
Principal Component Analysis
(PCA)

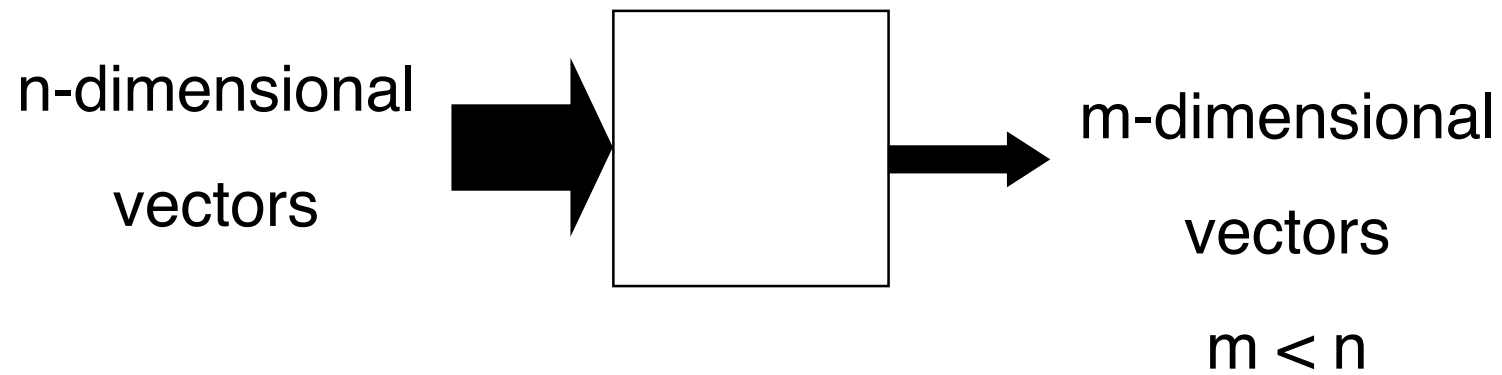
and

PCA Neural Networks:
An Application of Unsupervised Hebbian Learning
and intro to
Independent Components Analysis (ICA)

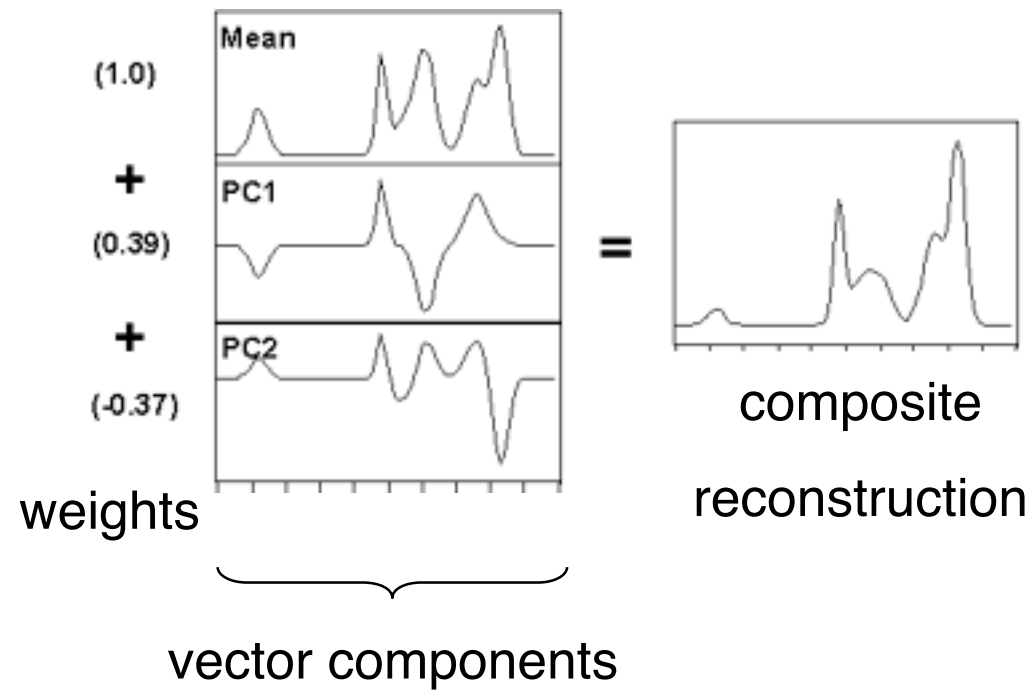
What is PCA?

- A standard statistical technique for reducing dimensionality of data (without using neural approach).
- Purpose: Better understanding or communication of data; used a lot in the sciences to select the most important features.
- In so reducing, we want to lose as little information as possible, given the before- and after- dimensions.
- Also known as Karhuenen-Loeve (K-L) transformation (Watanabe, 1969).
- Could be used to preprocess input to, say, MLP.

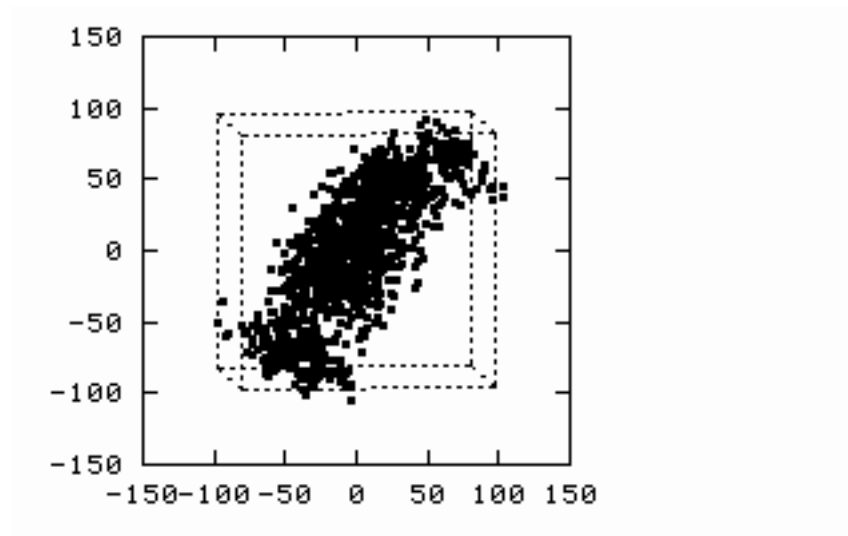
PCA



Reconstruction from Components

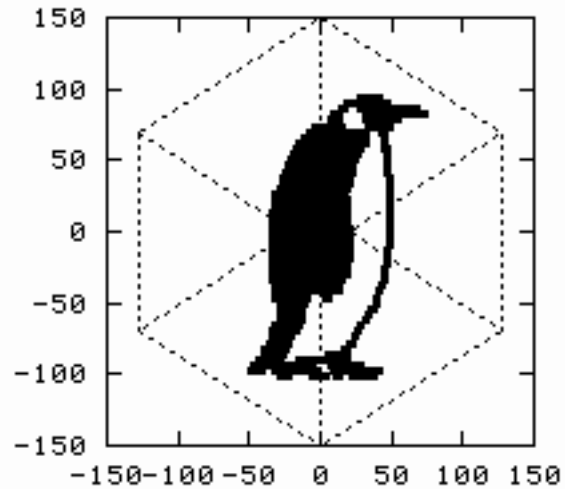


Scientific Uses



Transform coordinates to get a better understanding of the underlying phenomena.

Scientific Uses



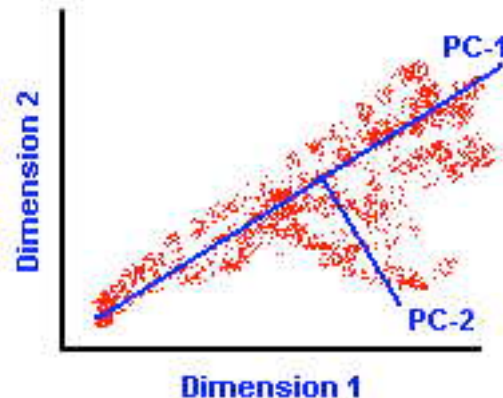
Transform coordinates to get a better understanding of the underlying phenomena.

Comparison with Linear Regression

- Linear regression requires one to ***pre-identify*** dependent vs. independent variables.
- PCA does not.

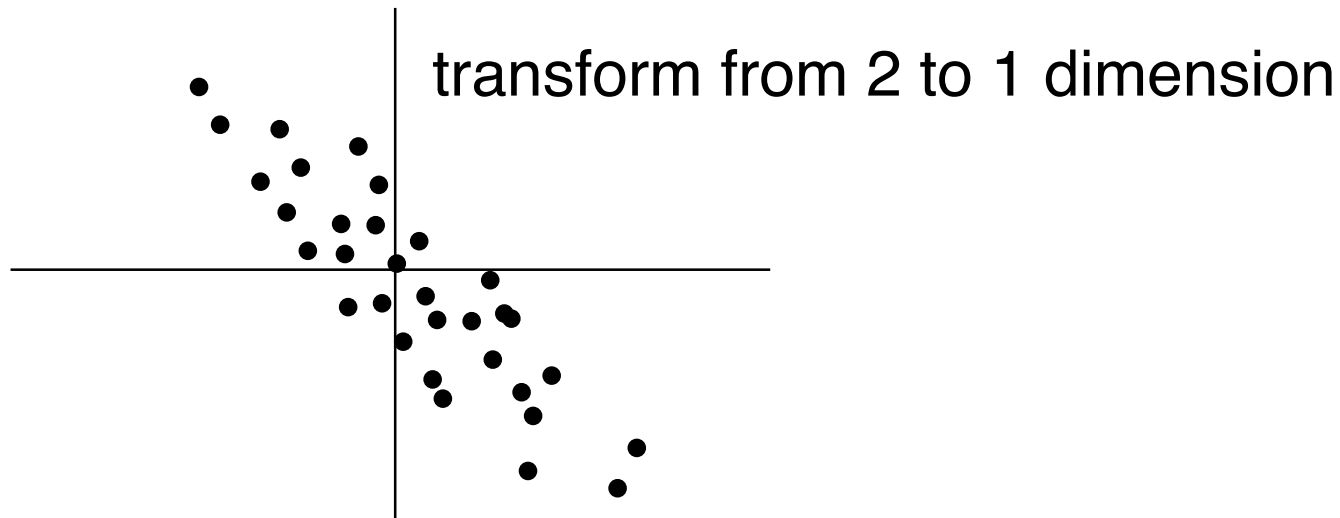
What is a “Principal Component”?

- If we were to plot the data, the **first principal component** would be the **predominant direction** in the data.
- The second principal component would be the second-most predominant direction, etc. up to the number of **dimensions** of the data points.



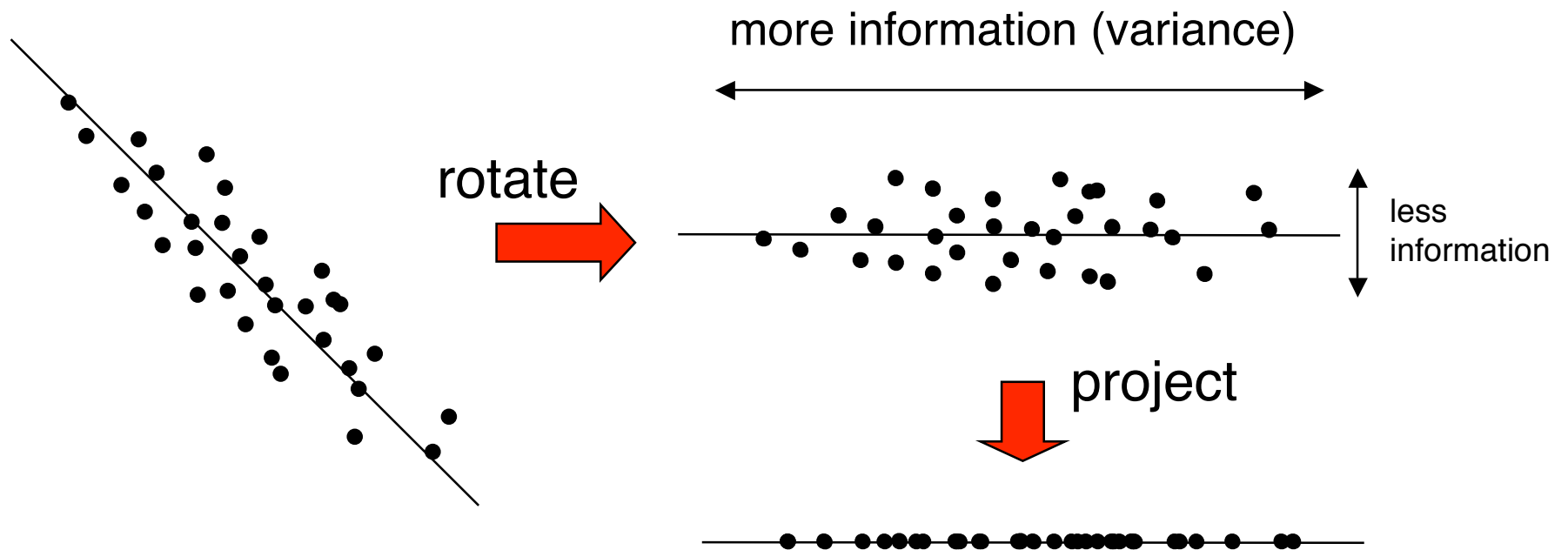
The main idea

- Transform the input data into fewer dimensions
- Preserve as much of the variance as possible



