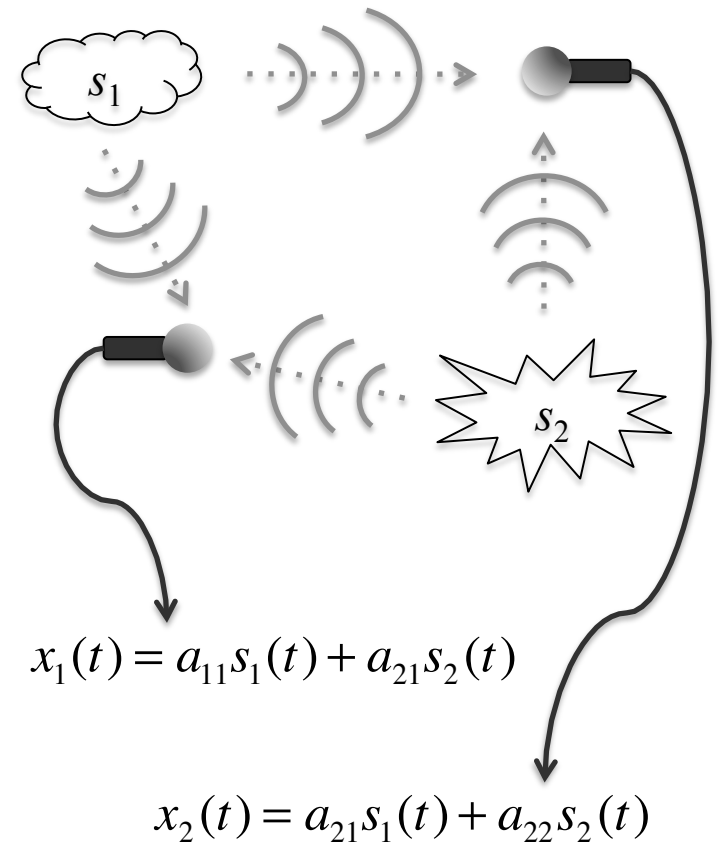
Formalizing the problem

- Each mic will receive a mix of both sounds
 - Sound waves superimpose linearly
 - We'll ignore propagation delays for now

- The simplified mixing model is:

$$\mathbf{x}(t) = \mathbf{A} \cdot \mathbf{s}(t)$$

- We know $\mathbf{x}(t)$, but nothing else
 - How do we solve this system and find $\mathbf{s}(t)$?



When can we solve this?

- The mixing equation is:

$$\mathbf{x}(t) = \mathbf{A} \cdot \mathbf{s}(t)$$

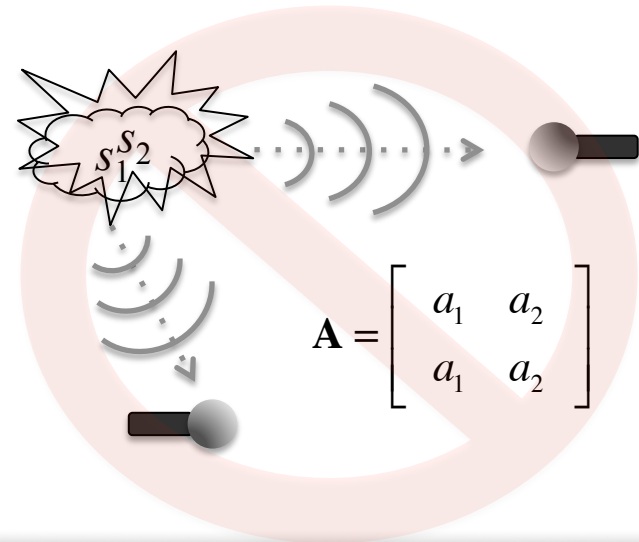
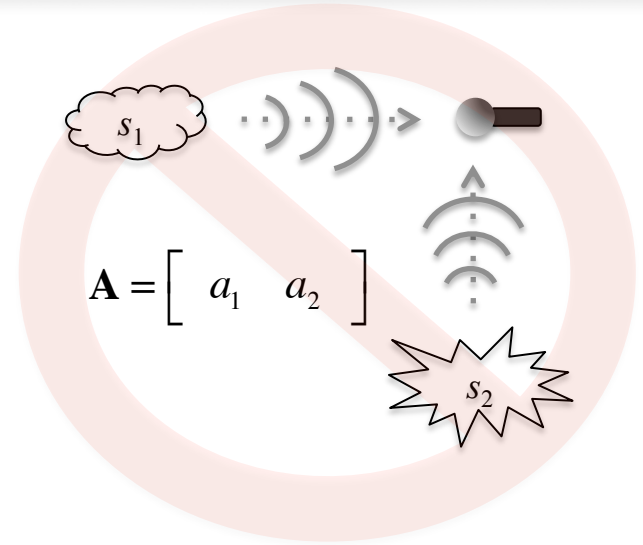
- Our estimates of $\mathbf{s}(t)$ will be:

$$\hat{\mathbf{s}}(t) = \mathbf{A}^{-1} \cdot \mathbf{x}(t)$$

- To recover $\mathbf{s}(t)$, \mathbf{A} must be invertible:

- We need as many mics as sources
- The mics/sources must not coincide
- All sources must be audible

- Otherwise this is a different story ...



A simple example

$$\begin{pmatrix} \mathbf{x}(t) \\ \mathbf{x}(t) \end{pmatrix} = \mathbf{A} \begin{pmatrix} \mathbf{s}(t) \\ \mathbf{s}(t) \end{pmatrix}$$

- A simple invertible problem
 - $\mathbf{s}(t)$ contains two structured waveforms
 - \mathbf{A} is invertible (but we don't know it)
 - $\mathbf{x}(t)$ looks messy, doesn't reveal $\mathbf{s}(t)$ clearly
- How do we solve this? Any ideas?

What to look for

$$\mathbf{x}(t) = \mathbf{A} \cdot \mathbf{s}(t)$$

- We can only use $\mathbf{x}(t)$
- Is there a property we can take advantage of?
- Yes! We know that different sounds are “*statistically unrelated*”
- The plan: Find a solution that enforces this “*unrelatedness*”

A first try

- Find $\mathbf{s}(t)$ by minimizing cross-correlation

- Our estimate of $\mathbf{s}(t)$ is computed by:

$$\hat{\mathbf{s}}(t) = \mathbf{W} \cdot \mathbf{x}(t)$$

- If $\mathbf{W} \approx \mathbf{A}^{-1}$ then we have a good solution
- The goal is that the output becomes uncorrelated:

$$\langle \hat{s}_i(t) \cdot \hat{s}_j(t) \rangle = 0, \forall i \neq j$$

- We assume here that our signals are zero mean
- So the overall problem to solve is:

$$\arg \min_{\mathbf{W}} \left\langle \sum_k a_{ik} x_k(t) \cdot \sum_k a_{jk} x_k(t) \right\rangle, \forall i \neq j$$

How to solve for uncorrelatedness

- Let's use matrices instead of time-series:

$$\mathbf{x}(t) \rightarrow \mathbf{X} = \begin{bmatrix} x_1(1), & \dots, & x_1(N) \\ x_2(1), & \dots, & x_2(N) \end{bmatrix}, \text{etc} \quad \begin{aligned} \mathbf{x}(t) &= \mathbf{A} \cdot \mathbf{s}(t) \quad \text{and} \quad \hat{\mathbf{s}}(t) = \mathbf{W} \cdot \mathbf{x}(t) \Rightarrow \\ &\Rightarrow \mathbf{X} = \mathbf{A} \cdot \mathbf{S} \quad \text{and} \quad \hat{\mathbf{S}} = \mathbf{W} \cdot \mathbf{X} \end{aligned}$$

- The uncorrelatedness translates to:

$$\frac{1}{N} \hat{\mathbf{S}} \cdot \hat{\mathbf{S}}^T = \begin{bmatrix} c_{11} & 0 \\ 0 & c_{22} \end{bmatrix}$$

- We then need to diagonalize:

$$\frac{1}{N} \hat{\mathbf{S}} \cdot \hat{\mathbf{S}}^T = \frac{1}{N} (\mathbf{W} \cdot \mathbf{X})(\mathbf{W} \cdot \mathbf{X})^T$$

How to solve for uncorrelatedness

- This is actually a well known problem in linear algebra
- One solution is Eigenanalysis:

$$\begin{aligned} \text{Cov}(\hat{\mathbf{S}}) &= \frac{1}{N} \hat{\mathbf{S}} \cdot \hat{\mathbf{S}}^T \\ &= \frac{1}{N} (\mathbf{W} \cdot \mathbf{X})(\mathbf{W} \cdot \mathbf{X})^T \\ &= \frac{1}{N} \mathbf{W} \cdot \mathbf{X} \cdot \mathbf{X}^T \cdot \mathbf{W}^T \\ &= \mathbf{W} \cdot \text{Cov}(\mathbf{X}) \cdot \mathbf{W}^T \end{aligned}$$

- $\text{Cov}(\mathbf{X})$ is a symmetric matrix

How to solve for uncorrelatedness

- For any symmetric matrix \mathbf{Z} we know that:

$$\mathbf{Z} = \mathbf{U} \cdot \mathbf{D} \cdot \mathbf{U}^T = \begin{bmatrix} \uparrow & & \uparrow \\ \mathbf{u}_1 & \cdots & \mathbf{u}_N \\ \downarrow & & \downarrow \end{bmatrix} \cdot \begin{bmatrix} d_1 & & \\ & \ddots & \\ & & d_N \end{bmatrix} \cdot \begin{bmatrix} \leftarrow & \mathbf{u}_1 & \rightarrow \\ & & \\ \leftarrow & \mathbf{u}_N & \rightarrow \end{bmatrix}$$

- Where \mathbf{u}_i and d_i are \mathbf{Z} 's eigenvectors and eigenvalues respectively
- In our case if $[\mathbf{U}, \mathbf{D}] = \text{eig}(\text{Cov}(\mathbf{X}))$

$$\begin{aligned} \text{Cov}(\hat{\mathbf{S}}) &= \mathbf{W} \cdot \text{Cov}(\mathbf{X}) \cdot \mathbf{W}^T \\ &= \mathbf{W} \cdot \mathbf{U} \cdot \mathbf{D} \cdot \mathbf{U}^T \cdot \mathbf{W}^T && \text{let us replace } \mathbf{W} \text{ with } \mathbf{U}^T \\ &= \mathbf{U}^T \cdot \mathbf{U} \cdot \mathbf{D} \cdot \mathbf{U}^T \cdot \mathbf{U} && \mathbf{U} \text{ is orthonormal } (\mathbf{U} \cdot \mathbf{U}^T = \mathbf{I}) \\ &= \mathbf{D} \end{aligned}$$

- This is actually Principal Component Analysis

Another solution

- We can also solve for a matrix inverse square root:

$$\begin{aligned} \text{Cov}(\hat{\mathbf{S}}) &\propto \mathbf{W} \cdot \mathbf{X} \cdot \mathbf{X}^T \cdot \mathbf{W}^T \\ &= (\mathbf{X} \cdot \mathbf{X}^T)^{-1/2} \cdot \mathbf{X} \cdot \mathbf{X}^T \cdot (\mathbf{X}^T \cdot \mathbf{X})^{-1/2} \quad \text{replace } \mathbf{W} \text{ with } (\mathbf{X} \cdot \mathbf{X}^T)^{-1/2} \\ &= \mathbf{I} \end{aligned}$$

$$(\mathbf{X} \cdot \mathbf{X})^{-1/2} = \mathbf{U} \cdot \begin{bmatrix} d_1^{-1/2} & & \\ & \ddots & \\ & & d_N^{-1/2} \end{bmatrix} \cdot \mathbf{U}^T, \quad \text{where } [\mathbf{U}, \mathbf{D}] = \text{eig}(\mathbf{X} \cdot \mathbf{X}^T)$$

Another approach

- What if we want to do this in real-time?