Transformation

- Preserve as much of the variance as possible



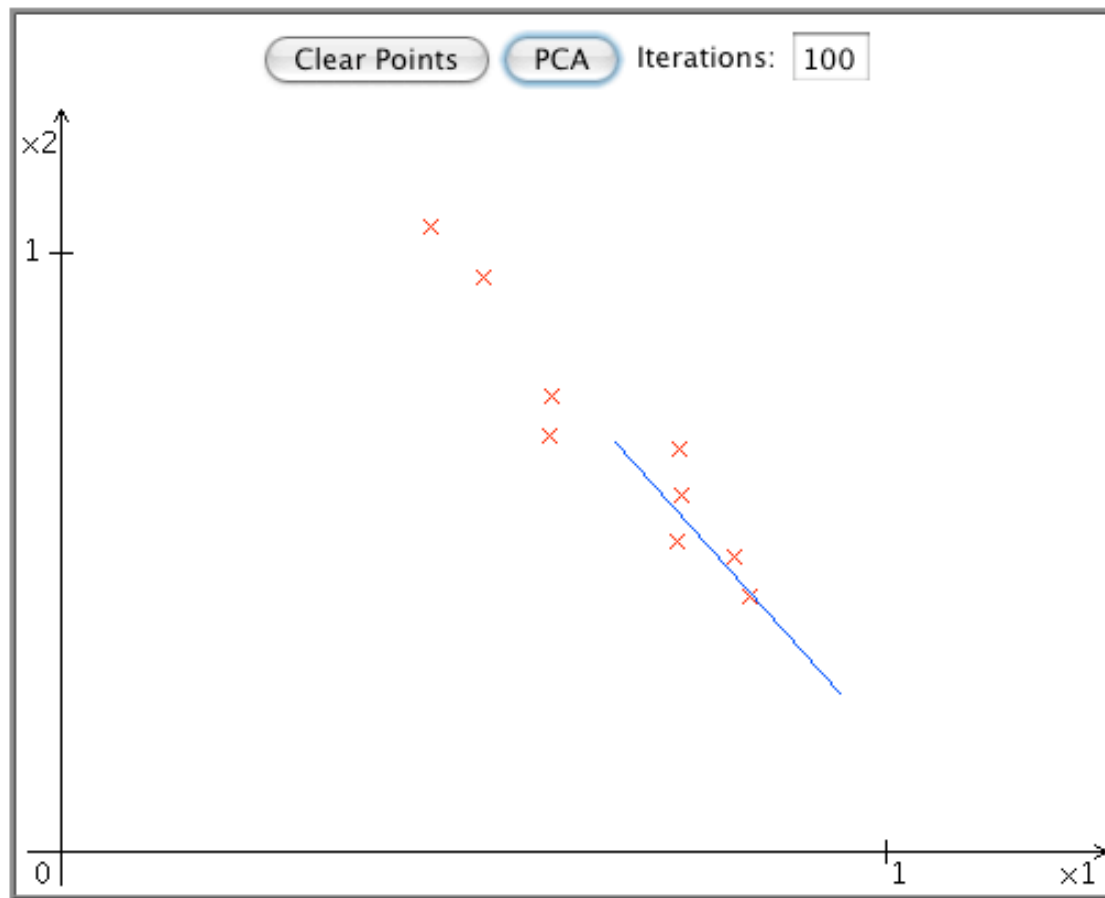
What is a Predominant Direction?

- This may seem subjective, but a mathematical definition can be provided.
- Let A be the matrix of data points:
 - Each point or observation is a column.
 - The rows correspond to the observed values of variables.

First P. C. Demo, you pick the points

http://www.igi.tugraz.at/lehre/CI/algorithms/pca_applet.html

This applet uses an iterative algorithm (APEX) to find the first PC.



Covariance vs. Correlation

- Most PCA presentations are based on covariance, which assumes input variables have commensurate units.
- Can also base PCA on correlation, which doesn't.
- To use covariance in general, first normalize by dividing by each mean-adjusted variable by its variance.

Linear Transformation

- The types of transformations shown are linear:
 - $B = WA$
where
 - A is the matrix of data (each point as a column vector), which is already mean-adjusted.
 - W is an $m \times n$ matrix, $m \leq n$
 - B is the transformed data matrix (called the “scores”)
- We want the columns of W to be **orthonormal** (i.e. $WW^T = I$).
- The reconstruction A' of A from B will be obtained by:
 - $A' = W^TB$
 $= W^T(WA) = (W^TW)A.$
(Note: W^TW won't generally be I , although WW^T is.)

Computing Principal Components

- PCA tries to minimize the *expectation* of reconstruction error:

$$E[\| A - A' \|^2] =$$

$$E[\text{tr}((A - A')(A-A')^T)] =$$

$$\text{tr}(E[AA']) - \text{tr}(W E[AA'] W^T) =$$

$$\text{tr}(R) - \text{tr}(WRW^T), \text{ where}$$

R is the *covariance matrix*, defined as $E[AA^T]$

- $E[]$ is the expectation, i.e. average over the data points
- tr is the trace (sum of diagonal elements).

What are the Principal Components, really?

- Want W such that **error** $\text{tr}(R) - \text{tr}(WRW^T)$ is **minimized**.
- Equivalently, WRW^T (i.e. the **variance** of the transformed data) is **maximized**.
- The **eigenvectors** of the covariance matrix $R = AA^T$, **ordered** corresponding to largest to smallest eigenvalue, are the **principal components**.
- Construct matrix W as using the top so-many correspondingly-ordered eigenvectors as rows.

Example (worked in Matlab)

Turtle Shell Classification (D. Morrison)

data = (columns are data points)

1.3000	1.4000	1.5000	1.2000	1.1000	(Height)
3.2000	2.8000	3.1000	2.9000	3.0000	(Width)
3.7000	4.1000	4.6000	4.8000	4.8000	(Length)

Can fewer than 3
variables explain all?

means =

1.3000
3.0000
4.4000

```
[M,N] = size(data);
```

```
means = mean(data, 2);
```

```
mean_adjusted = data - repmat(means,1,N);
```

mean_adjusted =

0	0.1000	0.2000	-0.1000	-0.2000
0.2000	-0.2000	0.1000	-0.1000	0
-0.7000	-0.3000	0.2000	0.4000	0.4000

Example

covariance =

```
0.0250  0.0025  -0.0275
0.0025  0.0250  -0.0250
-0.0275 -0.0250  0.2350
```

variances =

```
0.2415
0.0225
0.0210
```

explained_variation =

```
0.8472
0.0791
0.0737
```

```
covariance = (mean_adjusted * mean_adjusted') / (N-1);
[eigenvectors, eigenvalues] = eig(covariance);
[junk, rindices] = sort(-1*eigenvalues);
variances = eigenvalues(rindices);
explained_variation = variances/sum(variances)
```

Example: Principal Components

PC = (column vectors)

```
-0.1265  0.5534 -0.8232
-0.1153 -0.8325 -0.5419
 0.9852 -0.0263 -0.1691
```

```
PC = eigenvectors(:, rindices);

% transform the original data set by column
scores = PC * mean_adjusted;
```

scores = (transformed data, columns)

```
-0.4656 -0.3703  0.1947  0.2866  0.3546
-0.5459 -0.0076  0.0021  0.3116  0.2398
-0.1236  0.0531  0.2282 -0.0283 -0.1294
```

Example: Orthonormality Check

orthogonality_check =

1.0000	0.0000	0
0.0000	1.0000	0.0000
0	0.0000	1.0000

$$\text{orthogonality_check} = \text{PC} * \text{PC}'$$

Example: Reconstruction

subtracted_means =

1.3000	1.3000	1.3000	1.3000	1.3000
3.0000	3.0000	3.0000	3.0000	3.0000
4.4000	4.4000	4.4000	4.4000	4.4000

reconstruction_using_all_components =

1.3000	1.4000	1.5000	1.2000	1.1000
3.2000	2.8000	3.1000	2.9000	3.0000
3.7000	4.1000	4.6000	4.8000	4.8000

reconstruction_error =

0	0	0	0	0
0	0	0	0	0
0	0	0	0	0

reconstruction_using_all_components = ...
(PC' * scores) + subtracted_means

reconstruction_error = reconstruction_using_all_components - data

Example: Reconstruction

reconstruction_using_first_two_components =

1.4218	1.3477	1.2751	1.2278	1.2275
3.1967	2.8014	3.1060	2.8993	2.9966
3.7209	4.0910	4.5614	4.8048	4.8219

reconstruction_error =

0.1218	-0.0523	-0.2249	0.0278	0.1275
-0.0033	0.0014	0.0060	-0.0007	-0.0034
0.0209	-0.0090	-0.0386	0.0048	0.0219

mse_reconstruction =

0.0058

```
reconstruction_using_first_two_components = ...  
    (components(1:2, :) * scores(1:2, :)) + subtracted_means  
  
reconstruction_error = reconstruction_using_first_two_components - data  
  
mse_reconstruction = mse(reconstruction_error)
```

Example: Reconstruction

reconstruction_using_first_component_only =

1.3589	1.3468	1.2754	1.2637	1.2551
2.7423	2.7951	3.1077	3.1586	3.1962
4.0167	4.0952	4.5603	4.6359	4.6919

reconstruction_error =

0.0589	-0.0532	-0.2246	0.0637	0.1551
-0.4577	-0.0049	0.0077	0.2586	0.1962
0.3167	-0.0048	-0.0397	-0.1641	-0.1081

mse_reconstruction =

0.0360

```
reconstruction_using_first_component_only = ...  
    (components(1:1, :) * scores(1:1, :)) + subtracted_means  
  
reconstruction_error = reconstruction_using_first_component_only - data  
  
mse_reconstruction = mse(reconstruction_error)
```

Matlab cov Function

A = (mean adjusted, by row)

```
      0      0.2000     -0.7000
    0.1000     -0.2000     -0.3000
    0.2000      0.1000      0.2000
   -0.1000     -0.1000      0.4000
   -0.2000          0      0.4000
```

```
>> R = cov(A)
```

R =

```
    0.0250    0.0025   -0.0275
    0.0025    0.0250   -0.0250
   -0.0275   -0.0250    0.2350
```

Matlab pcacov function

```
>> [pc, variances, explained] = pcacov(R)
```

```
pc =
```

```
-0.1265    0.5534    0.8232  
-0.1153   -0.8325    0.5419  
 0.9852   -0.0263    0.1691
```

```
variances =
```

```
 0.2415 }  
 0.0225 }  
 0.0210 }
```

Eigenvalues

```
explained =
```

```
84.7212  
 7.9114  
 7.3674
```

% Variance explained

Singular Value Decomposition: Another approach to computing PCs

- The singular-value decomposition (SVD) of A is:
$$A = USV^T$$
- where:
 - The columns of U are the eigenvectors of AA^T .
 - The columns of V are the eigenvectors of $A^T A$, i.e. the PC's.
 - S is pseudo-diagonal (diagonal insofar as this is possible with a rectangular matrix, the rest of the entries being 0).

- PC from SVD in Matlab:

$$Y = \text{mean_adjusted} / \text{sqrt}(N-1)$$

$$[U,S,PC] = \text{svd}(Y')$$

$$[\text{Note: } Y' = U S PC']$$

Planets Example

(with the help of

<http://www.cs.mcgill.ca/~sqrt/dimr/dimreduction.html>)

- Consider a 3-dimensional data set where the variables are the logarithms of
 - distance to the sun
 - equatorial diameter
 - density

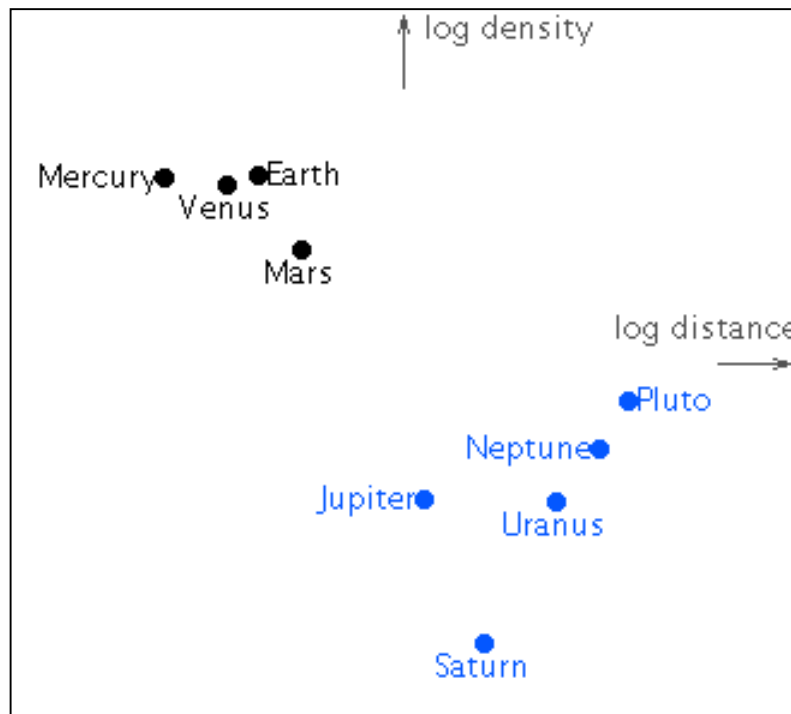
Data set (prior to taking logs)

planet	distance	diameter	density
Mercury	0.387	4878	5.42
Venus	0.723	12104	5.25
Earth	1.000	12756	5.52
Mars	1.524	6787	3.94
Jupiter	5.203	142800	1.314
Saturn	9.539	120660	0.69
Uranus	19.18	51118	1.29
Neptune	30.06	49528	1.64
Pluto	39.53	2300	2.03

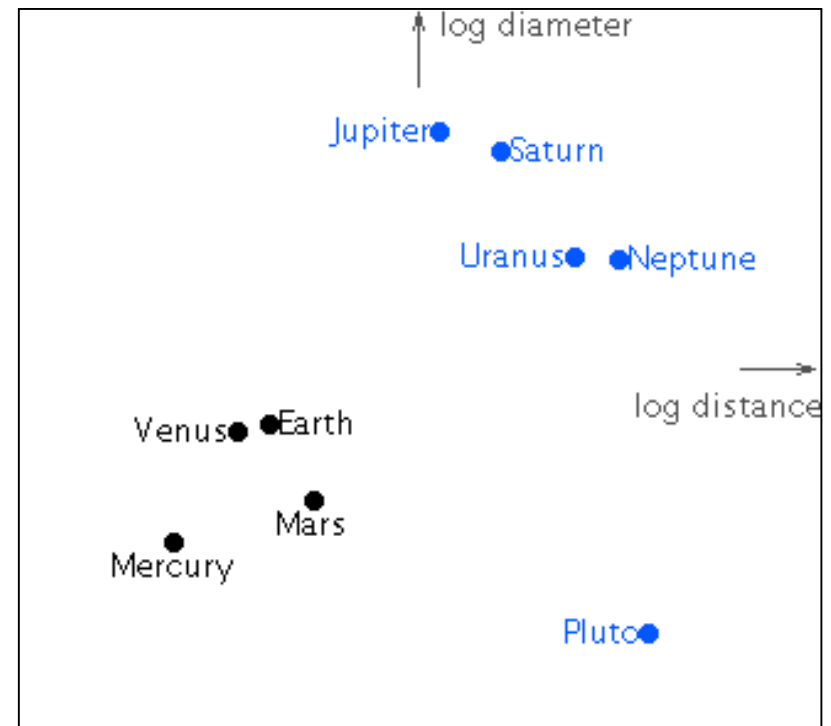
Log Data Projections

(projections being a special case of linear transformation)

In two dimensions, we can plot any one variable against another, e.g.



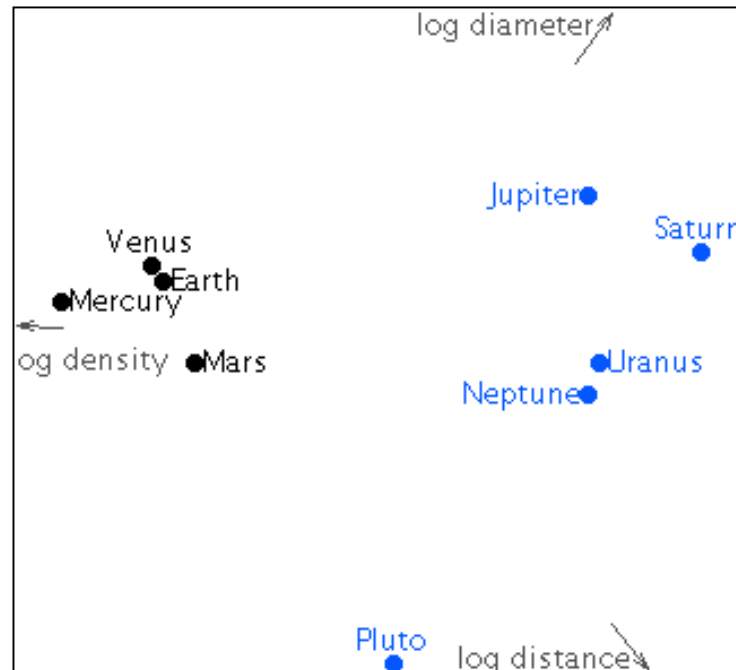
or



Of various possible plots, which show the widest variation among planets?

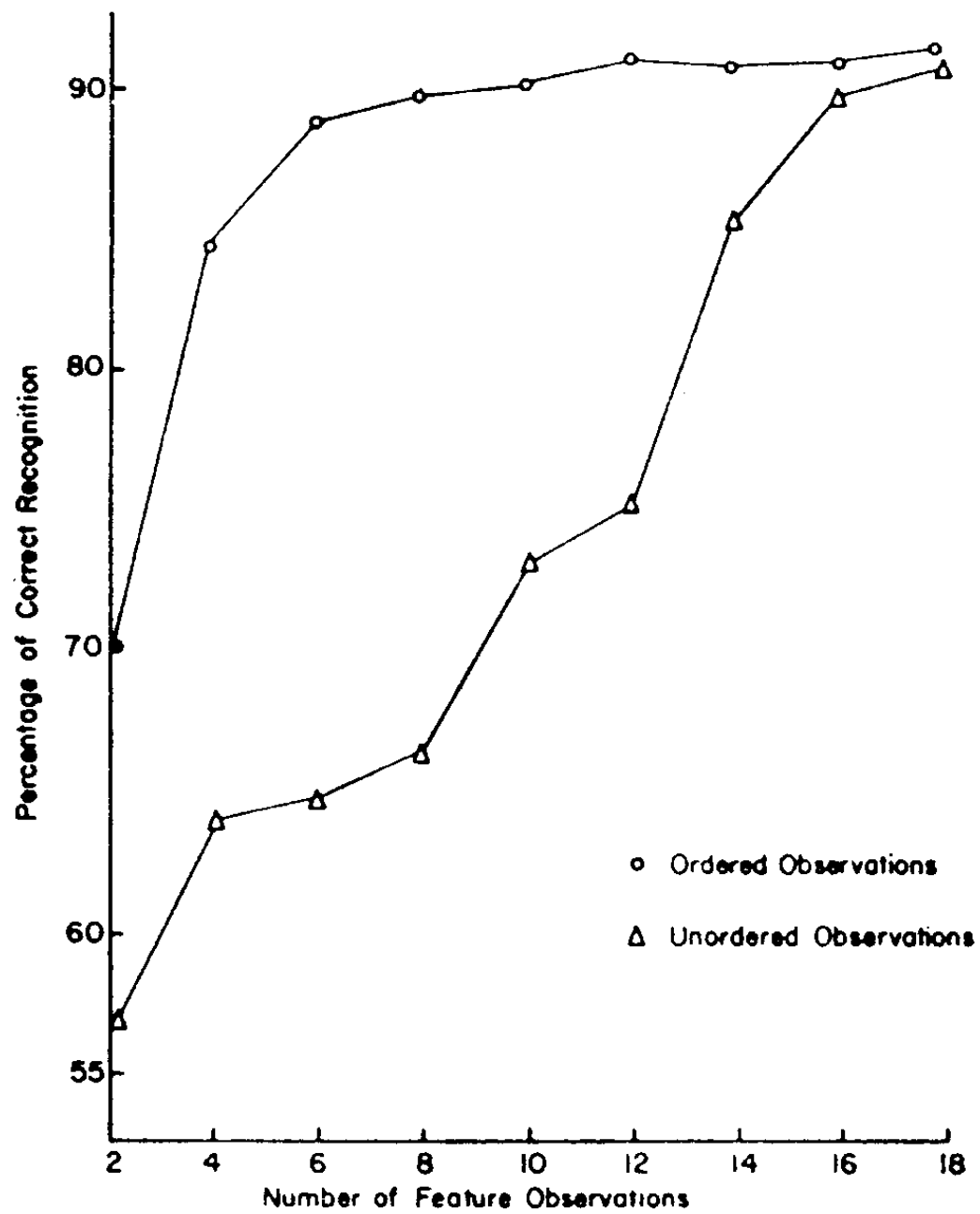
Maximizing Variance

- Transforming using first two principal components preserves more of the variance (summing variances in each dimension) in the projection than does projecting on any 2 variables:



Application: Handwritten Character recognition (K.S. Fu, 1968)

- Attempt recognition based upon **18 radial measurements**, spaced at 20 degree increments, of characters (240 samples).
- Recognition rate was computed vs. the number of measurements used (2, 4, 6, ... out of 18) in no particular order.
- The same computation was done with the measurements **ordered by principal components**.
- The next slide compares the two approaches.

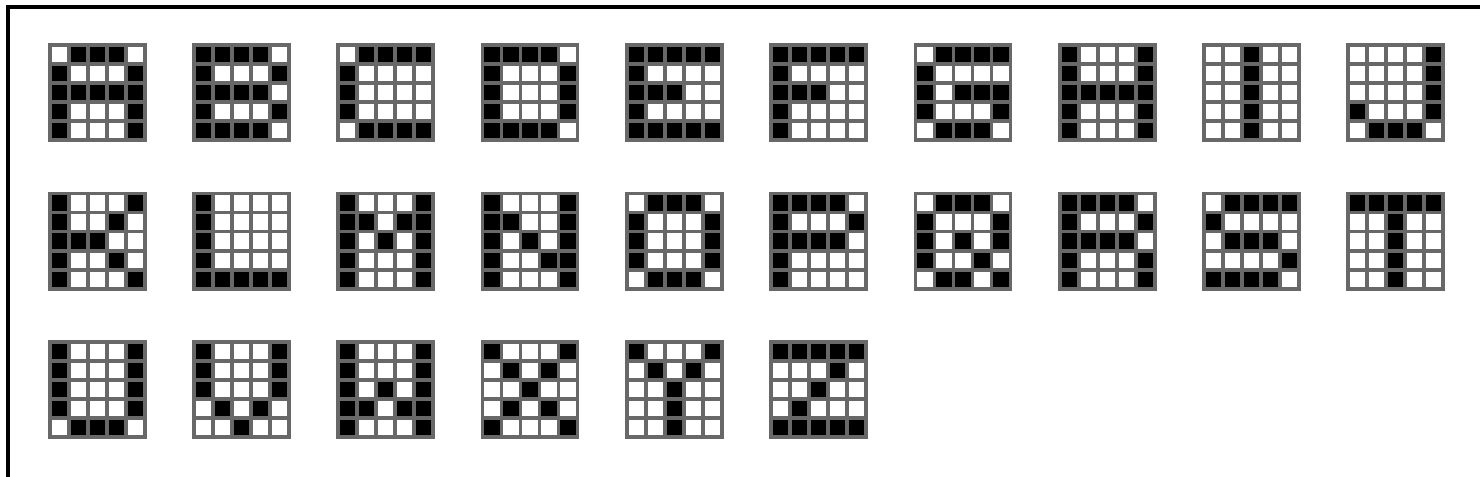


A Related Example

(also with the help of

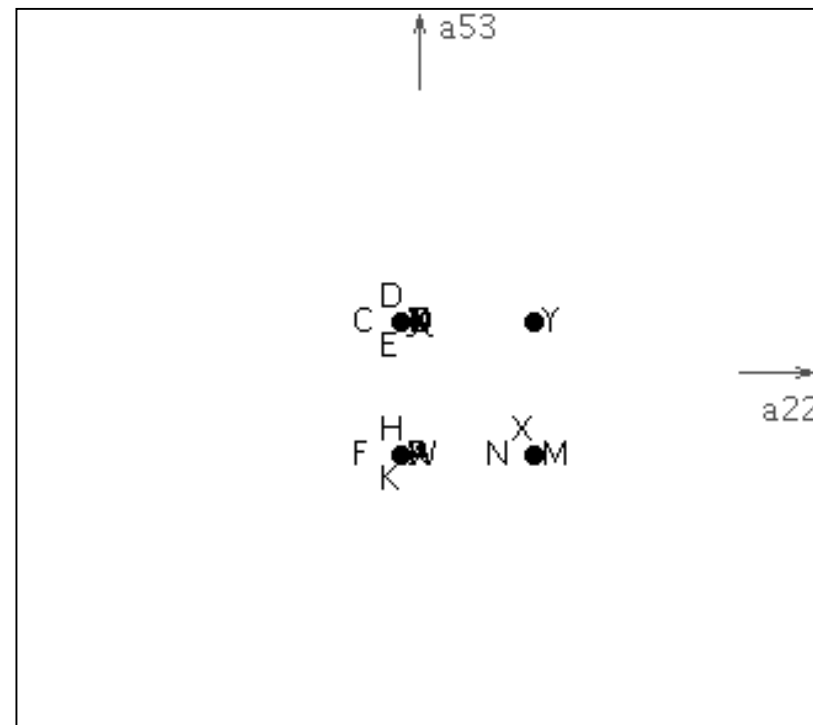
<http://www.cs.mcgill.ca/~sqrt/dimr/dimreduction.html>)

- A 25-dimensional data set:



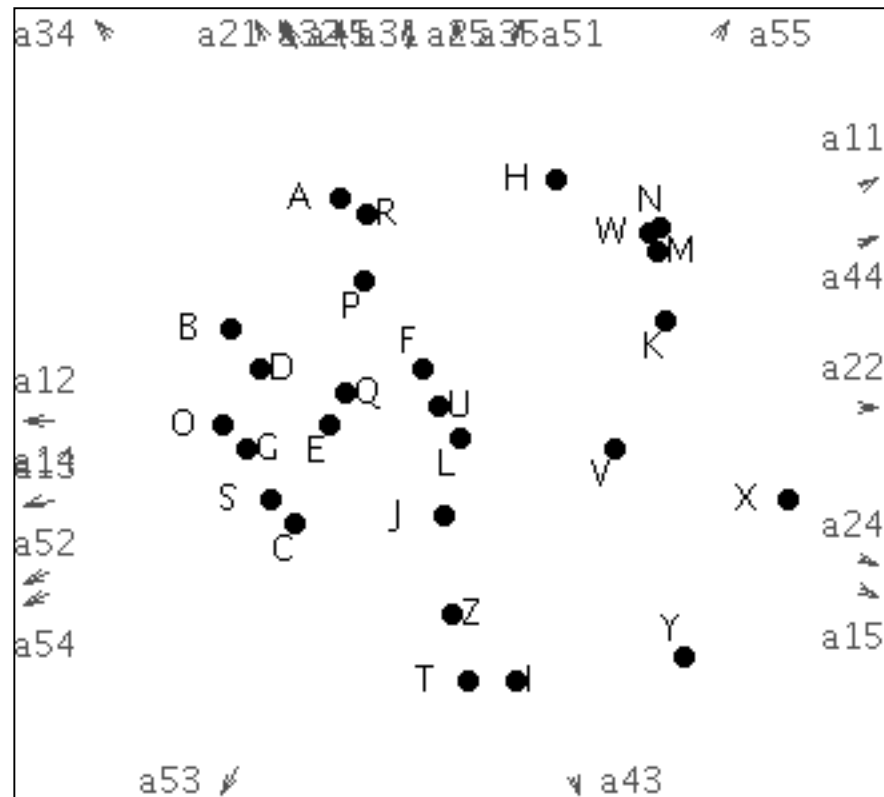
Projected Data

- Projecting on 2 variables does not help much in discriminating the points, e.g.