- We can also estimate \mathbf{W} in an online manner:

$$\Delta\mathbf{W} \propto \mu(\mathbf{I} - \mathbf{W} \cdot \mathbf{x}(t) \cdot \mathbf{x}(t)^T \cdot \mathbf{W}^T) \mathbf{W}$$

- Every time we see a new observation $\mathbf{x}(t)$ we update \mathbf{W}

- Using this adaptive approach we can see that:

$$\text{Cov}(\mathbf{W} \cdot \mathbf{x}(t)) = \mathbf{I}, \quad \Delta\mathbf{W} = 0$$

- We'll skip the derivation details for now

Summary so far

- For a mixture:

$$\mathbf{X} = \mathbf{A} \cdot \mathbf{S}$$

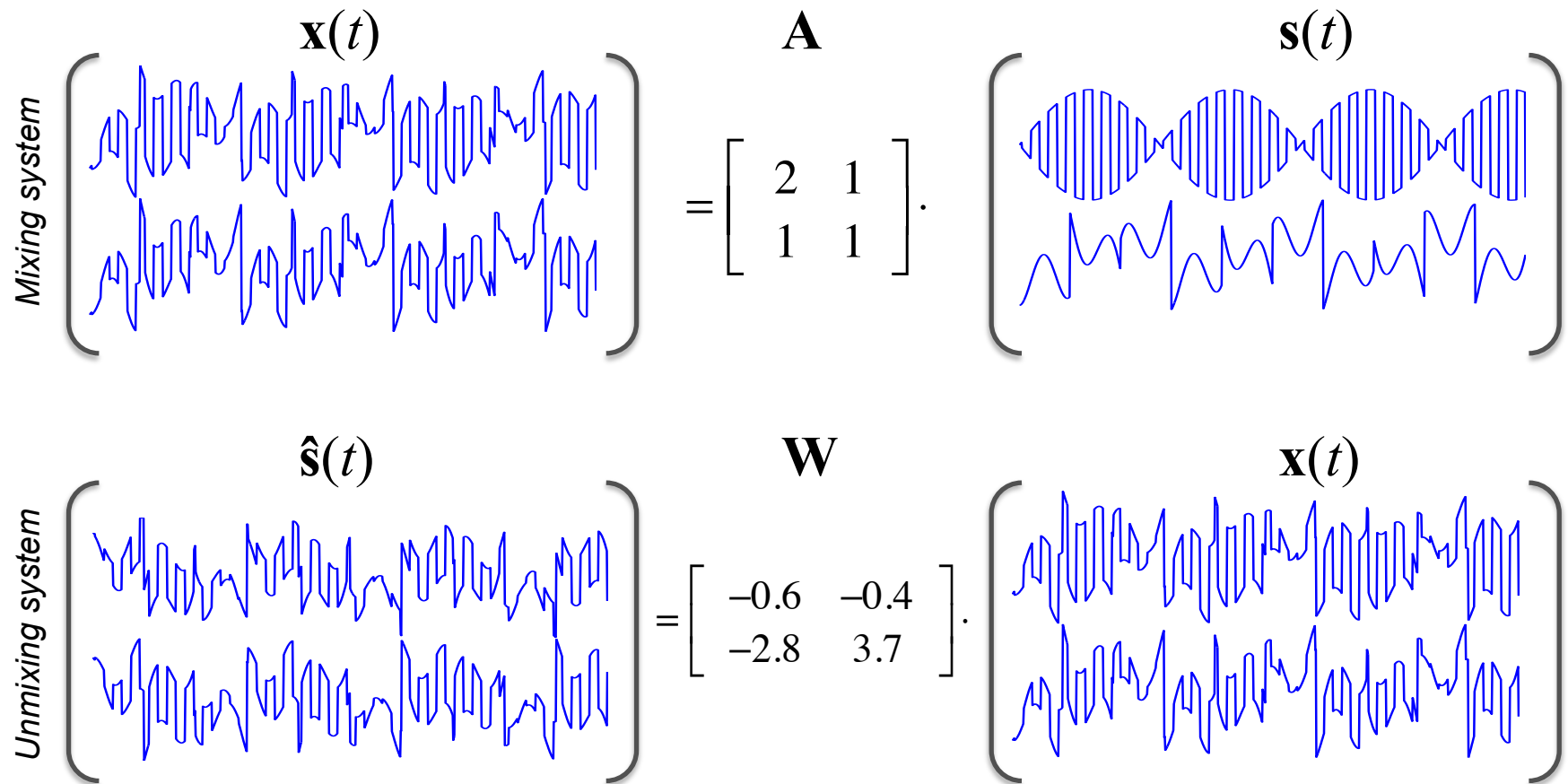
- We can algebraically recover an uncorrelated output using

$$\hat{\mathbf{S}} = \mathbf{W} \cdot \mathbf{X}$$

- If \mathbf{W} is the eigenvector matrix of $\text{Cov}(\mathbf{X})$
 - Or with $\mathbf{W} = \text{Cov}(\mathbf{X})^{-1/2}$
- Or we can use an online estimator:

$$\Delta \mathbf{W} \propto \mu \left(\mathbf{I} - \mathbf{W} \cdot \mathbf{x}(t) \cdot \mathbf{x}(t)^T \cdot \mathbf{W}^T \right) \mathbf{W}$$

So how well does this work?

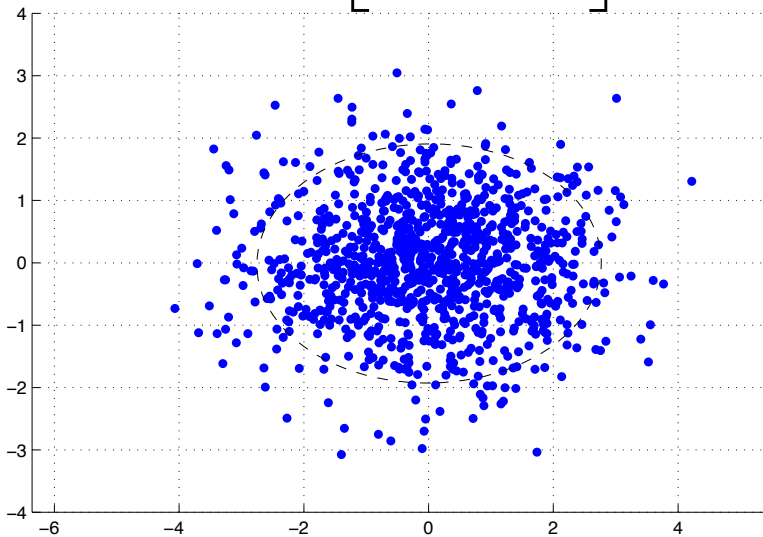


- Well, that was a waste of time ...

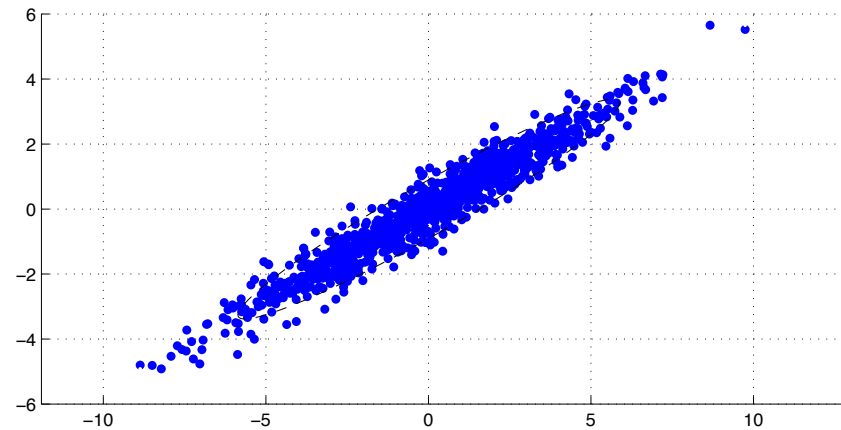
What went wrong?

- What does decorrelation mean?
 - That the two things compared are “not related”
- Consider a mixture of two Gaussians

$$\mathbf{s}(t) = \begin{bmatrix} \mathcal{N}(0,2) \\ \mathcal{N}(0,1) \end{bmatrix}$$



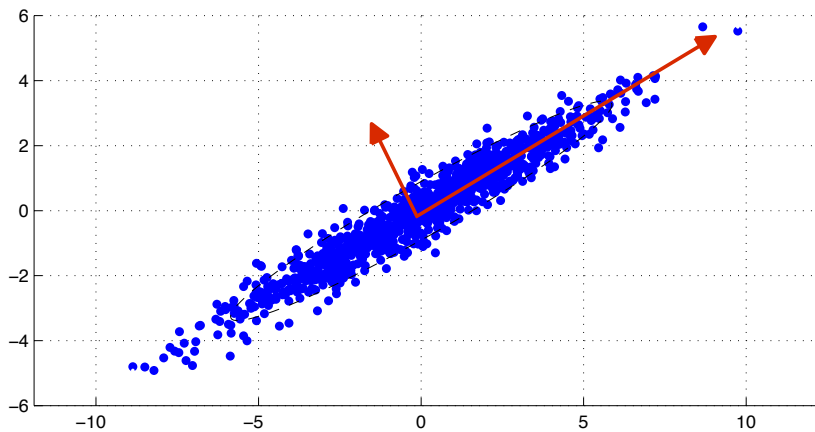
$$\mathbf{x}(t) = \begin{bmatrix} 2 & 1 \\ 1 & 1 \end{bmatrix} \cdot \mathbf{s}(t)$$



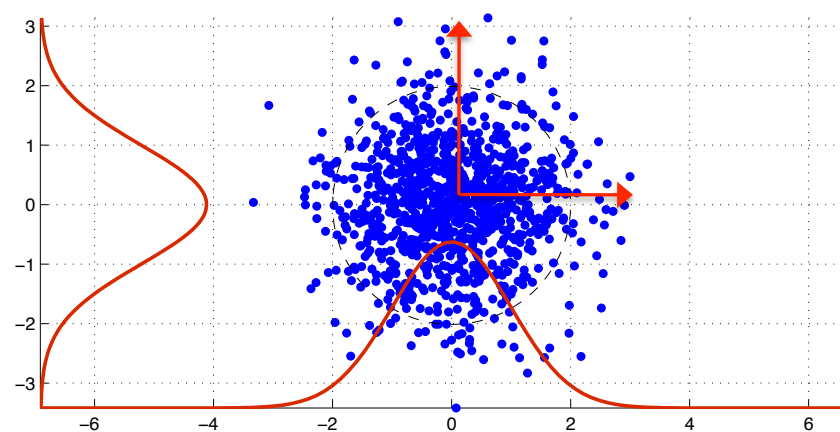
Decorrelation

- Now let us do what we derived so far on this signal:

a) Find the eigenvectors

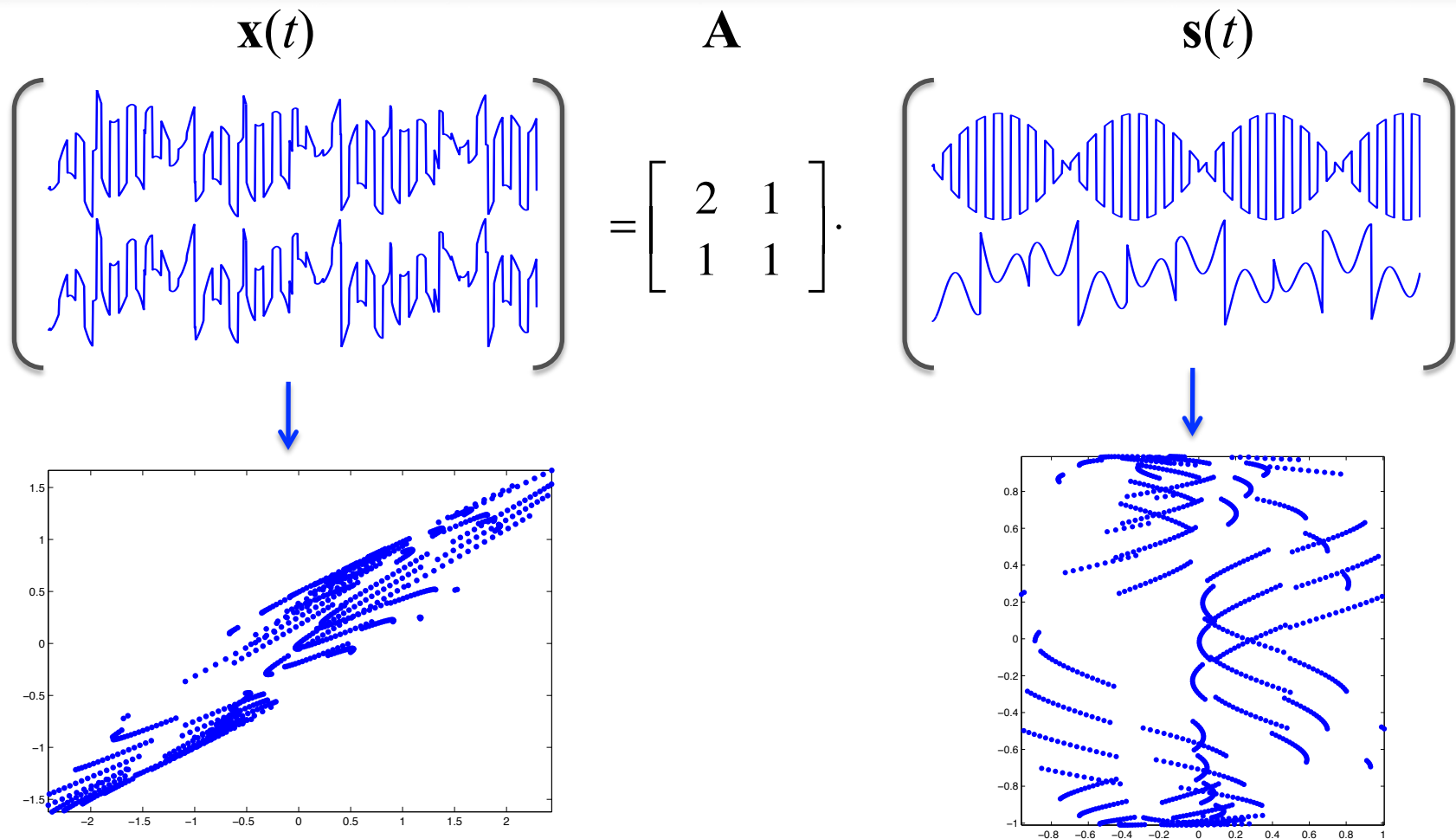


b) Rotate and scale so that covariance is \mathbf{I}



- After we are done the two Gaussians are “*statistically independent*”
 - i.e., $P(s_1, s_2) = P(s_1)P(s_2)$
- We have in effect separated the original signals
 - Save for a scaling ambiguity

Now lets try this on the original data

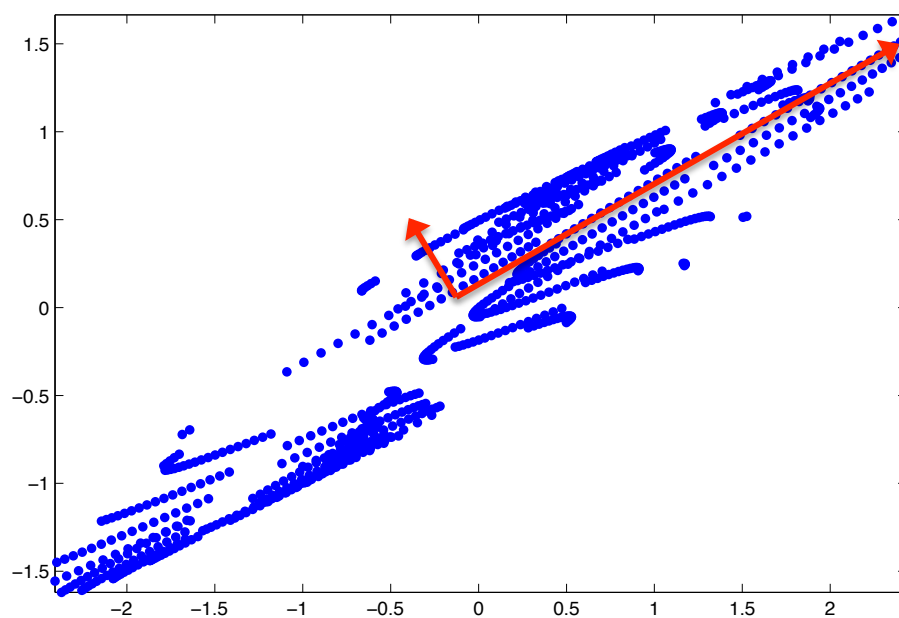


- Stating the obvious: These are not very Gaussian signals!!

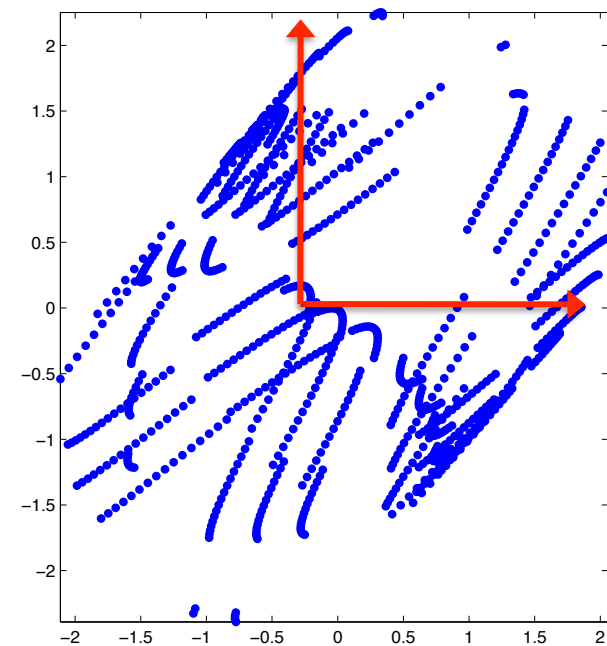
Doing PCA doesn't give us the right solution

- The result is not what we want
 - We are off by a rotation
- This idea doesn't seem to work for non-Gaussian signals

a) Find the eigenvectors



b) Rotate and scale so that covariance is \mathbf{I}



So what's wrong?

- For Gaussian data decorrelation means independence
 - Gaussians have up to second order statistics (1st is mean, 2nd is variance)
 - By minimizing the 2nd-order cross-statistics we achieve independence
 - These statistics can be expressed by the 2nd-order cumulants:

$$\text{cum}(x_i, x_j) = \langle x_i x_j \rangle$$

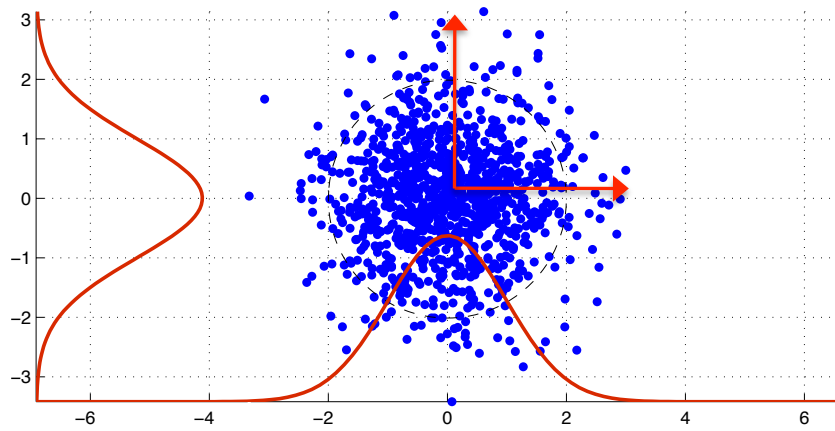
- Which happen to be the diagonals of the covariance matrix
- But real-world data are seldom Gaussian
 - Non-Gaussian data have higher orders which are not taken care of with PCA
 - We can measure their dependence using higher order cumulants:

$$3^{\text{rd}} \text{ order: } \text{cum}(x_i, x_j, x_k) = \langle x_i x_j x_k \rangle$$

$$4^{\text{th}} \text{ order: } \text{cum}(x_i, x_j, x_k, x_l) = \langle x_i x_j x_k x_l \rangle - \langle x_i x_j \rangle \langle x_k x_l \rangle - \langle x_i x_k \rangle \langle x_j x_l \rangle - \langle x_i x_l \rangle \langle x_k x_j \rangle$$

Cumulants for Gaussian vs non-Gaussian case

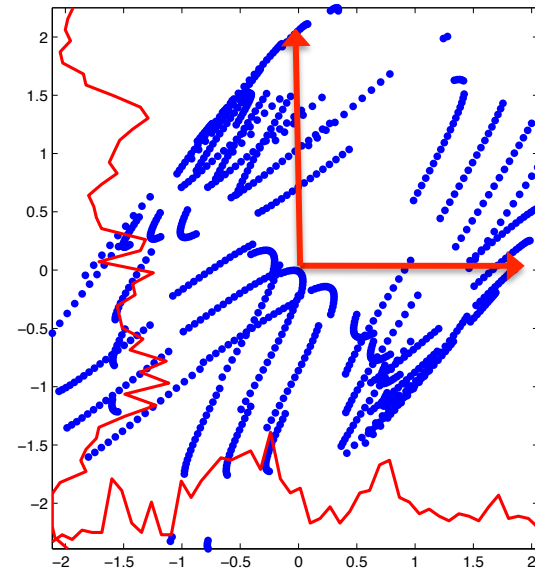
Gaussian case



Cross-cumulants tend to zero

$cum(x_i, x_j)$	$1e^{-13}$
$cum(x_i, x_j, x_k)$	0.0008, 0.0004
$cum(x_i, x_j, x_k, x_l)$	-0.003, 0.0005, 0.0007

Non-Gaussian case



Only 2nd order cross-cumulants tend to zero

$cum(x_i, x_j)$	$-4e^{-14}$
$cum(x_i, x_j, x_k)$	-0.08, -0.1
$cum(x_i, x_j, x_k, x_l)$	0.42, -0.3, 0.16

The real problem to solve

- For statistical independence we need to minimize all cross-cumulants
 - In practice up to 4th order is enough
- For 2nd order we minimized the **off-diagonal** covariance elements

$$\begin{bmatrix} cum(x_1, x_1) & cum(x_1, x_2) \\ cum(x_2, x_1) & cum(x_2, x_2) \end{bmatrix}$$

- For 4th order we will do the same for a *tensor*

$$Q_{i,j,k,l} = cum(x_i, x_j, x_k, x_l)$$

- The process is similar to PCA, but in more dimensions
 - We now find “*eigenmatrices*” instead of eigenvectors
- Algorithms like JADE and FOBI solve this problem
 - Can you see a potential problem though?

An alternative approach

- Tensorial methods can be very very computationally intensive
- How about an on-line method instead?
- Independence can also be coined as “*non-linear decorrelation*”
 - x and y are independent if and only if:

$$\langle f(x)g(y) \rangle = \langle f(x) \rangle \langle g(y) \rangle$$

- For all continuous functions f and g
 - This is a non-linear extension of 2nd order independence where $f(x) = g(x) = x$
- We can try solving for that then

Online ICA

- Conceptually this is very similar to online decorrelation

- For decorrelation:

$$\Delta \mathbf{W} \propto \mu \left(\mathbf{I} - \mathbf{W} \cdot \mathbf{x}(t) \cdot \mathbf{x}(t)^T \cdot \mathbf{W}^T \right) \mathbf{W}$$

- For non-linear decorrelation:

$$\Delta \mathbf{W} \propto \mu \left(\mathbf{I} - f(\mathbf{W} \cdot \mathbf{x}(t)) \cdot g(\mathbf{W} \cdot \mathbf{x}(t))^T \right) \mathbf{W}$$

- This adaptation method is known as the Cichocki-Unbehauen update

- But we can obtain it using many different ways

- But how do we pick the non-linearities?

- Depends on the prior we have on the sources

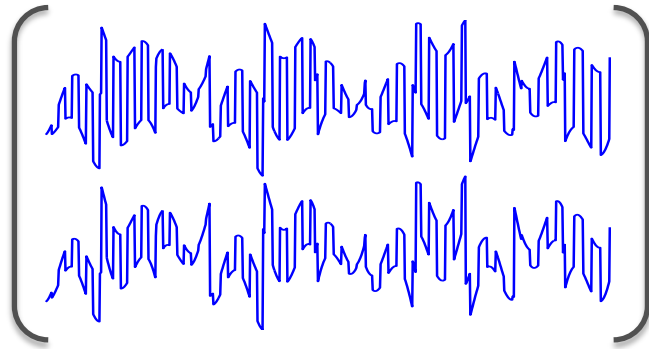
$$f(x_i) = \begin{cases} x + \tanh(x), & \text{for super-Gaussians} \\ x - \tanh(x), & \text{for sub-Gaussians} \end{cases}$$

Other popular approaches

- **Minimum Mutual Information**
 - Minimize the mutual information of the output
 - Creates maximally statistically independent outputs
- **Infomax**
 - Maximize the entropy of the output or Mutual Information of input/output
- **Non-Gaussianity**
 - Adding signals tends towards Gaussianity (Central Limit Theorem)
 - Find the maximally non-Gaussian outputs undoes the mixing
- **Maximum Likelihood**
 - Less straightforward at first, but elegant nevertheless
- **Geometric methods**
 - Trying to “eyeball” the proper way to rotate

Trying this on our dataset

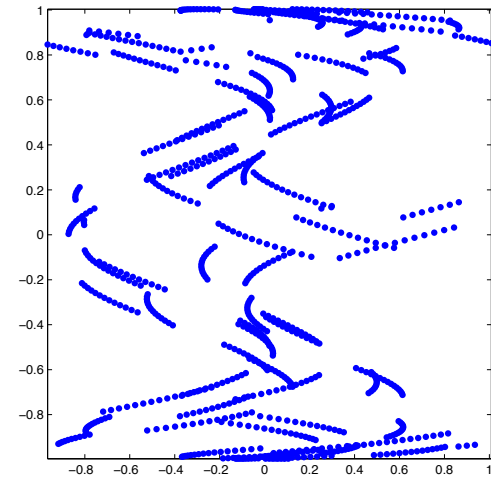
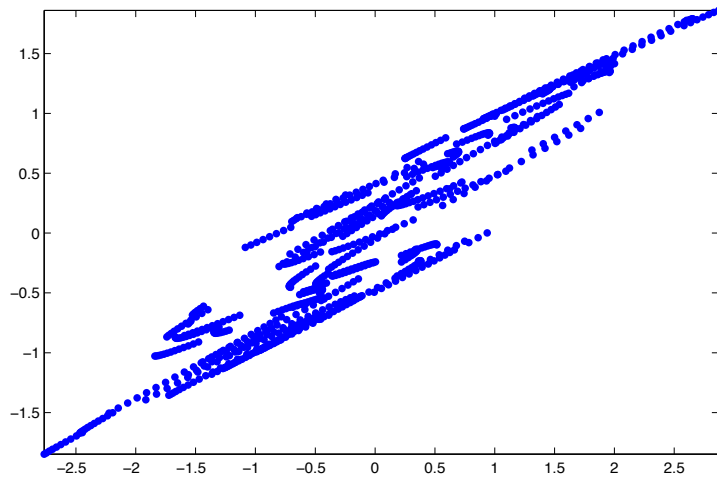
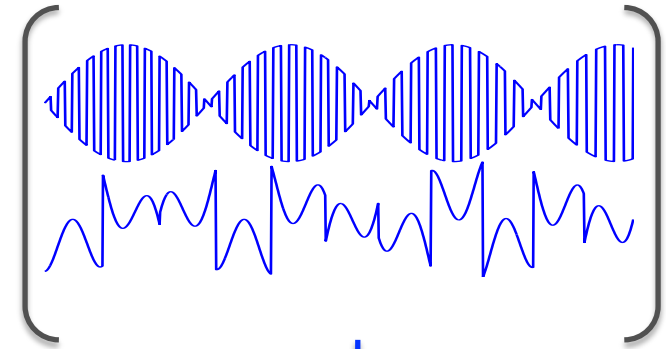
$\mathbf{x}(t)$