Projected Data

- Projecting on the first two principal components achieves the maximum variation in two dimensions:

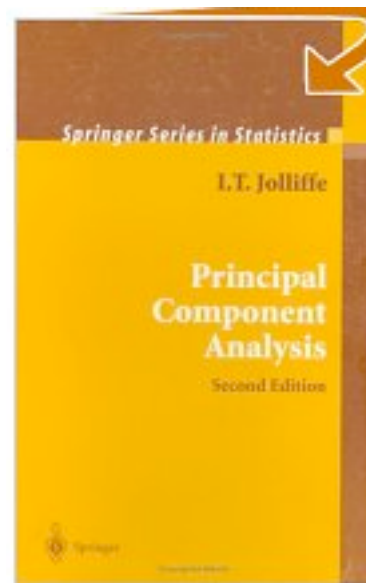


Eigenfaces Example

- Principal components of a raster image
- cf. <http://vismod.media.mit.edu/vismod/demos/facerec/basic.html>



PCA Source

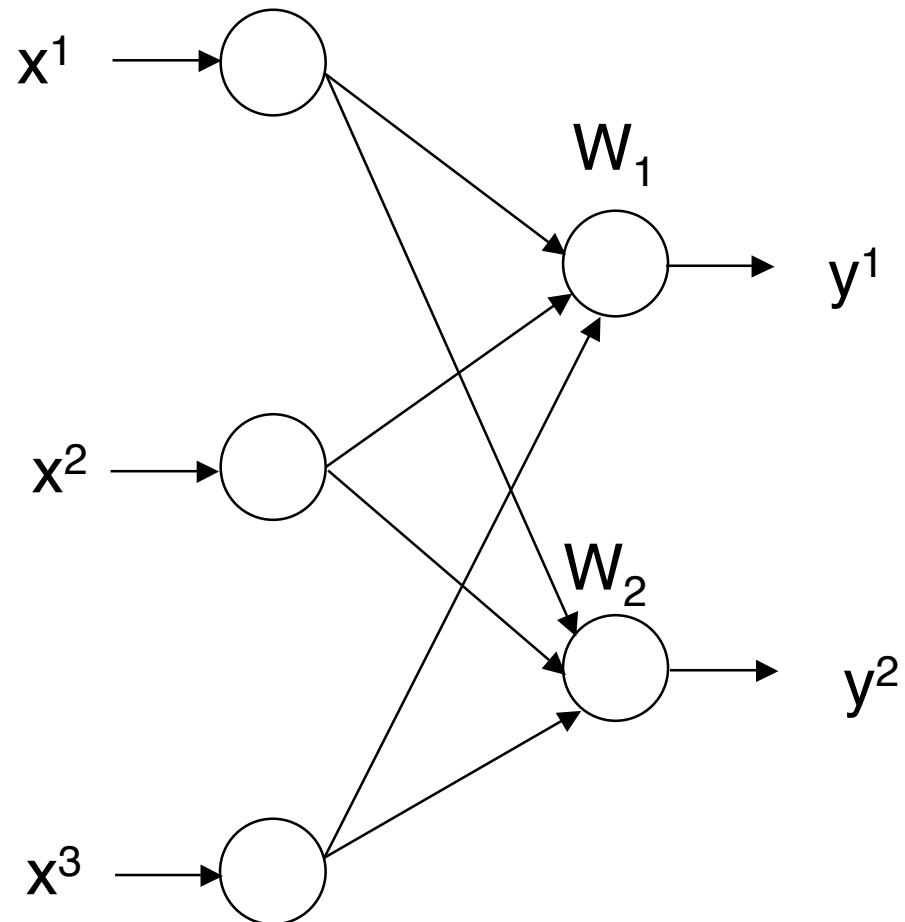


L.T. Jolliffe

What is a PCA Neural Network?

- The matrix W of PCA can be interpreted as **weights** of a 1-layer neural network.
- Training is by unsupervised Hebbian learning.
- The user pre-selects the output variables.
- The network effectively learns the first PC by outputting the corresponding scores.

3→2 PCA Network



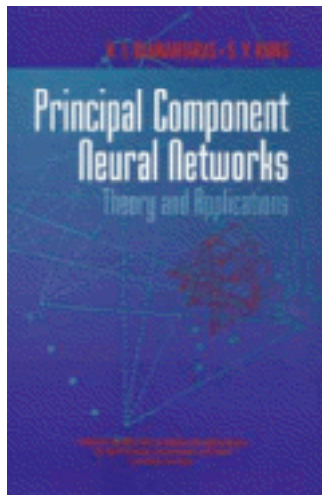
PCA Hebbian-Learning Rules

- $\Delta W = \eta y_i x_i^T - K_i W$, where
 - x_i is an input vector
 - y_i is the corresponding output vector ($= W x_i$)
 - $y_i x_i^T$ is the **Hebbian** component
 - K_i is **one of the following** (purpose: **weight decay**):
 - 0 pure Hebbian (unbounded)
 - $y_i y_i^T$ (Williams' rule, 1985)
 - $3D(y_i y_i^T) + 2L(y_i y_i^T)$ (Oja and Karhunen rule, 1985)
 - $L(y_i y_i^T)$ (Sanger's rule, 1989)

where $D(M)$ maps the **diagonal** entries of M to themselves, and other entries to 0, and

- $L(M)$ maps entries **below** the main diagonal to themselves, and other entries to 0.

Other Variations Exist: Source for PCA NN



Diamantaras, K.I. / Kung, S.Y.

Principal Component Neural
Networks
Theory and Applications

1996. 256 pages.

ISBN 0-471-05436-4 John Wiley & Sons

The coverage in this book is extensive. It gives several additional learning algorithms, including ones with **lateral** connections as well as feed-forward ones.

Other means of learning for dimensionality reduction

- Self-Organizing Map
(# of dimensions in superstructure = reduced dimension)
- LVQ
(# of neurons = reduced dimension)
- Projection pursuit, another statistical technique, effectively learns PC's sequentially

PCA Shortcomings

- The PCA user must select **how many** target dimensions to use.
- PCA only finds **linear** sub-spaces.
- PCA works best if the individual components are Gaussian-distributed.
- There are many variations, some of which overcome some of these issues.
- Related area: “Factor Analysis”

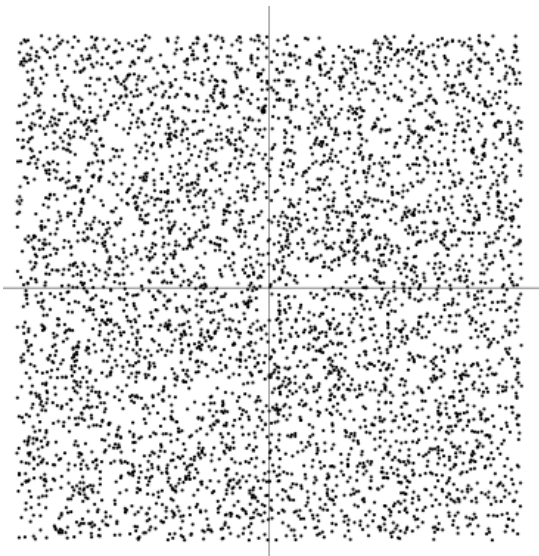
Independent Component Analysis (ICA)

- A technique for BSS (Blind Source Separation).
- Proposed for neuro-mimetic hardware in 1983 by Herault and Jutten.
- ICA seeks to transform to components that are **statistically independent**.
- Two variables x, y are *statistically independent* iff
$$P(x, y) = P(x)P(y).$$
Equivalently,
$$E\{g(x)h(y)\} - E\{g(x)\} E\{h(y)\} = 0$$
where g and h are any functions.

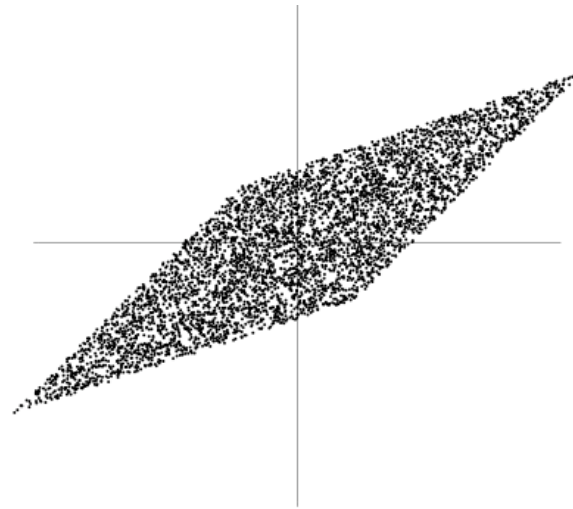
PCA vs. ICA

- PCA is finding uncorrelated components.
- ICA finds independent components.
- Independent implies uncorrelated, but not necessarily conversely.
- The input signals to ICA, being mixtures, will usually be correlated. The outputs, being independent, will not be.

Signals vs. Mixtures



Joint distribution of two independent signals



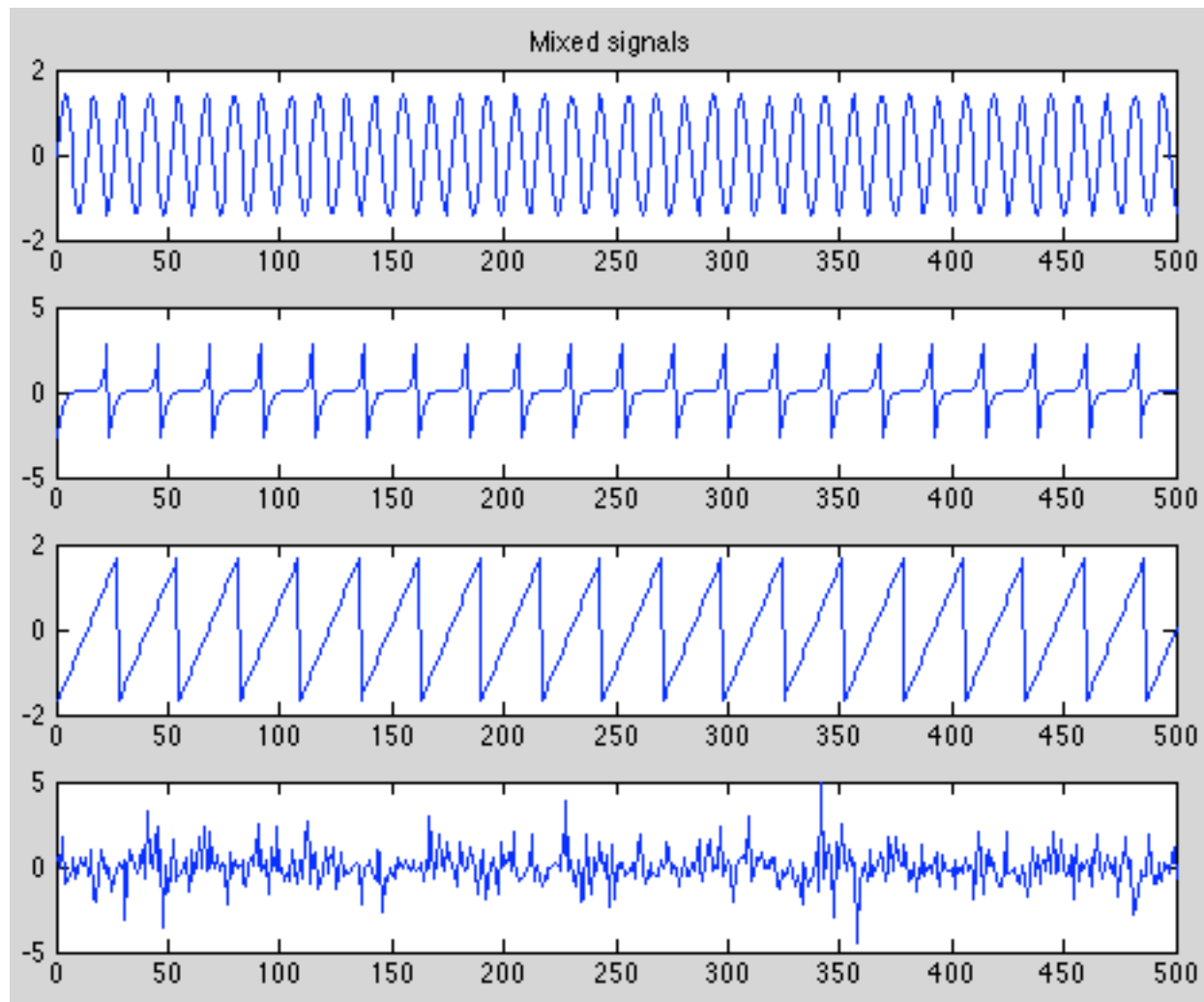
Joint distribution of a mixture of the same signals