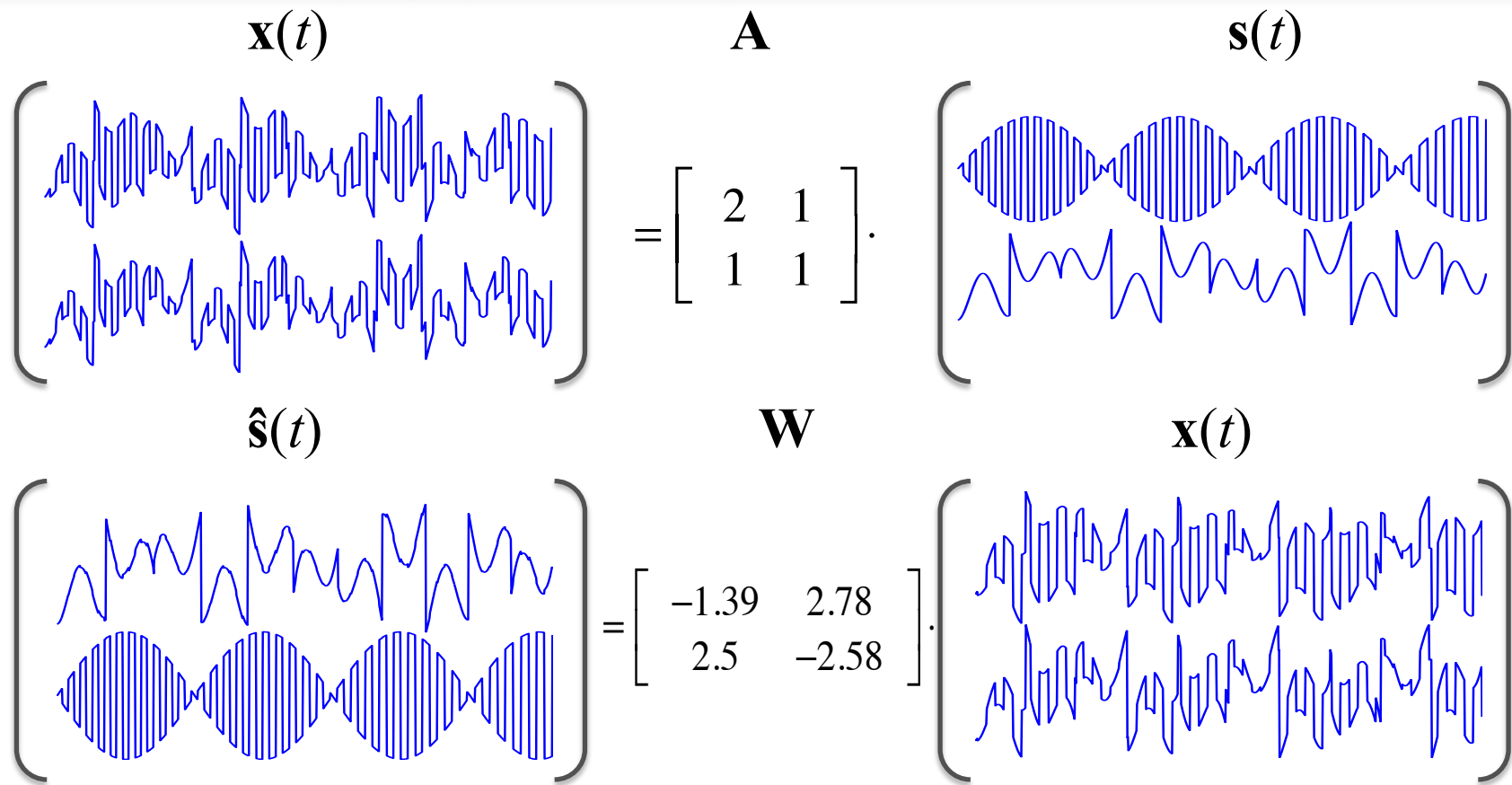
\mathbf{A}

$$= \begin{bmatrix} 2 & 1 \\ 1 & 1 \end{bmatrix}.$$

$\mathbf{s}(t)$

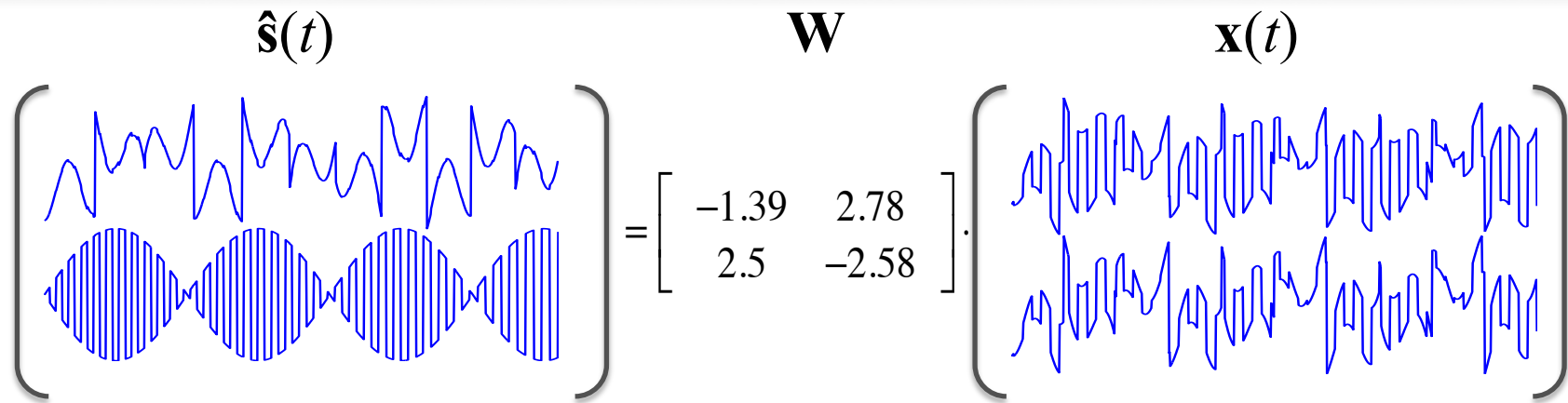


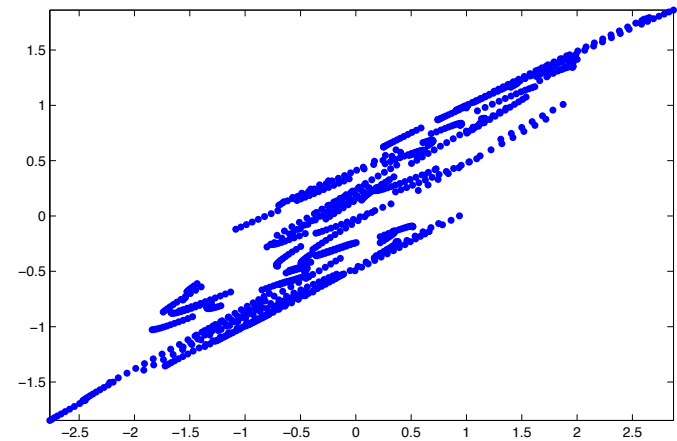
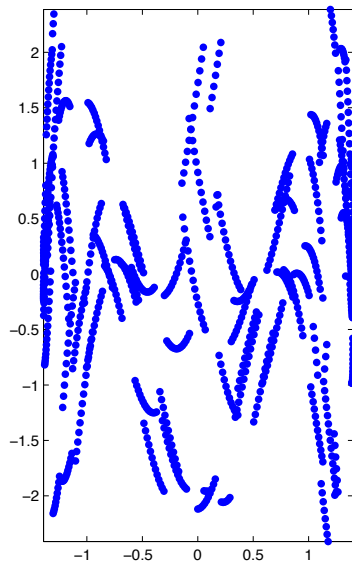
Trying this on our dataset



- We actually separated the mixture!

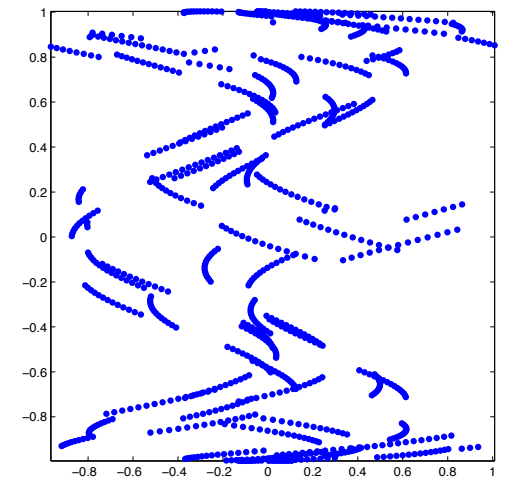
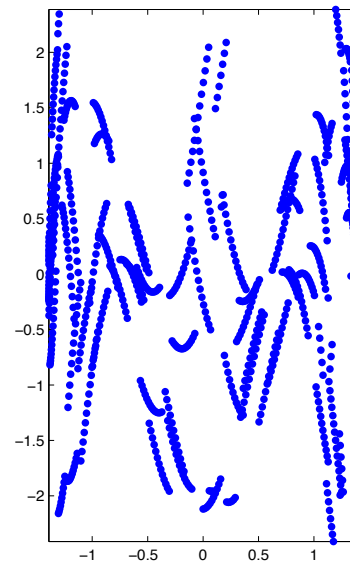
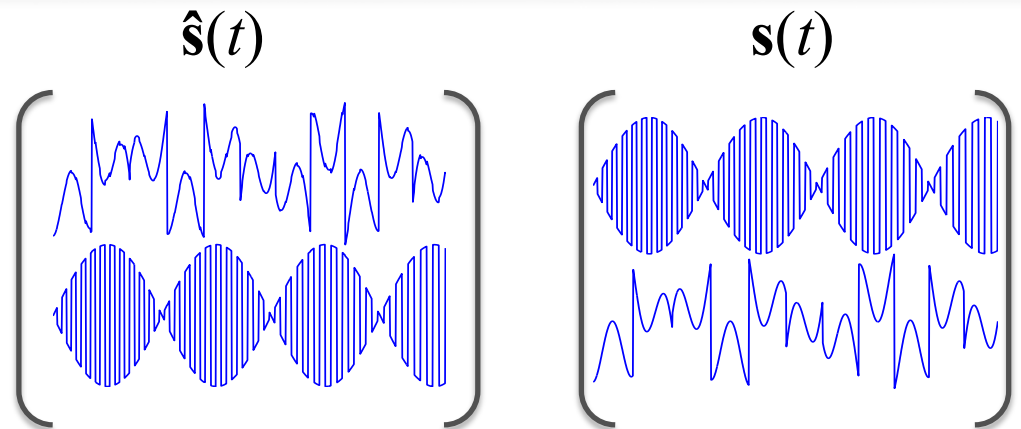
Trying this on our dataset

$$\hat{\mathbf{s}}(t) = \mathbf{W} \mathbf{x}(t)$$


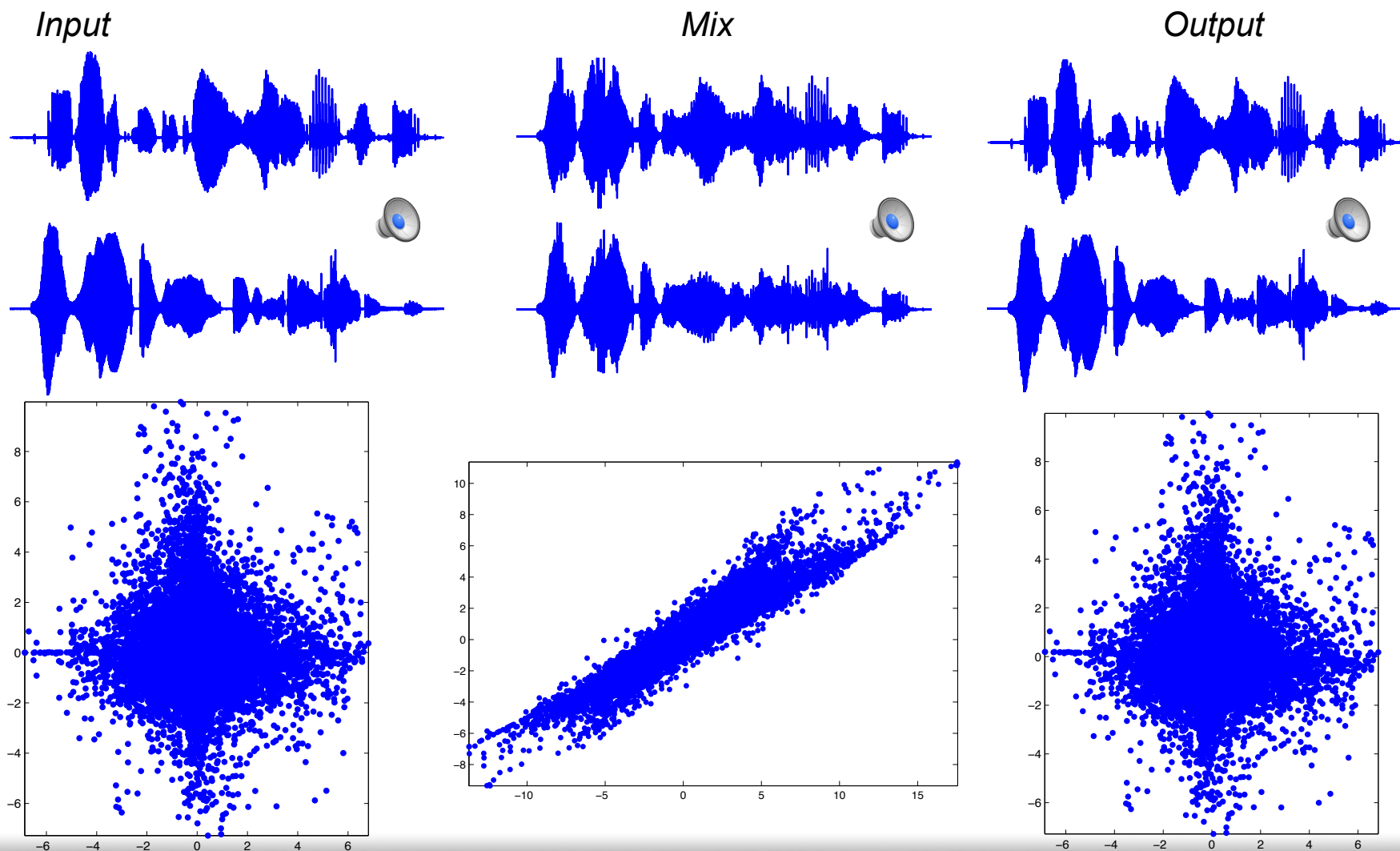


But something is amiss ..

- There are some things that ICA will not resolve
- Scale
 - Statistical independence is invariant of scale (and sign)
- Order of inputs
 - Order of inputs is irrelevant when talking about independence
- ICA will actually recover:
$$\hat{\mathbf{s}}(t) = \mathbf{D} \cdot \mathbf{P} \cdot \mathbf{s}(t)$$
- Where \mathbf{D} is diagonal and \mathbf{P} is a permutation matrix

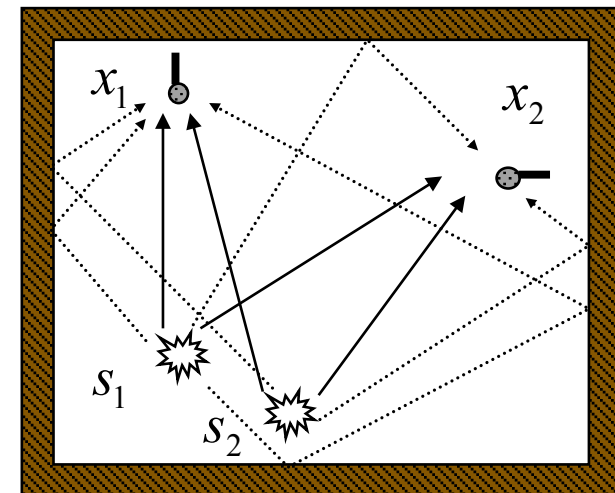


This works really well for audio mixtures!



Problems with instantaneous mixing

- Sounds don't really mix instantaneously
- There are multiple effects
 - Room reflections
 - Sensor response
 - Propagation delays
 - Propagation and reflection filtering
- Most can be seen as filters
- We need a *convolutive mixing* model



*Estimated sources using
the instantaneous model
on convolutive mix*

Convolutional mixing

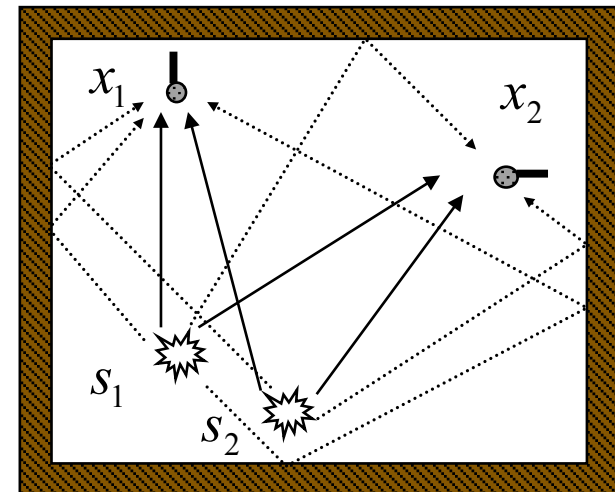
- Instead of instantaneous mixing:

$$x_i(t) = \sum_{j=1} a_{ij} s_j(t)$$

- We now have *convolutional* mixing:

$$x_i(t) = \sum_j \sum_k a_{ij}(k) s_j(t - k)$$

- The mixing filters $a_{ij}(k)$ encapsulate all the mixing effects in this model
- But how do we do ICA now?
 - This is an ugly equation!



FIR matrix algebra

- Matrices with FIR filters as elements

$$\underline{\mathbf{A}} = \begin{bmatrix} \underline{a}_{11} & \underline{a}_{12} \\ \underline{a}_{21} & \underline{a}_{22} \end{bmatrix}$$

$$\underline{a}_{ij} = \left[a_{ij}(0) \quad \cdots \quad a_{ij}(k-1) \right]$$

- FIR matrix multiplication performs convolution and accumulation

$$\underline{\mathbf{A}} \cdot \underline{\mathbf{b}} = \begin{bmatrix} \underline{a}_{11} & \underline{a}_{12} \\ \underline{a}_{21} & \underline{a}_{22} \end{bmatrix} \cdot \begin{bmatrix} \underline{b}_1 \\ \underline{b}_2 \end{bmatrix} = \begin{bmatrix} \underline{a}_{11} * \underline{b}_1 + \underline{a}_{12} * \underline{b}_2 \\ \underline{a}_{21} * \underline{b}_1 + \underline{a}_{22} * \underline{b}_2 \end{bmatrix}$$

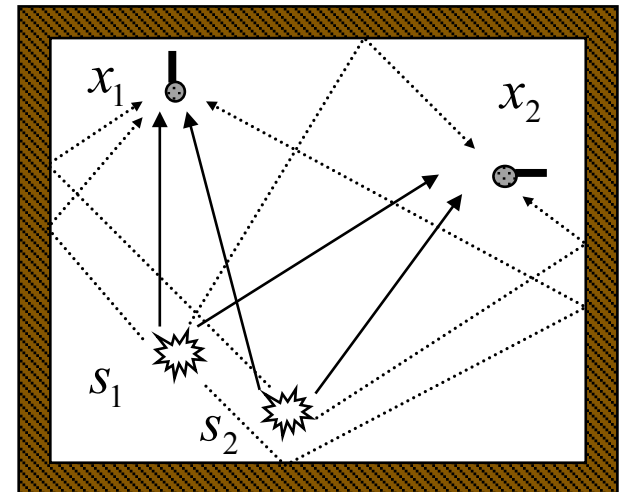
Back to convolutive mixing

- Now we can rewrite convolutive mixing as:

$$x_i(t) = \sum_j \sum_k a_{ij}(k) s_j(t-k) \Rightarrow$$

$$\Rightarrow \underline{\mathbf{x}}(t) = \underline{\mathbf{A}} \cdot \underline{\mathbf{s}}(t) = \begin{bmatrix} a_{11} * s_1(t) + a_{12} * s_2(t) \\ a_{21} * s_1(t) + a_{22} * s_2(t) \end{bmatrix}$$

- Tidier formulation!
- We can use the FIR matrix abstraction to solve this problem now



An easy way to solve convolutive mixing

- Straightforward translation of instantaneous learning rules using FIR matrices:

$$\Delta \underline{\mathbf{W}} \propto \left(\mathbf{I} + f(\underline{\mathbf{W}} \cdot \underline{\mathbf{x}}) \cdot (\underline{\mathbf{W}} \cdot \underline{\mathbf{x}})^T \right) \cdot \underline{\mathbf{W}}$$

- Not so easy with algebraic approaches!
- Multiple other (and more rigorous/better behaved) approaches have been developed

Complications with this approach

- Required convolutions are expensive
 - Real-room filters are long
 - Their FIR inverses are very long
 - FIR products can become very time consuming
- Convergence is hard to achieve
 - Huge parameter space
 - Tightly interwoven parameter relationships
- A slow optimization nightmare!

FIR matrix algebra, part II

- FIR matrices have frequency domain counterparts:

$$\underline{\mathbf{A}} = \begin{bmatrix} \underline{a}_{11} & \underline{a}_{12} \\ \underline{a}_{21} & \underline{a}_{22} \end{bmatrix} \xrightarrow{\text{frequency domain}} \hat{\underline{\mathbf{A}}} = \begin{bmatrix} \hat{\underline{a}}_{11} & \hat{\underline{a}}_{12} \\ \hat{\underline{a}}_{21} & \hat{\underline{a}}_{22} \end{bmatrix}$$
$$\hat{\underline{a}}_{ij} = \text{DFT}[\underline{a}_{ij}]$$

- And their products are simpler:

$$\hat{\underline{\mathbf{A}}} \cdot \hat{\underline{\mathbf{b}}} = \begin{bmatrix} \hat{\underline{a}}_{11} \cdot \hat{\underline{b}}_1 + \hat{\underline{a}}_{12} \cdot \hat{\underline{b}}_2 \\ \hat{\underline{a}}_{21} \cdot \hat{\underline{b}}_1 + \hat{\underline{a}}_{22} \cdot \hat{\underline{b}}_2 \end{bmatrix}$$

$$\hat{\underline{a}} \cdot \hat{\underline{b}} = [a(0) \cdot b(0) \quad \cdots \quad a(k-1) \cdot b(k-1)]$$

Yet another convolutive mixing formulation

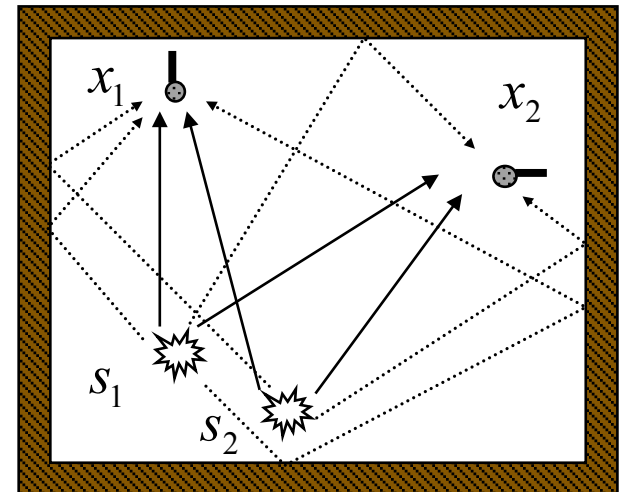
- We can now model the process in the frequency domain:

$$\underline{\hat{\mathbf{X}}} = \underline{\hat{\mathbf{A}}} \cdot \underline{\hat{\mathbf{S}}}$$

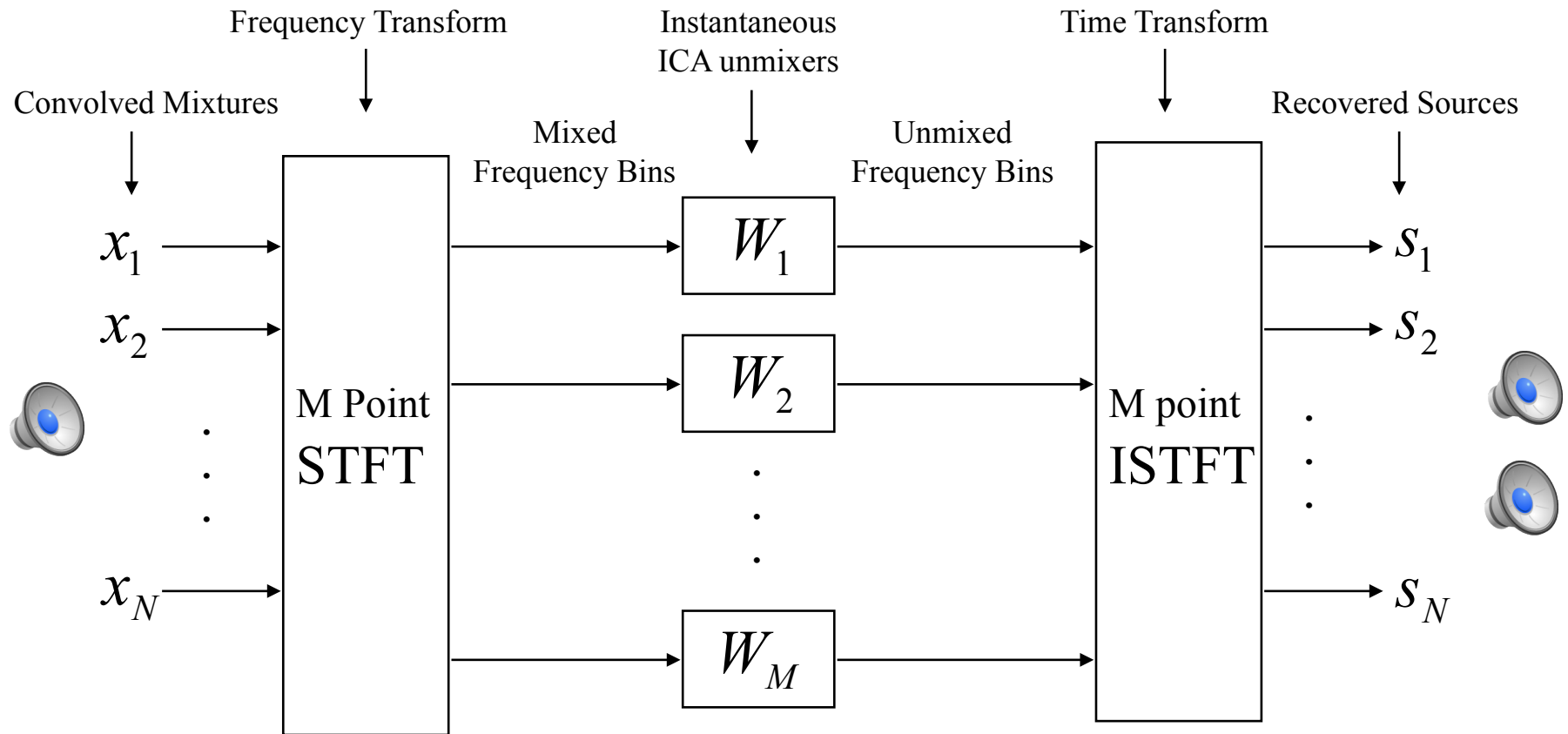
- For every frequency we have:

$$\mathbf{X}_f(t) = \mathbf{A}_f \cdot \mathbf{S}_f(t), \quad \forall f, t$$

- Hey, that's instantaneous mixing!
 - We can solve that!