ICA

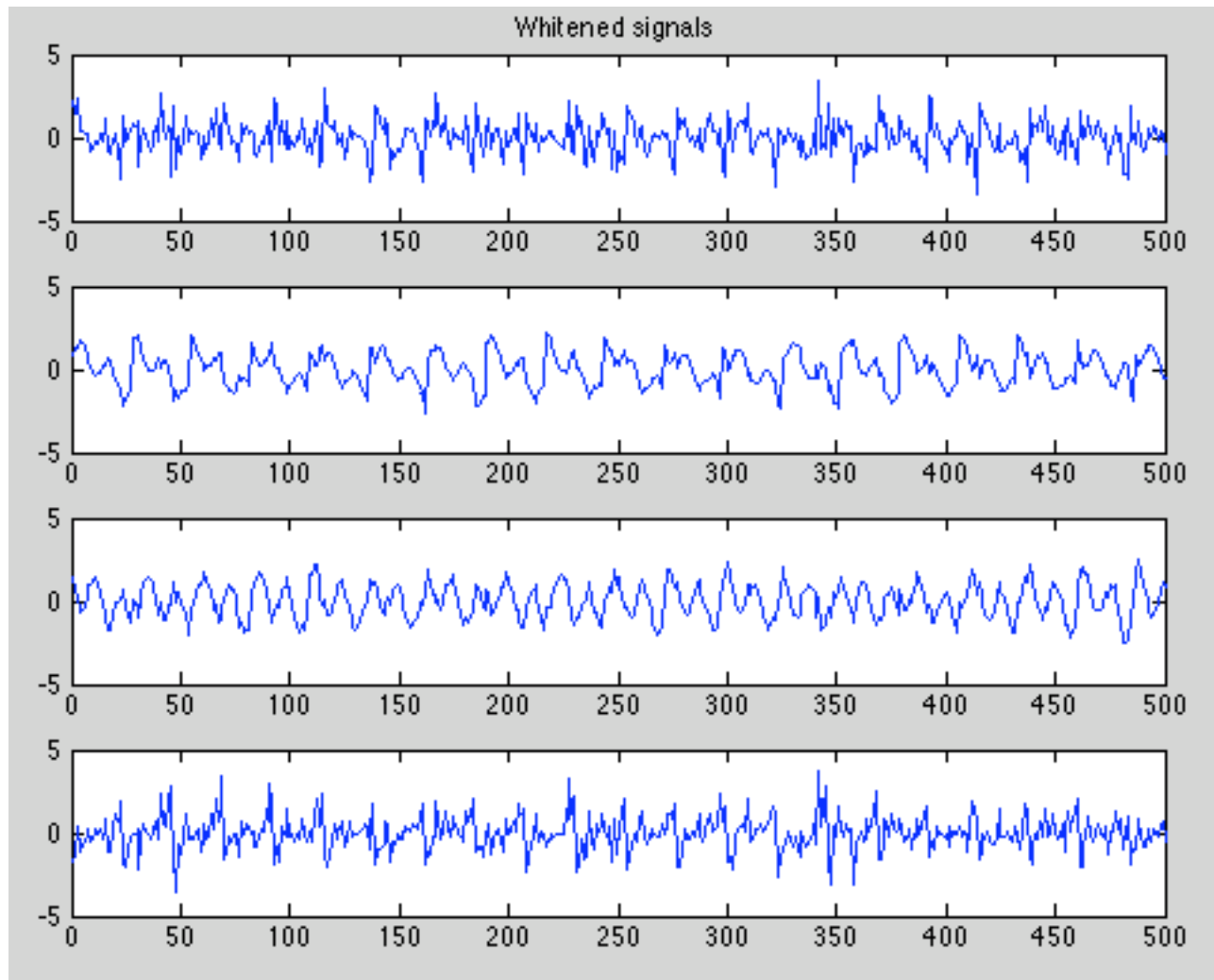
- Suppose we have n distinct mixtures of n statistically-independent signals.
- We wish to transform the mixtures to (signals as close as possible to) the original signals.

Fast-ICA Demo

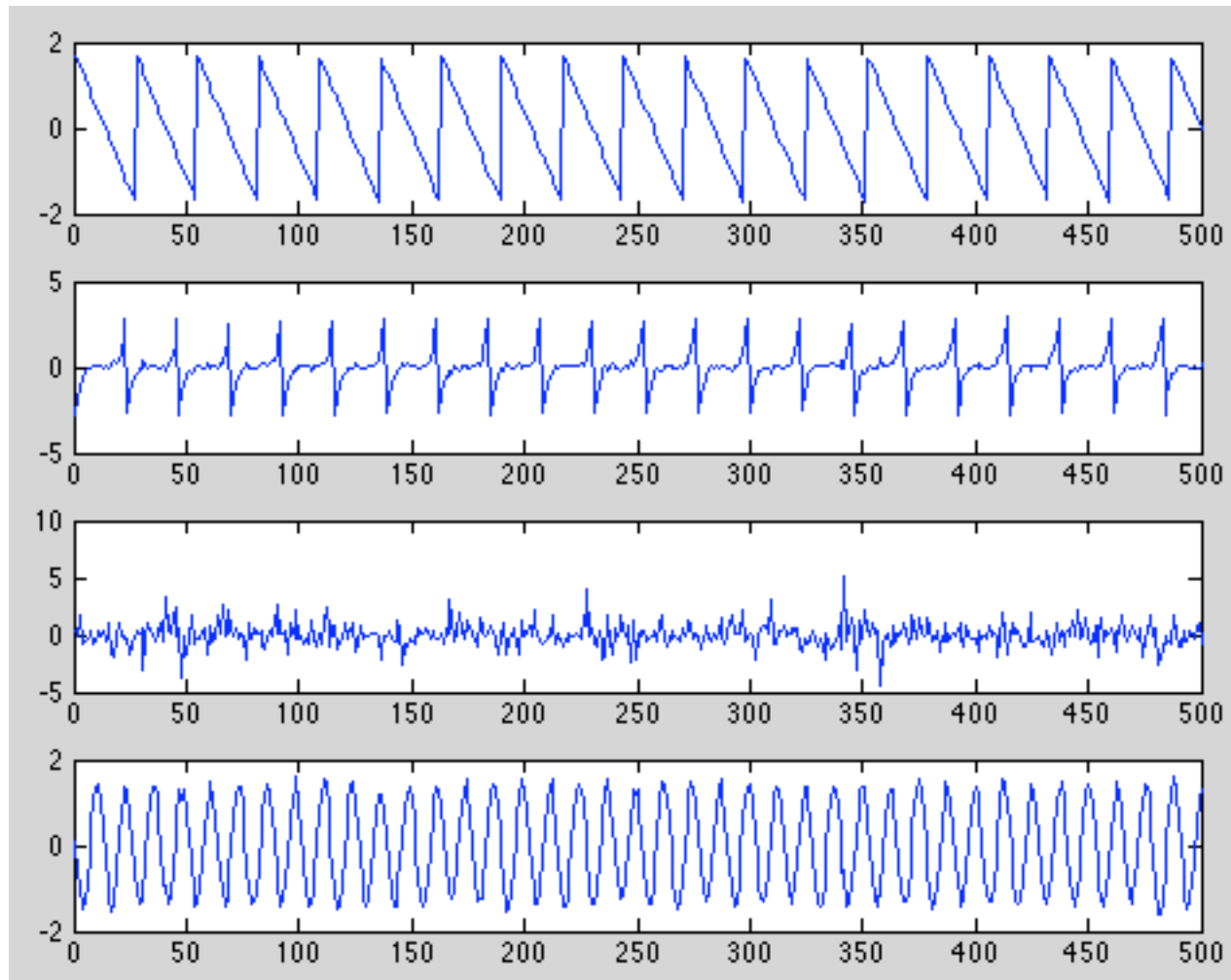
Signals to be Mixed



Whitened Signals after Mixing



Extracted Independent Components



ICA Demos

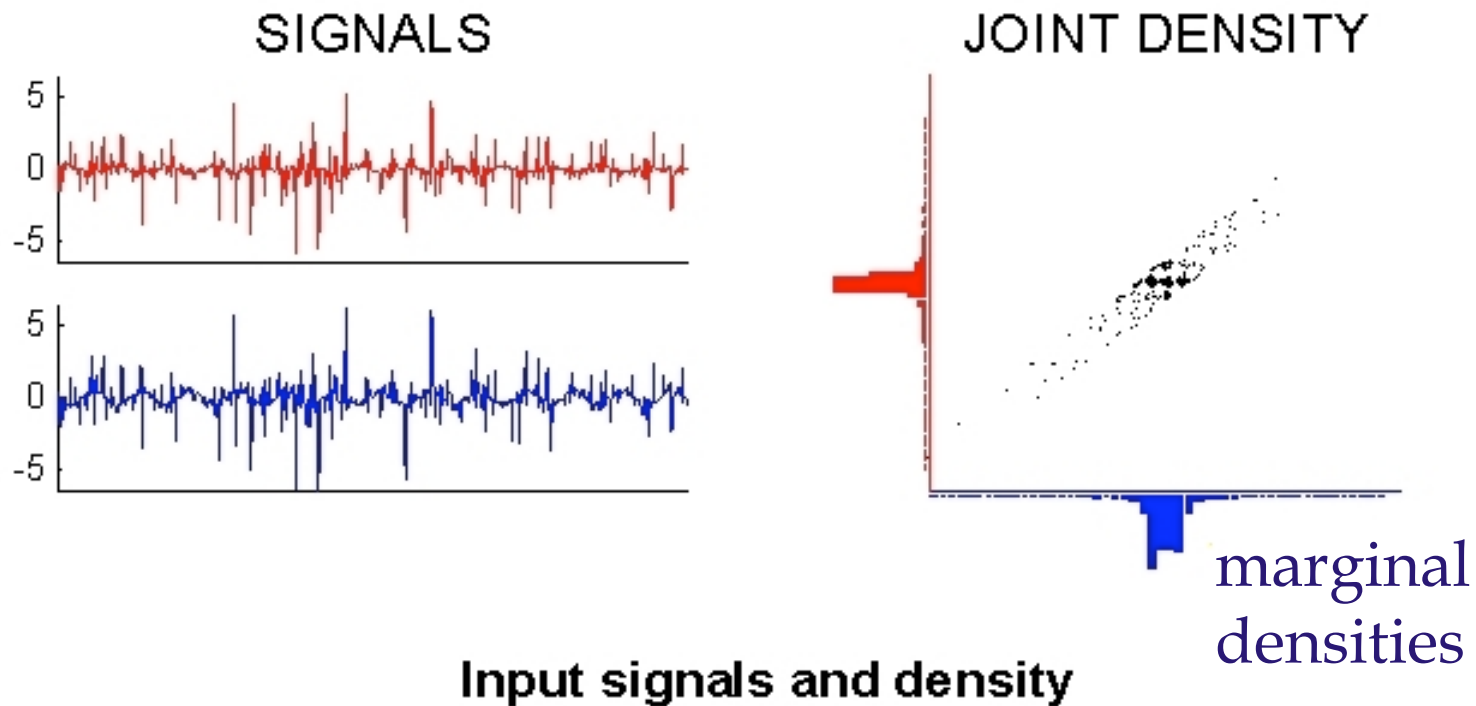
- http://www.cnl.salk.edu/~tewon/Blind/blind_audio.html
- http://www.cis.hut.fi/projects/ica/cocktail/cocktail_en.cgi

How can/does ICA work?

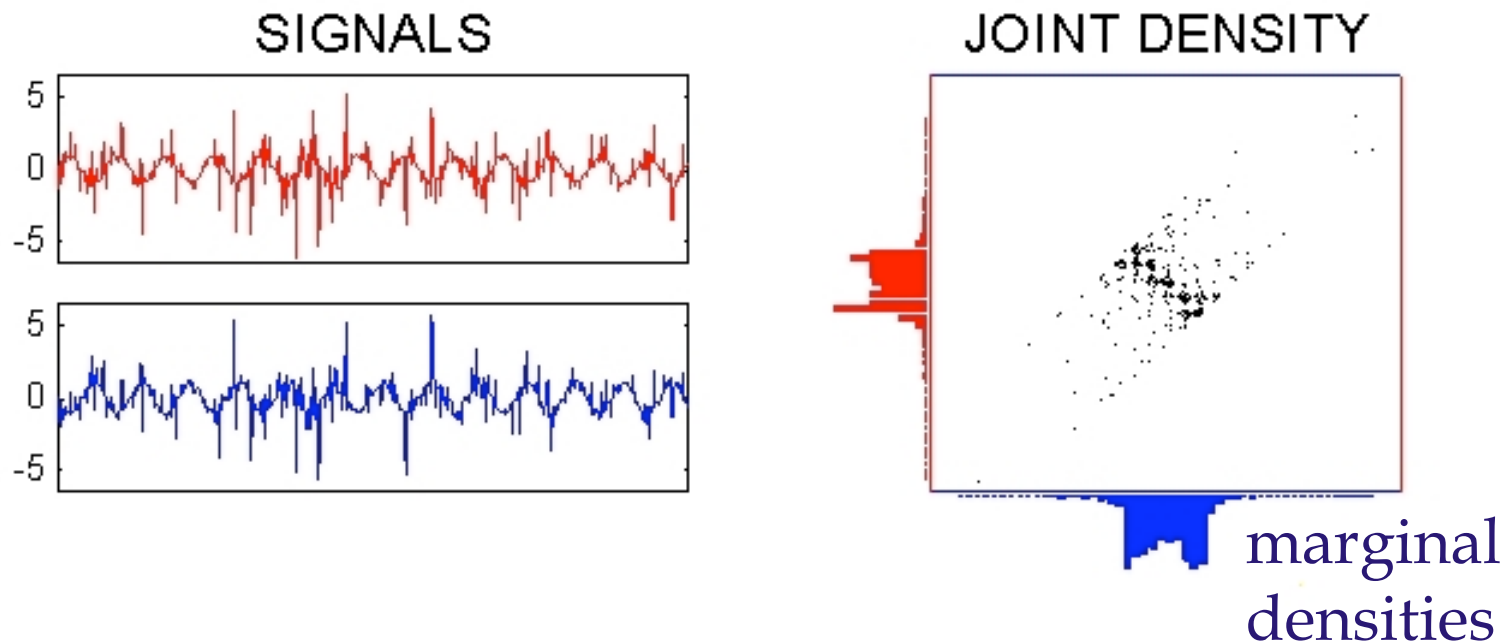
- One version: Fast ICA
- Throughout signals are assumed to be mean-adjusted.