Overall flowgraph



Some complications ...

- Permutation issues

- We don't know which source will end up in each narrowband output ...
- Resulting output can have separated narrowband elements from both sounds!

*Extracted source
with permutation*



- Scaling issues

- Narrowband outputs can be scaled arbitrarily
- This results in spectrally colored outputs

Original source



Colored source




Scaling issue


- One simple fix is to normalize the separating matrices

$$\mathbf{W}_f^{norm} = \mathbf{W}_f^{orig} \cdot \left| \mathbf{W}_f^{orig} \right|^{\frac{1}{N}}$$

- Results into more reasonable scaling
- More sophisticated approaches exist but this is not a major problem
- Some spectral coloration is however unavoidable

Original source 

Colored source 

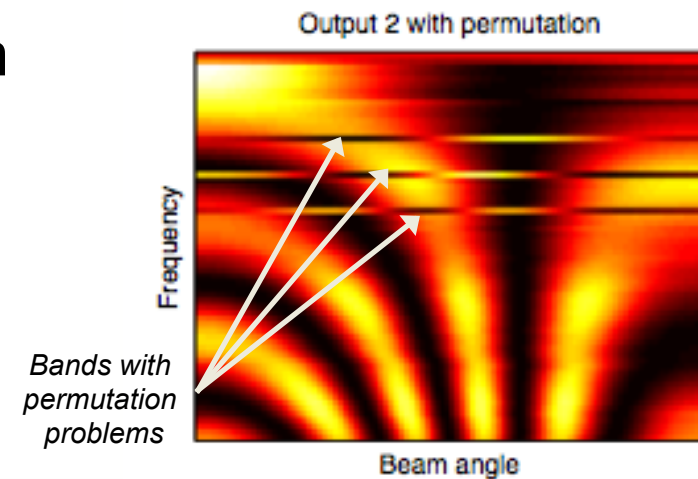
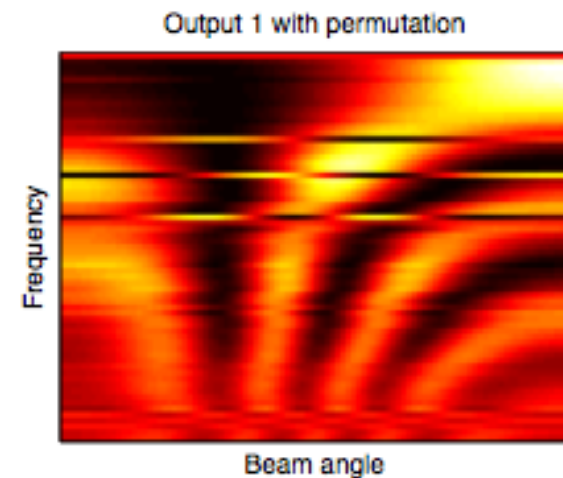
Corrected source 

Some solutions for permutation problems

- Continuity of unmixing matrices
 - Adjacent unmixing matrices tend to be a little similar, we can permute/bias them accordingly
 - Doesn't work that great
- Smoothness of spectral output
 - Narrowband components from each source tend to modulate the same way
 - Permute unmixing matrices to ensure adjacent narrowband output are similarly modulated
 - Works fine
- The above can fail miserably for more than two sources!
 - Combinatorial explosion!

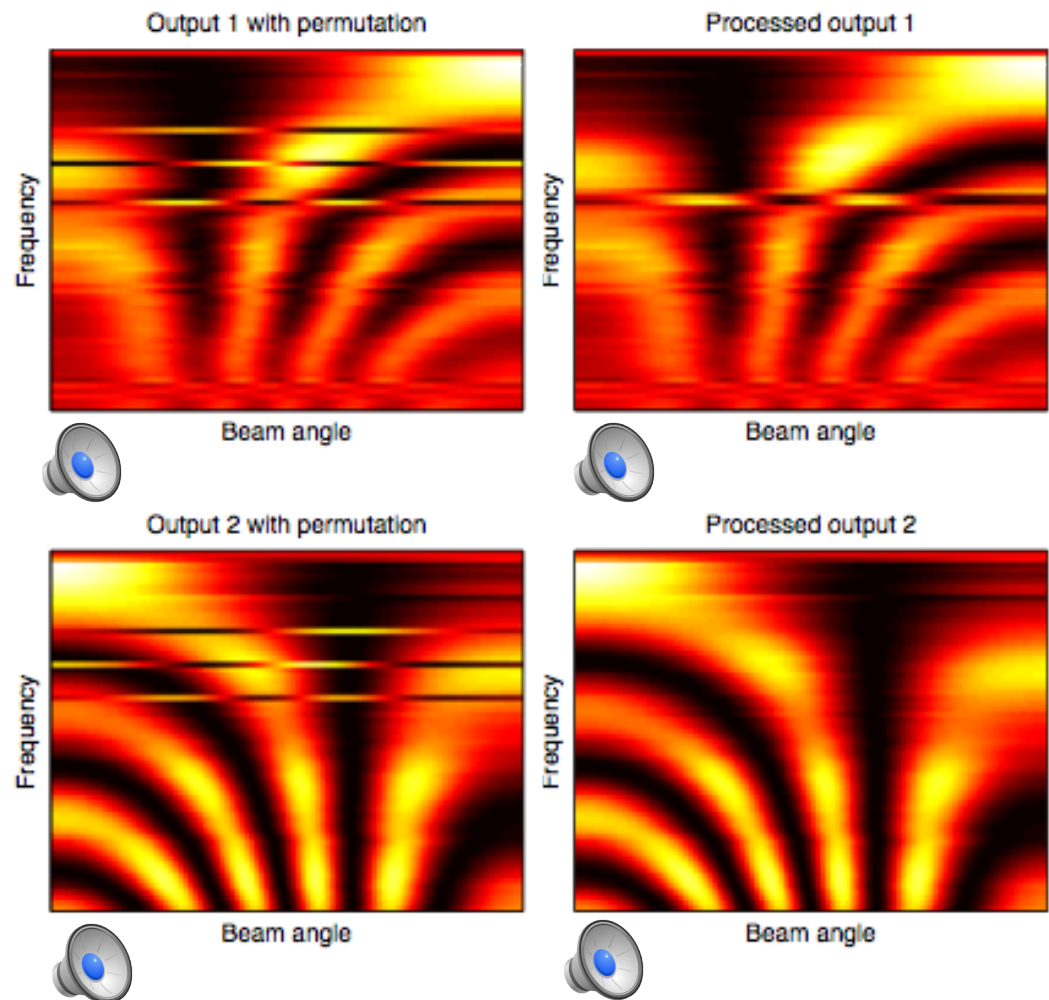
Beamforming and ICA

- If we know the placement of the sensors we can obtain the spatial response of the ICA solution
- ICA places nulls to cancel out interfering sources
 - Just as in the instantaneous case we cancel out sources
- We can visualize the permutation problem now
 - Out of place bands



Using beamforming to resolve permutations

- Spatial information can be used to resolve permutations
 - Find permutations that preserve zeros or smooth out the responses
- Works fine, although it can be flaky if the array response is not that clean



The N-input N-output problem

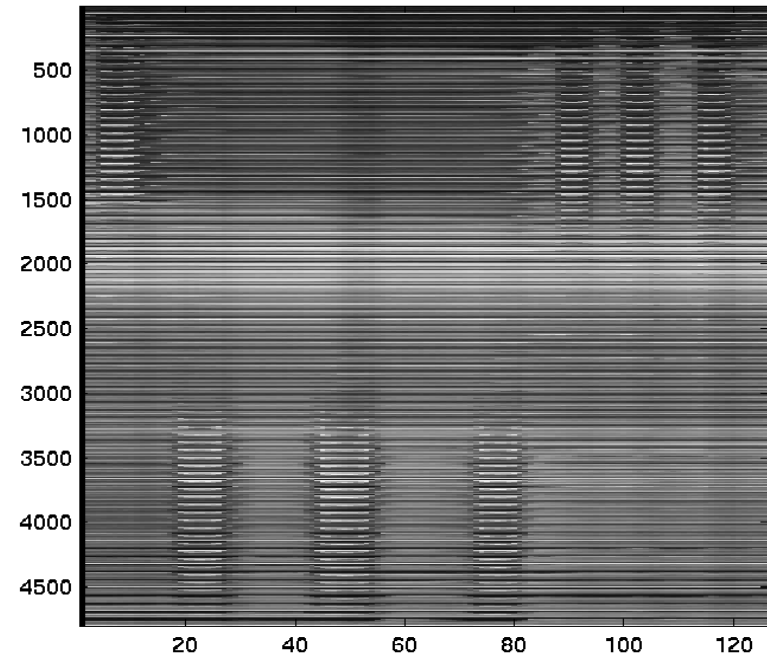
- ICA, in either formulation inverts a square matrix (whether scalar, or FIR)
 - This implies that we have the same number of input as outputs
 - E.g. in a street with 30 noise sources we need at least 30 mics!
- Solutions exist for M ins - N outs where $M > N$
- If $N > M$ we can only beamform
 - In some cases extra sources can be treated as noise
- This can be restrictive in some situations

Separation recap

- Orthogonality is not independence!!
 - Not all signals are Gaussian which is a usual assumption
- We can model instantaneous mixtures with ICA and get good results
 - ICA algorithms can optimize a variety of objectives, but ultimately result in statistical independence between the outputs
 - Same model is useful for all sorts of mixing situations
- Convolutional mixtures are more challenging but solvable
 - There's more ambiguity, and a closer link to signal processing approaches

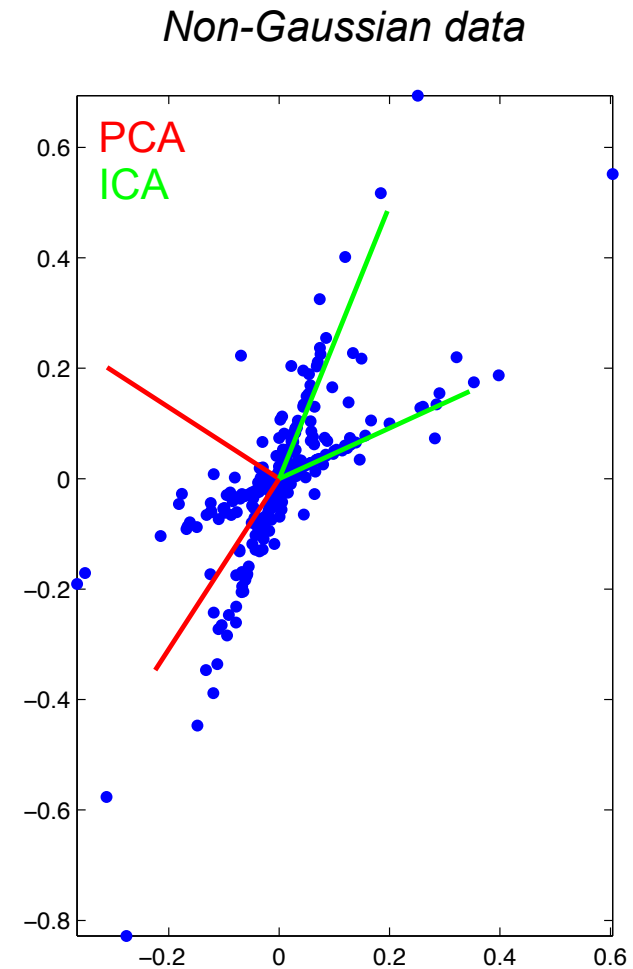
ICA for data exploration

- ICA is also great for data exploration
 - If PCA is, then ICA should be, right?
- With data of large dimensionalities we want to find structure
- PCA can reduce the dimensionality
 - And clean up the data structure a bit
- But ICA can find much more intuitive projections



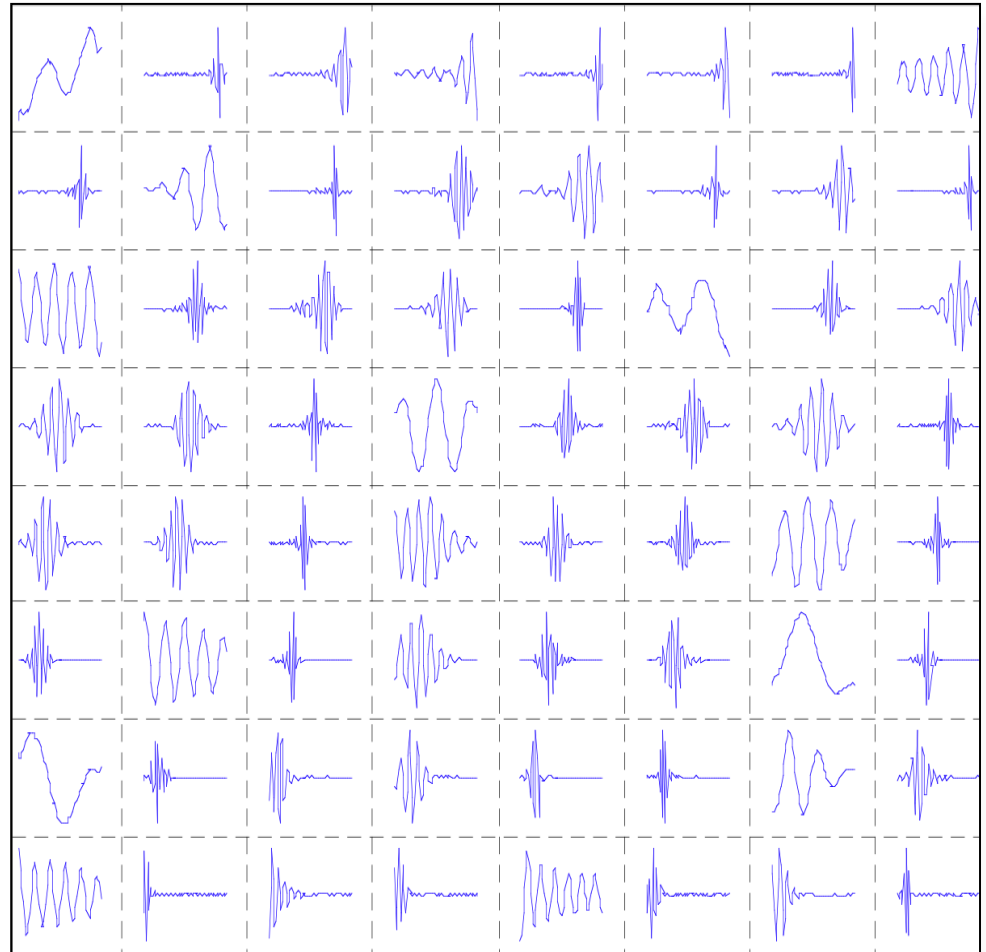
Example cases of PCA vs ICA

- Motivation for using ICA vs PCA
- PCA will indicate orthogonal directions of maximal variance
 - This is great for Gaussian data
 - Also great if we are into LS models
- Real-world is not Gaussian though
- ICA finds directions that are more “revealing”



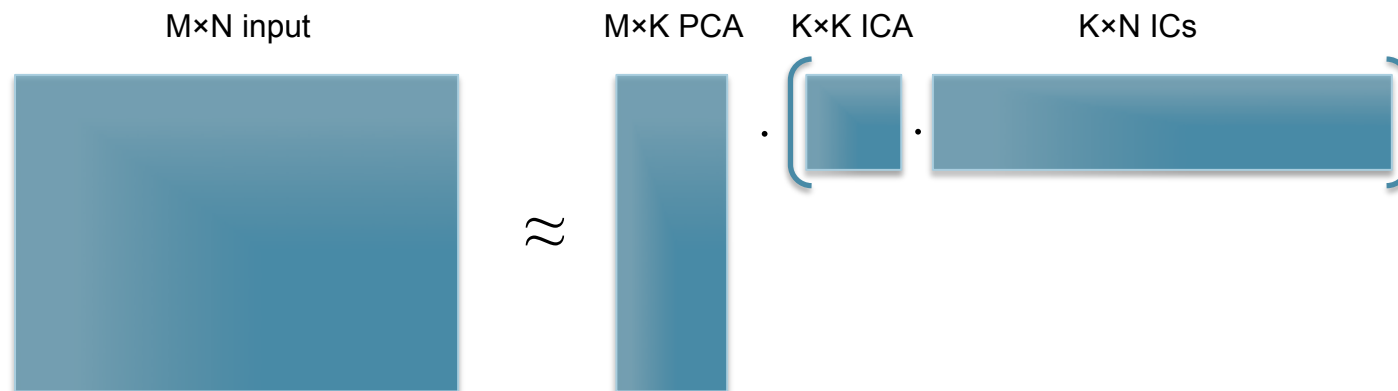
Finding useful transforms with ICA

- Audio preprocessing example
- Take a lot of audio snippets and concatenate them in a big matrix, do component analysis
- PCA results in the DCT bases
 - Do you see why?
- ICA returns time/freq localized sinusoids which is a better way to analyze sounds
- Ditto for images
 - ICA returns localizes edge filters



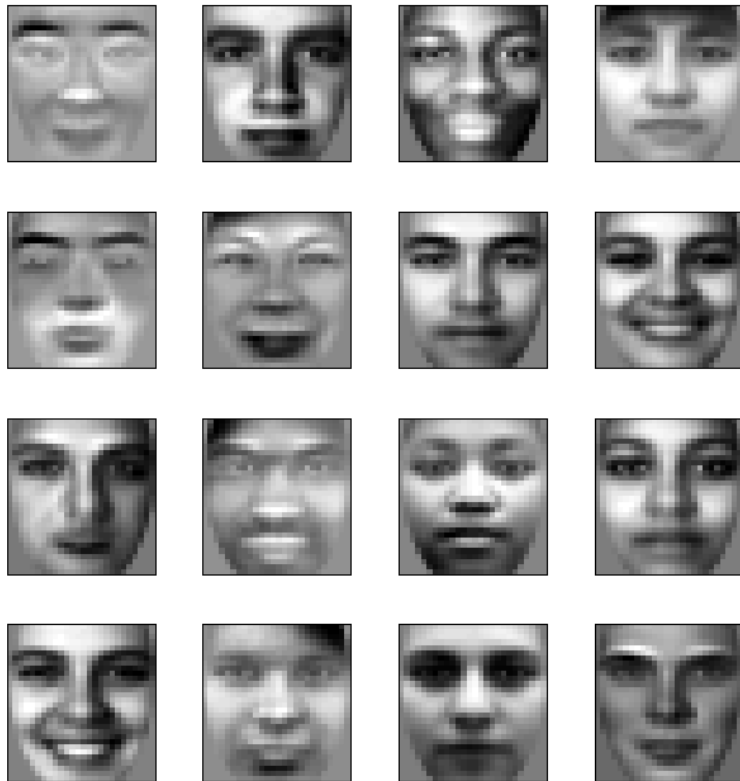
Enhancing PCA with ICA

- ICA cannot perform dimensionality reduction
 - The goal is to find independent components, hence there is no sense of order
- PCA does a great job at reducing dimensionality
 - Keeps the elements that carry most of the input's energy
- It turns out that PCA is a great preprocessor for ICA
 - There is no guarantee that the PCA subspace will be appropriate for the independent components but for most practical purposes this doesn't make a big difference



Example case: ICA-faces vs. Eigenfaces

ICA-faces

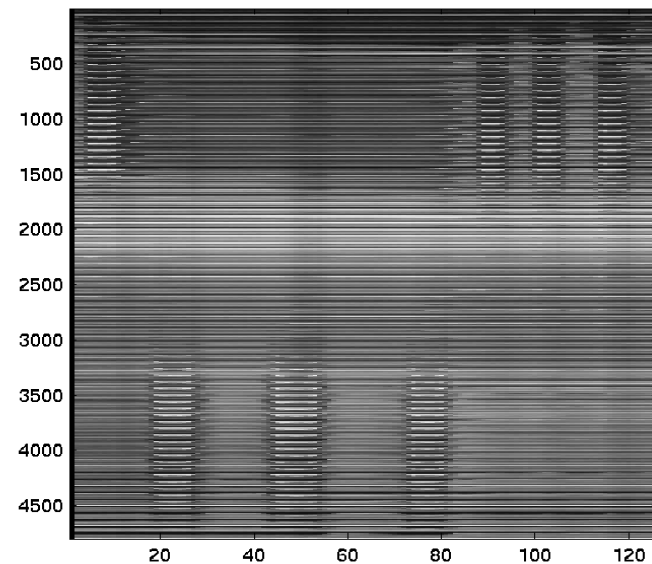


Eigenfaces



A Video Example

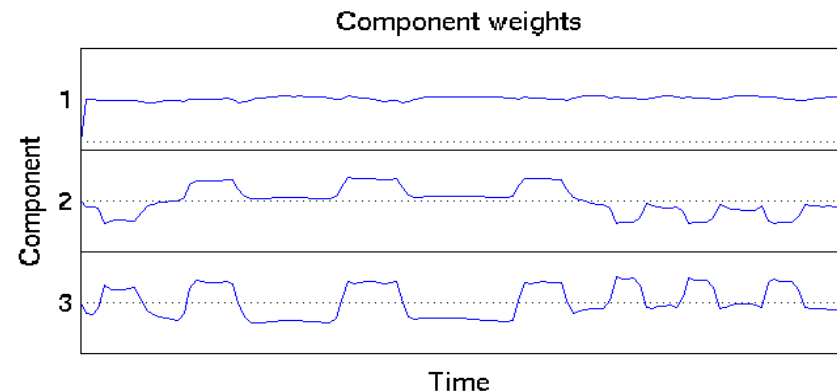
- The movie is a series of frames
 - Each frame is a data point
 - 126, 80×60 pixel frames
 - Data \mathbf{X} will be 4800×126
- Using PCA/ICA
 - $\mathbf{X} = \mathbf{W} \times \mathbf{H}$
 - \mathbf{W} will contain visual components
 - \mathbf{H} will contain their time weights



PCA Results

- Nothing special about the visual components
- They are orthogonal pictures
 - Does this mean anything? (not really ...)
 - Some segmentation of constant vs. moving parts
- Some highlighting of the action in the weights

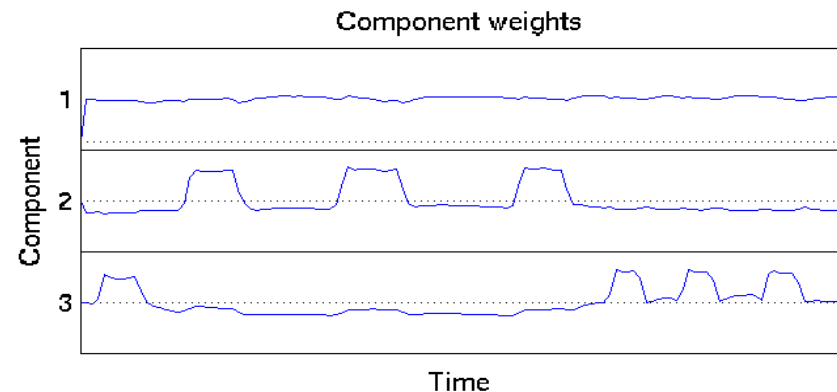
Video Component 1 Video Component 2 Video Component 3



ICA Results

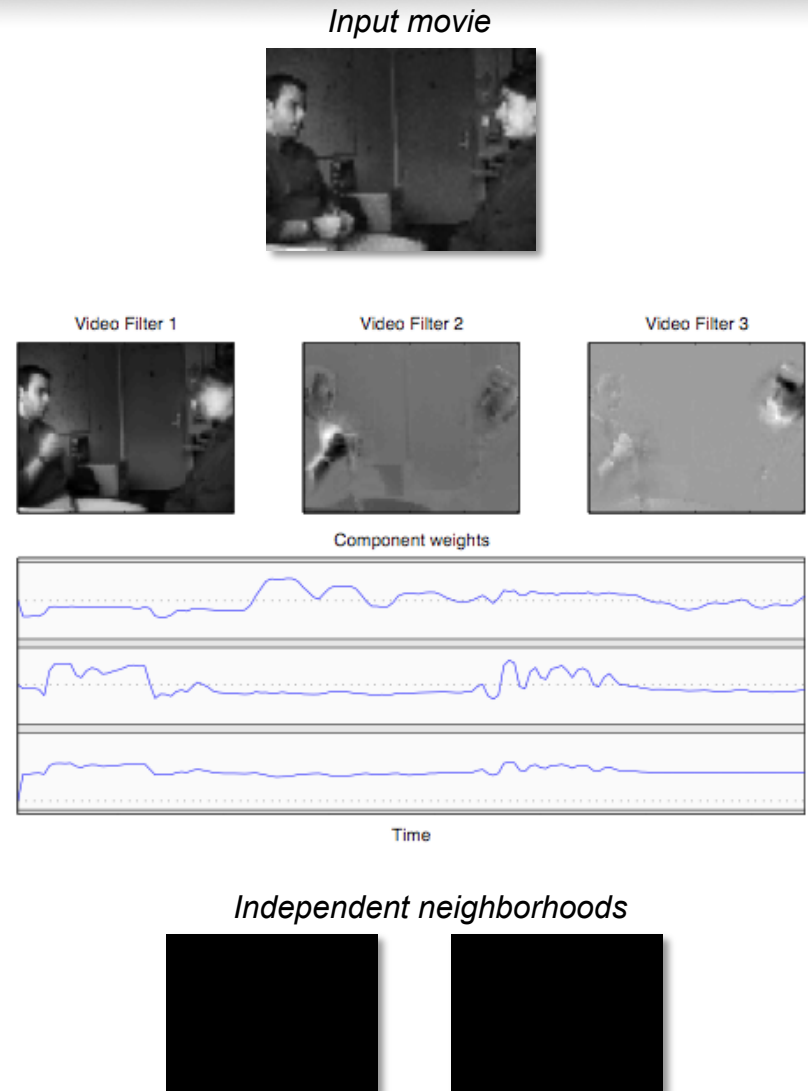
- Much more interesting visual components
- They are *independent*
 - Unrelated elements (left/ right hands, background) are now highlighted
 - We have some decomposition by parts
- Components weights are now describing the scene