Correlation within Signal Mixture

(from <http://www.cis.hut.fi/projects/ica/icademo/>)



Effect of whitening (aka “sphering”): removing correlations in mixed signals



Whitened signals and density

Whitening Transformation

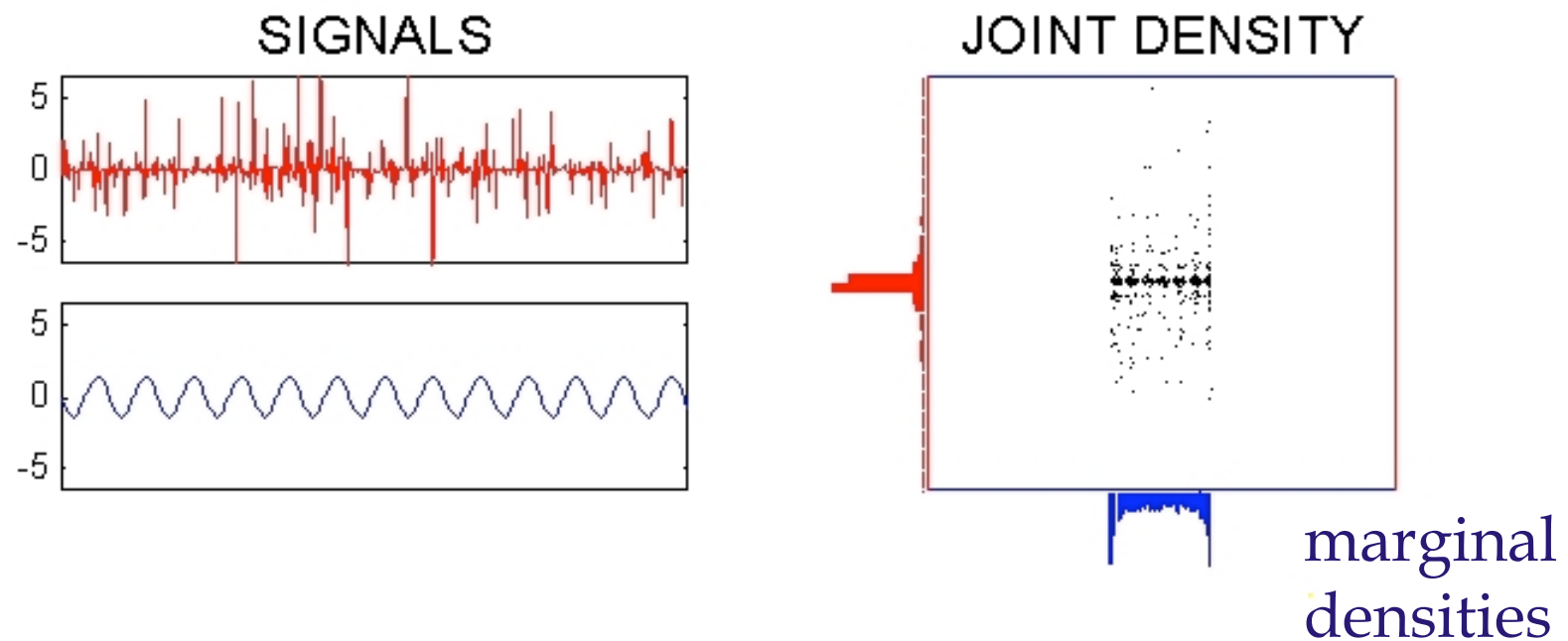
(essentially a form of PCA)

- Seek a linear transformation V such that when $y = Vx$ we get $E\{yy'\} = I$.
- Set $V = C^{-1/2}$, where $C = E\{xx'\}$ is the correlation matrix of the data.
- Then $E\{yy'\} = E\{Vxx'V'\} = C^{-1/2}CC^{-1/2} = I$.
- For computational aspects, see:
http://en.wikipedia.org/wiki/Square_root_of_a_matrix

Rotating the Whitened Signals

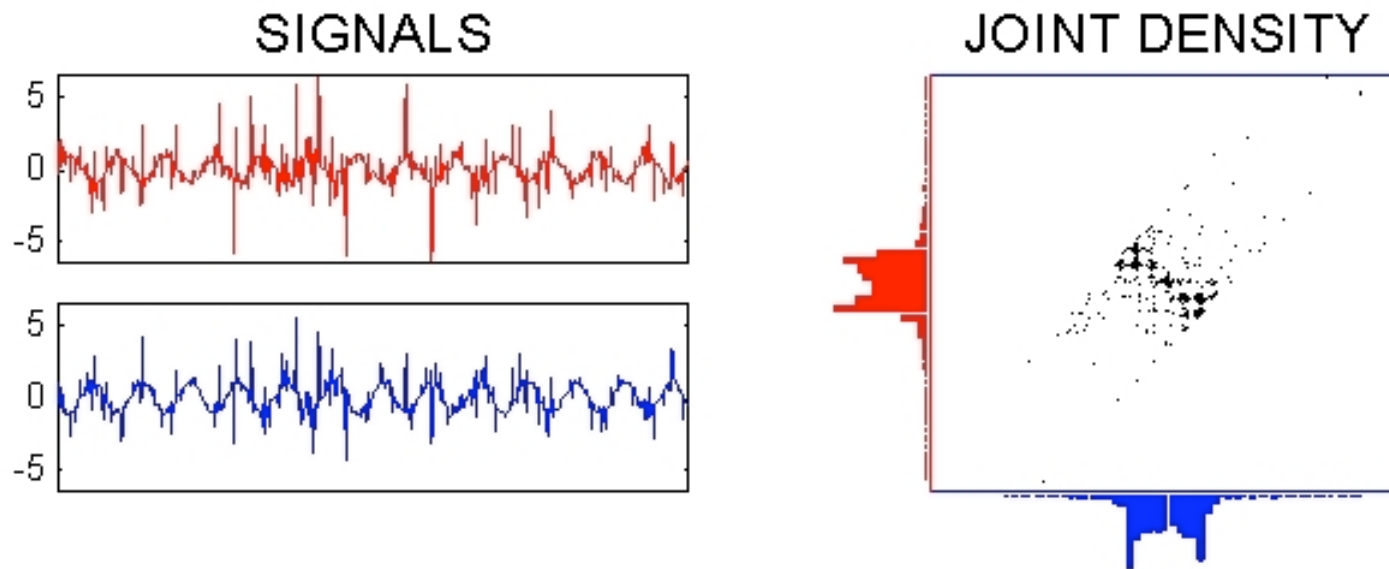
- The independent components are achieved by an appropriate rotation of the whitened signals.
- This involves **maximizing** the non-normal aspects of the marginal densities, since
- A linear mixture of independent random variables is necessarily more Gaussian than the original variables
- Maximization can be done incrementally, using an approach such as Fast-ICA.

Separated Signals



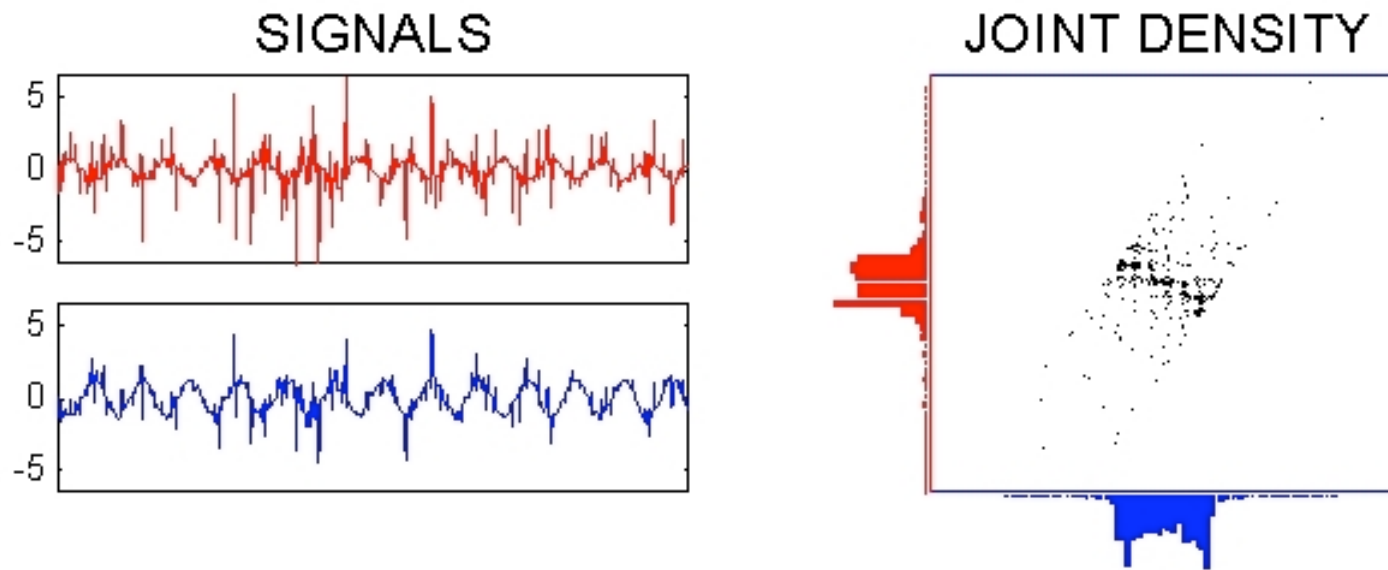
Separated signals after 5 steps of FastICA

Fast ICA Intermediate Steps



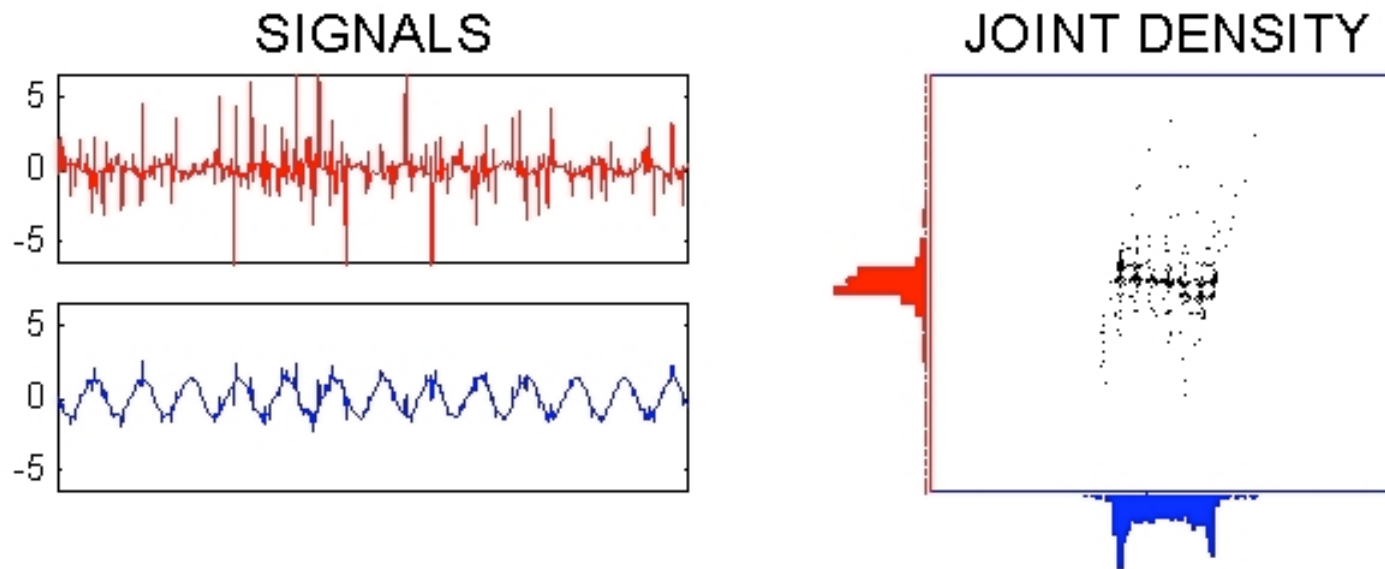
Separated signals after 1 step of FastICA