Video Component 1 Video Component 2 Video Component 3



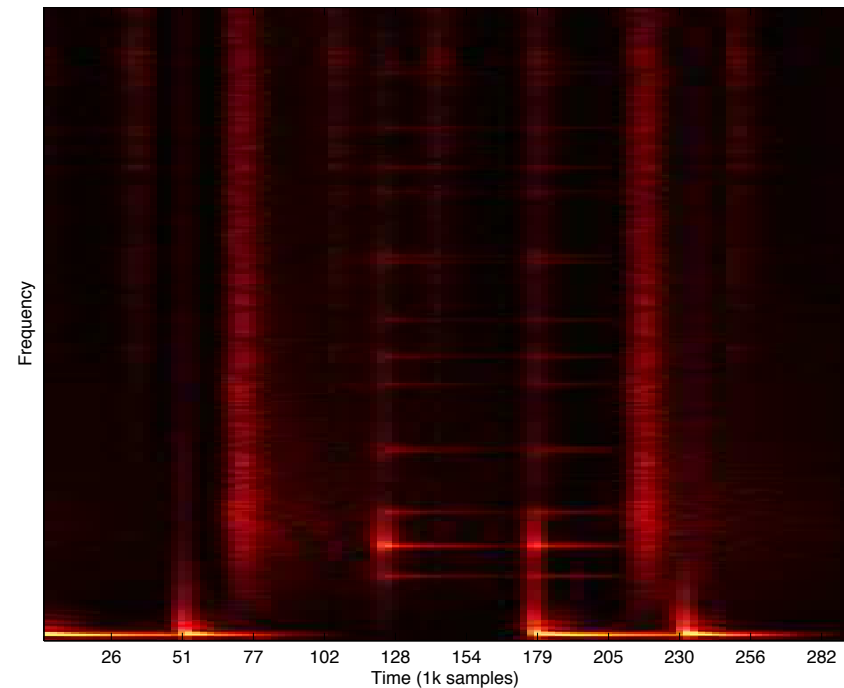
A Video Example

- The movie is a series of frames
 - Each frame is a data point
 - 315, 80×60 pixel frames
 - Data \mathbf{X} will be 4800×315
- Using PCA/ICA
 - $\mathbf{X} = \mathbf{W} \times \mathbf{H}$
 - \mathbf{W} will contain visual components
 - \mathbf{H} will contain their time weights



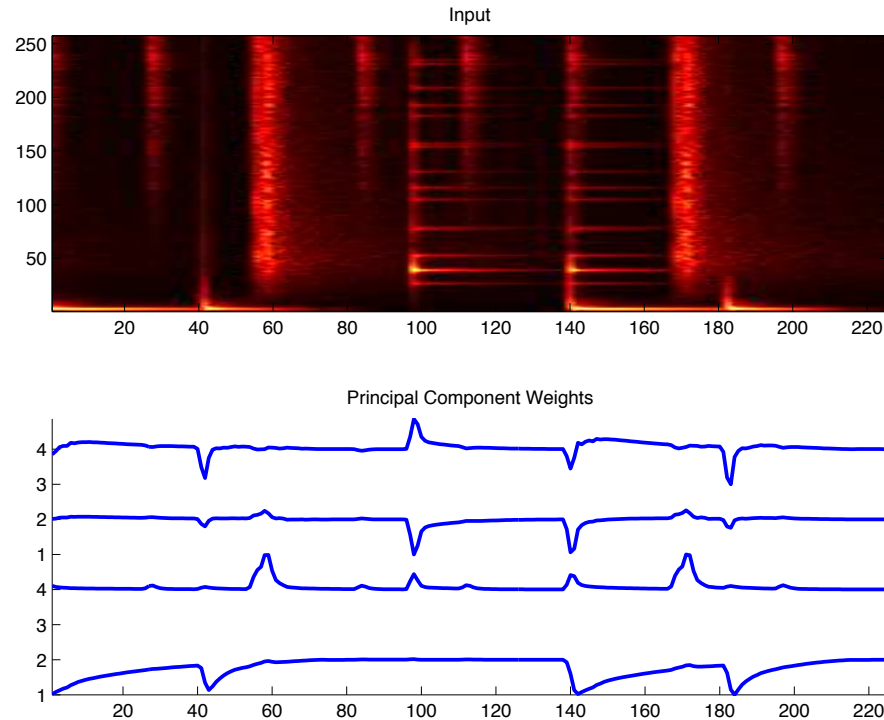
What about the soundtrack?

- We can also analyze audio in a similar way
- We do a frequency transform and get an audio spectrogram \mathbf{X}
 - \mathbf{X} is frequencies \times time
 - Distinct audio elements can be seen in \mathbf{X}
- Unlike before we have only one input this time



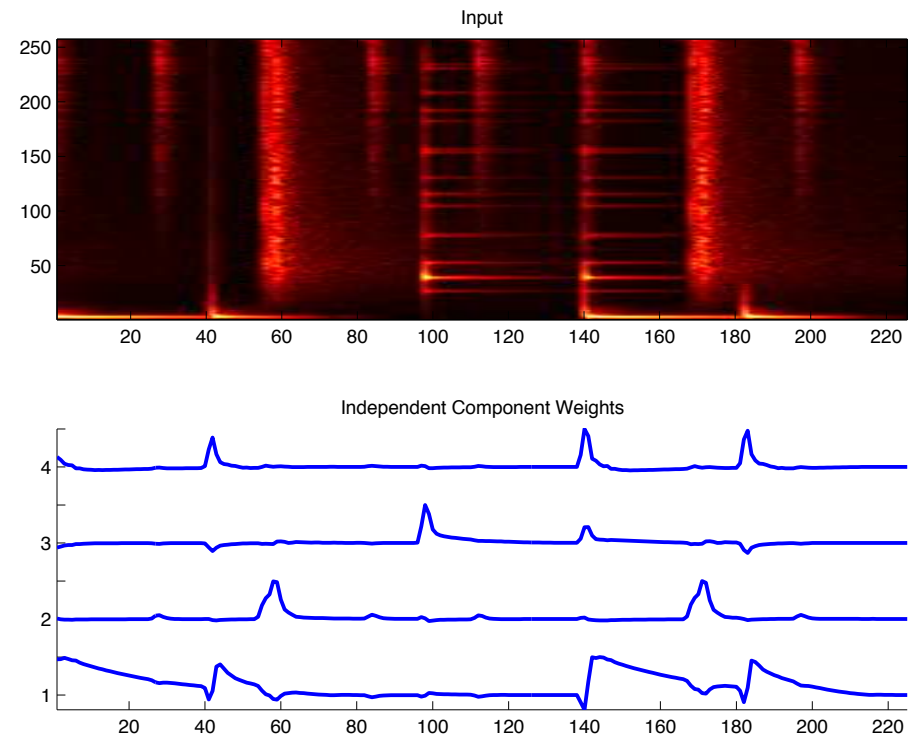
PCA on Audio

- Umm ... it sucks!
- Orthogonality doesn't mean much for audio components
 - Results are mathematically optimal, perceptually useless



ICA on Audio

- A definite improvement
- Independence helps pick up somewhat more meaningful sound objects
 - Not too clean results, but the intentions are clear
 - Misses some details



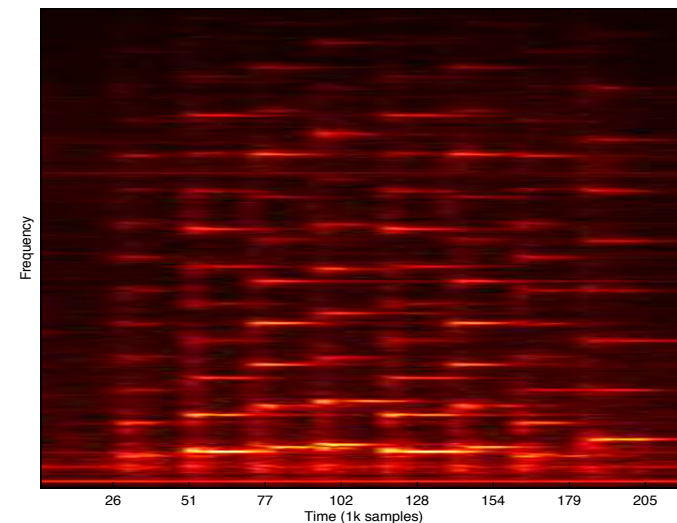
Audio Visual Components?

- We can can even take in both audio and video data and try to find structure
- Sometimes there is a very strong correlation between auditory and visual elements
- We should be able to discover that automatically

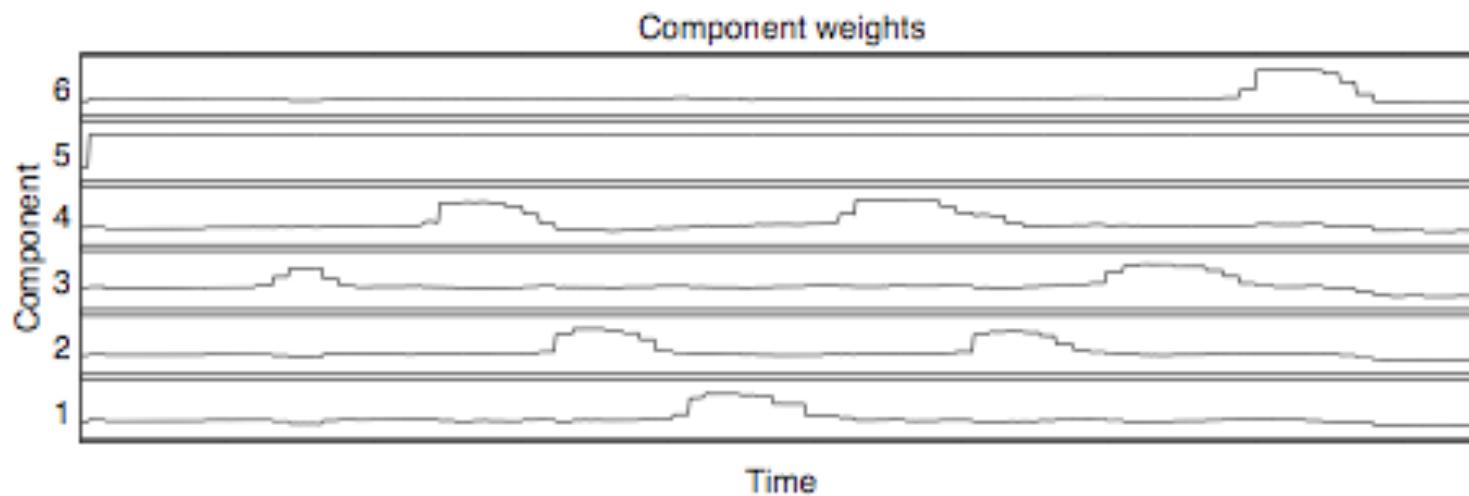
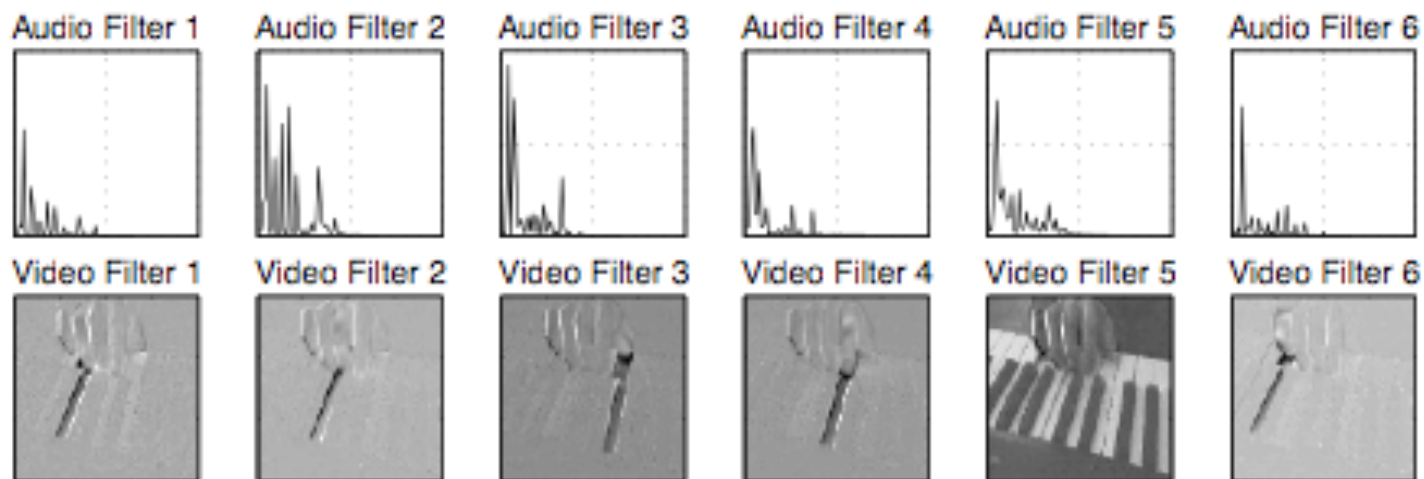
Input video



Input audio



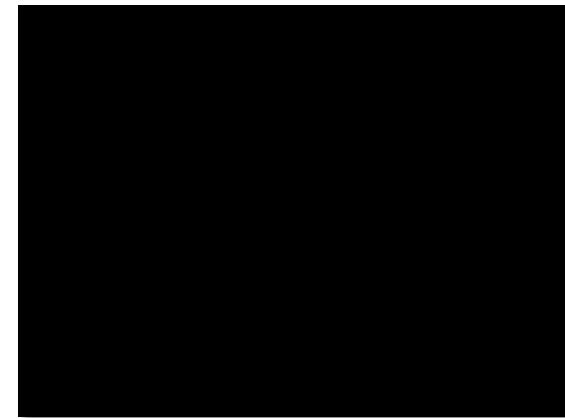
Audio/Visual Components



Which allows us to play with output

- And of course once we have such a nice description we can resynthesize at will

Resynthesis



Recap

- A motivating example
- The theory
 - Decorrelation
 - Independence vs decorrelation
 - Independent component analysis
- Separating sounds
 - Solving instantaneous mixtures
 - Solving convolutive mixtures
- Data exploration and independence
 - Extracting audio features
 - Extracting multimodal features