Fast ICA Intermediate Steps



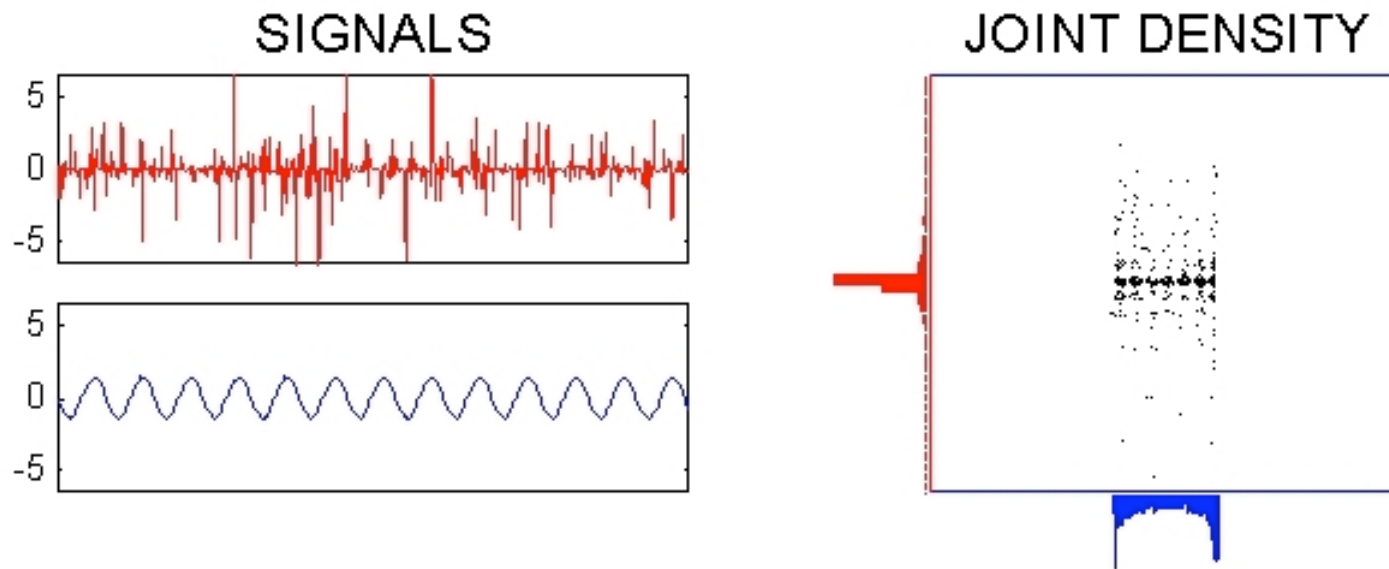
Separated signals after 2 steps of FastICA

Fast ICA Intermediate Steps



Separated signals after 3 steps of FastICA

Fast ICA Intermediate Steps



Separated signals after 4 steps of FastICA

Neural ICA Algorithms

- Bell and Sejnowski: “infomax” method, minimizing mutual information, using an iterative formulation similar to PCA NN:

$$\mathbf{B}_{k+1} = \mathbf{B}_k + \beta_k [\mathbf{B}_k^{-T} + \mathbf{z}_k \mathbf{x}_k^T]$$

$$\mathbf{z}(i) = \partial/\partial u(i) \partial u(i)/\partial y(i)$$

$$u = f(\mathbf{B}\mathbf{x}); f = \text{tansig, etc.}$$

- Also related to “maximum likelihood estimation”. Subsequent Bayesian derivation by Kevin Knuth

Matlab Core Code based on Bell-Sejnowski

```
% Begin gradient ascent on h ...
for iter=1:maxiter
    % Get estimated source signals, y.

    y = x*W; % wt vec in col of W.

    % Get estimated maximum entropy signals Y=cdf(y).

    Y = tanh(y);

    % Find value of function h.
    % h = log(abs(det(W))) + sum( log(eps+1-Y(:).^2) )/N;

    detW = abs(det(W));
    h = ( (1/N)*sum(sum(Y)) + 0.5*log(detW) );

    % Find matrix of gradients @h/@W_ji ...
    g = inv(W') - (2/N)*x'*Y;

    % Update W to increase h ...

    W = W + eta*g;

    % Record h and magnitude of gradient ...

    hs(iter)=h; gs(iter)=norm(g(:));
end;
```

Neural ICA Algorithms

- Bell-Sejnowski was similar to the earlier Herault-Jutten formulation:

$$B = (I+S)^{-1}$$

$$S_{k+1} = S_k + \beta_k g(y_k) h(y_k^T)$$

$g = t, h = t^3; g = \text{hardlim}, h = \text{tansig}$

Neural ICA Algorithms

- EASI (Equivariant Adaptive Separation via Independence): Cardoso et al

$$\mathbf{B}_{k+1} = \mathbf{B}_k - \beta_k [\mathbf{y}_k \mathbf{y}_k^T - \mathbf{I} + \mathbf{g}(\mathbf{y}_k) \mathbf{h}(\mathbf{y}_k)^T - (\mathbf{y}_k) \mathbf{g}(\mathbf{y}_k)^T] \mathbf{B}_k$$

$\mathbf{g}=\mathbf{t}$, $\mathbf{h}=\mathbf{tansig}$

ICA Image Separation Demo (David Gleich '03)



Original 3
images

Mixture

ICA Image Separation Example (David Gleich Gleich '03)

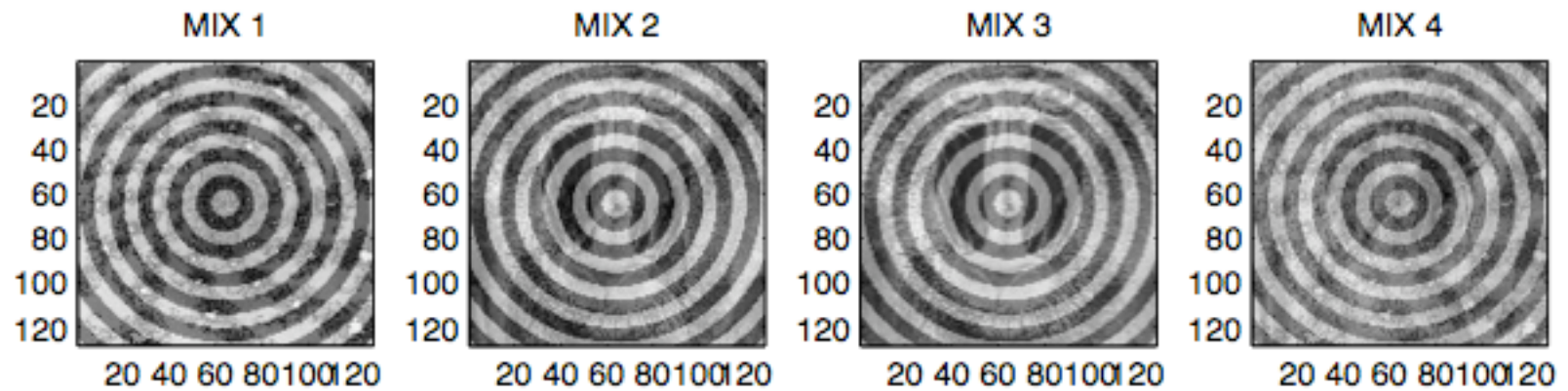


Bell and Sejnowski

EASI Algorithm

Original images

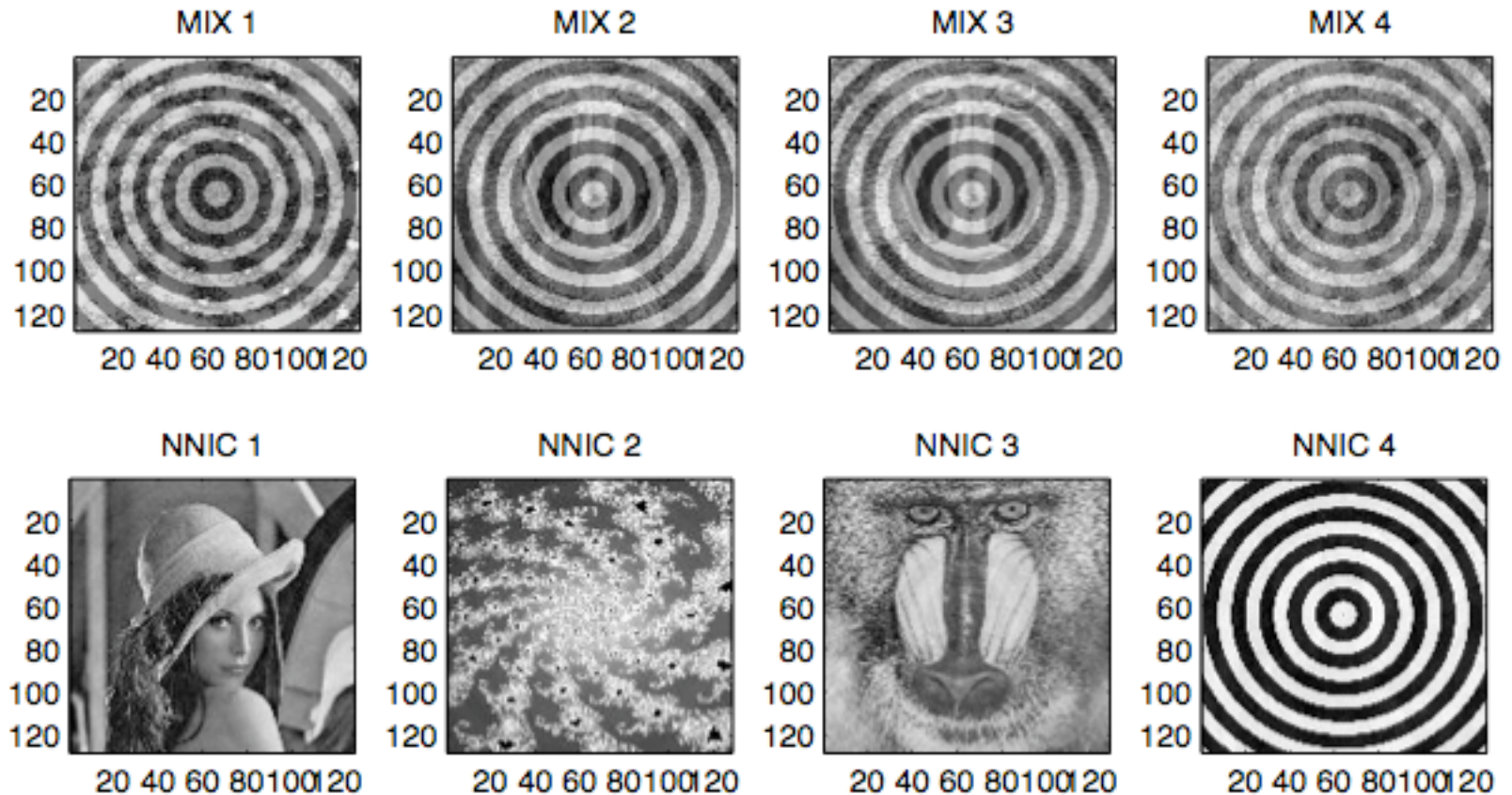
4-source mixture



4-source separation

Simone Fiori

Journal of Machine Learning Research 6 (2005) 743-781



Amari's Recurrent Neural ICA

- Fully recurrent neural network with self-inhibitory connections.

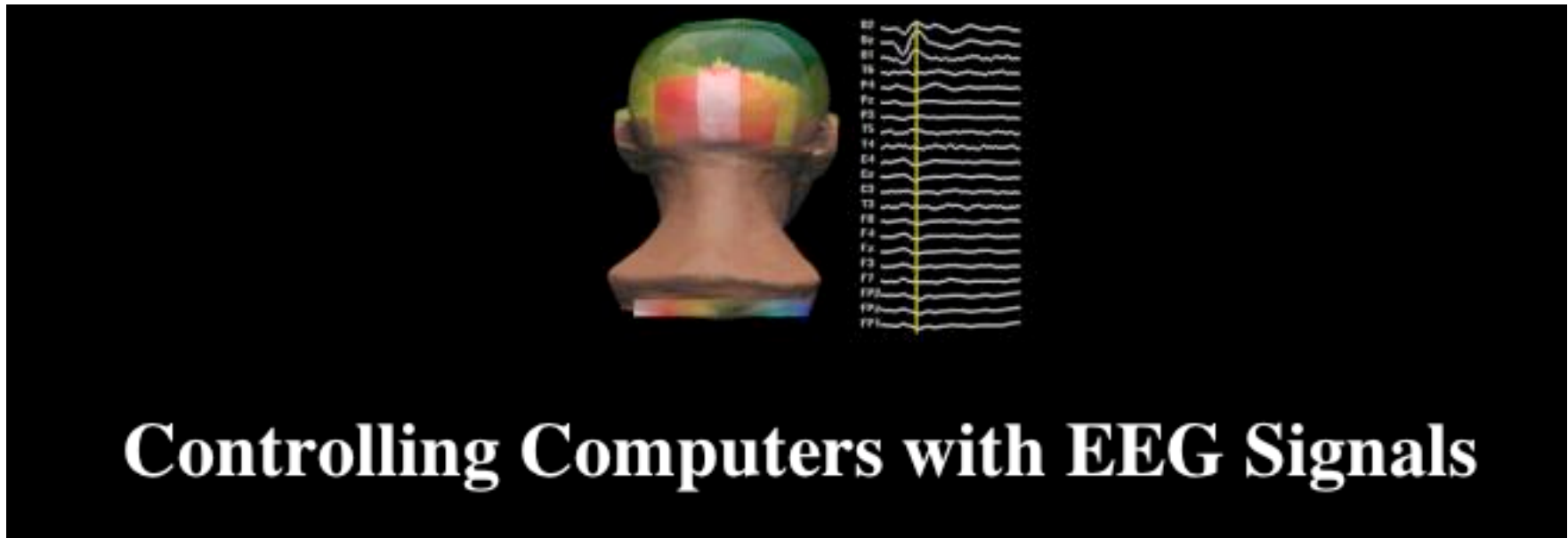
$$\tau_i \frac{dy_i}{dt} + y_i = x_i(t) - \sum_{j=1}^n \hat{w}_{ij}(t) y_j,$$

$$y(t) = (I + \hat{W}(t))^{-1} x(t),$$

$$y(t) = x(t) - \hat{W}(t) y(t - \tau),$$

$$\frac{d\hat{W}}{dt} = -\mu(t) [I + \hat{W}] [\Lambda - f(y(t)) g^T(y(t))].$$

Culpepper/Keller Project using:
<http://www.cs.hmc.edu/~bjc/research/>



ICA was used to achieve a separation among 12 EEG channels, used to discriminate between two or three modes of thought.

Eye-blink EEG removal (Peterson and Anderson, 1999)

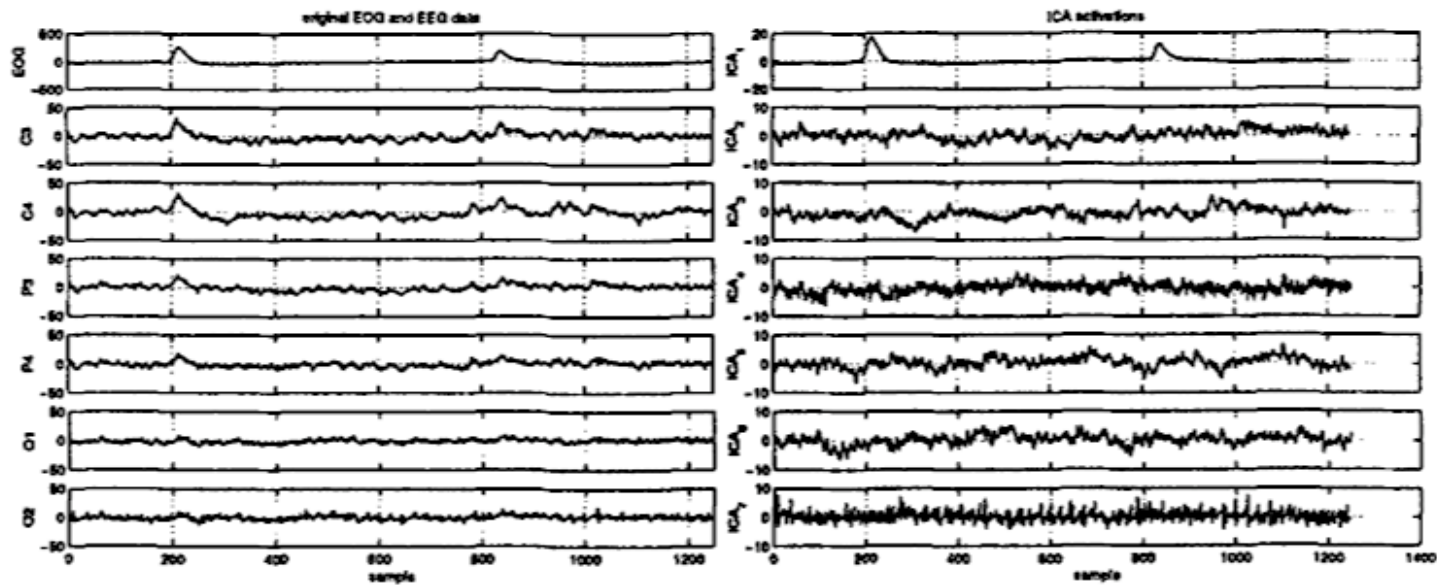


Fig. 1. Eye blink subtraction with ICA

Financial Application

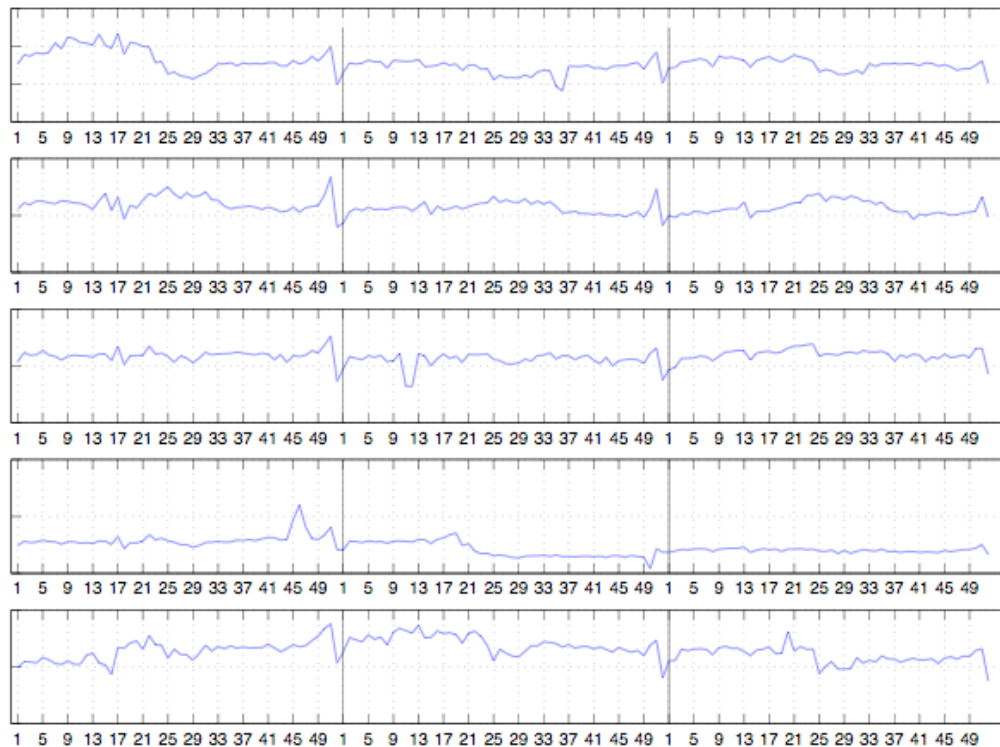


Figure 13: (from Kiviluoto and Oja, 1998). *Five samples of the original cashflow time series (mean removed, normalized to unit standard deviation). Horizontal axis: time in weeks.*

Financial Application

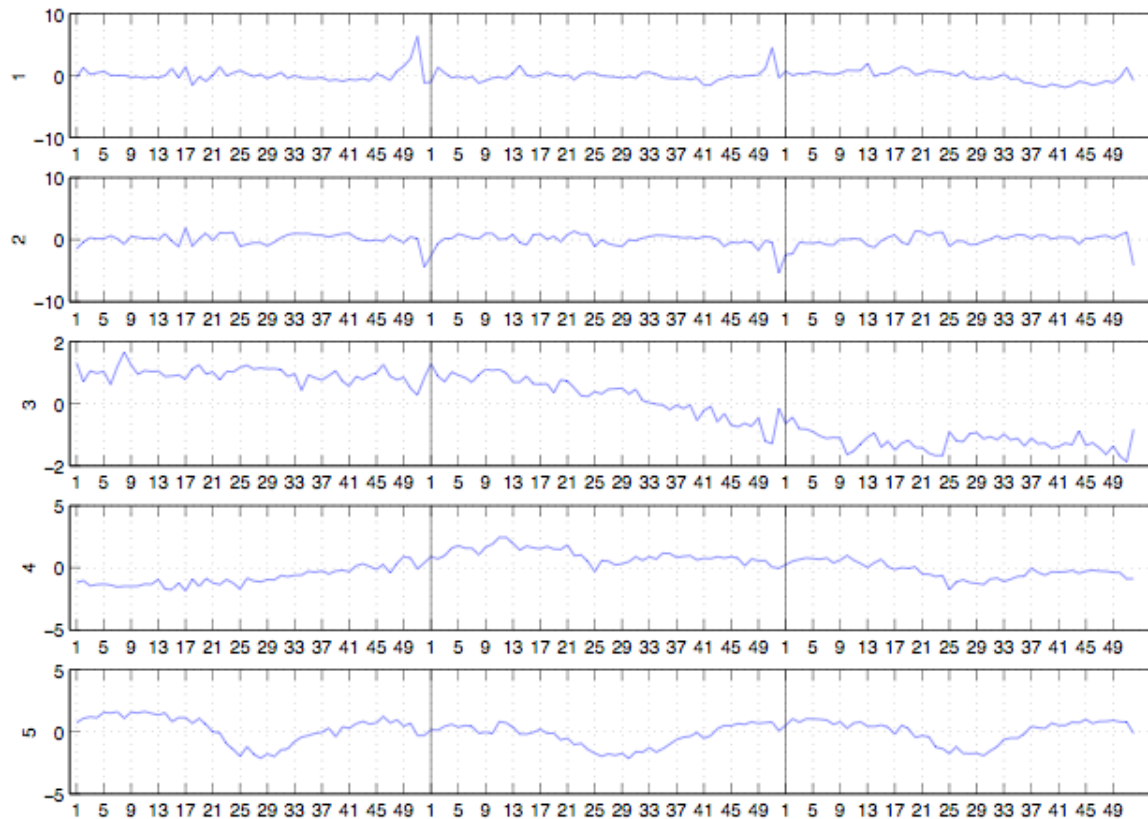


Figure 14: (from Kiviluoto and Oja, 1998). *Four independent components or fundamental factors found from the cashflow data.*

Image Noise Removal App.

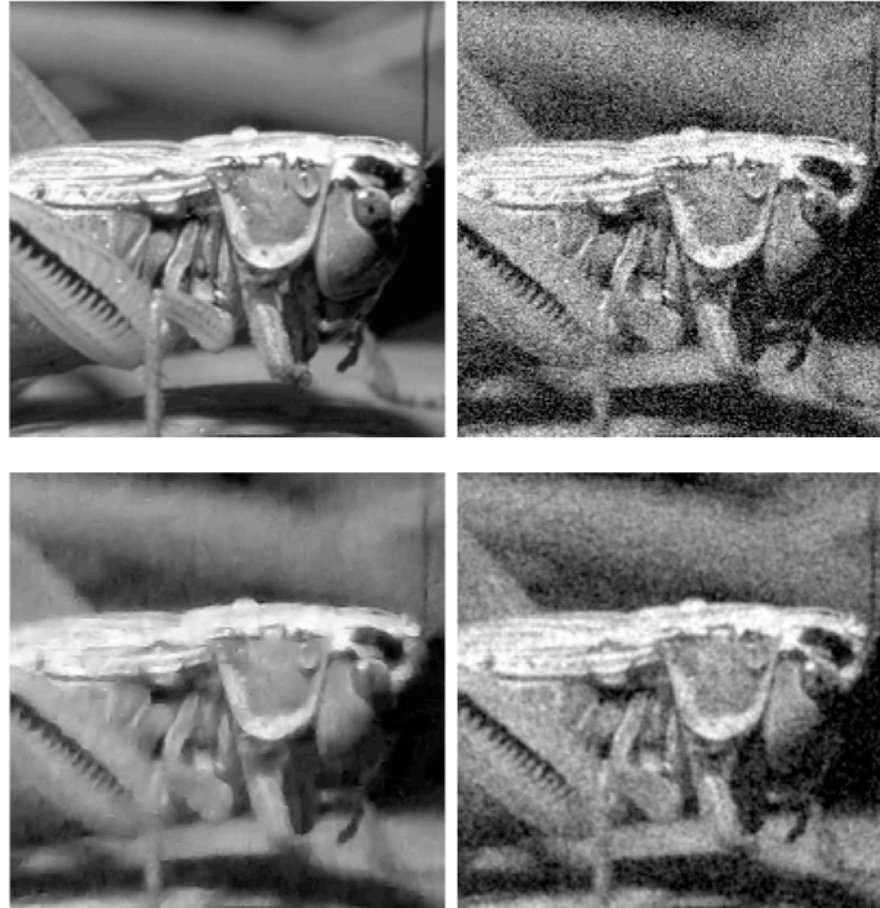


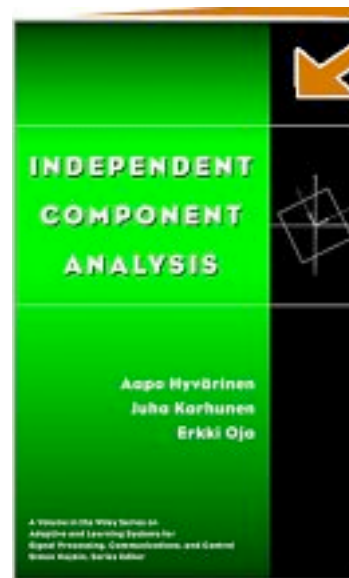
Figure 15: (from Hyvärinen, 1999d). *An experiment in denoising. Upper left: original image. Upper right: original image corrupted with noise; the noise level is 50 %. Lower left: the recovered image after applying sparse code shrinkage. Lower right: for comparison, a wiener filtered image.*

From Aapo Hyvärinen and Erkki Oja
Neural Networks, 13(4-5):411-430, 200

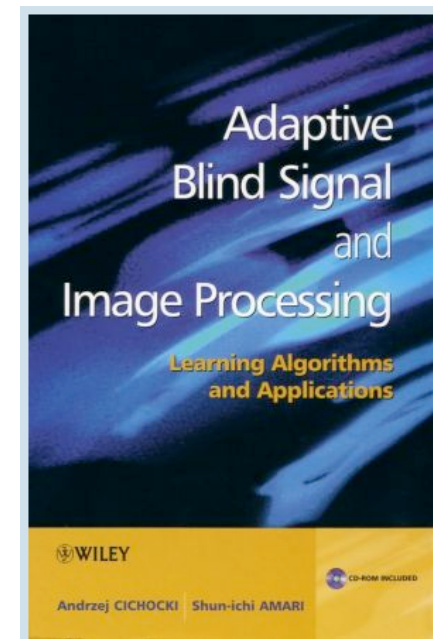
ICA Sources



James V. Stone



Aapo Hyvärinen,
Juha Karhunen,
Erkki Oja



Andrzej Cichocki,
Shun-ichi Amari



6th International Conference on Independent Component Analysis and Blind Source Separation

ICA Shortcomings

- ICA does not take sequence or structure into account; it is purely statistical.
- It might be possible to do better by considering the ignored information.