CS463/516

Lecture 12

Independent Component Analysis (ICA)

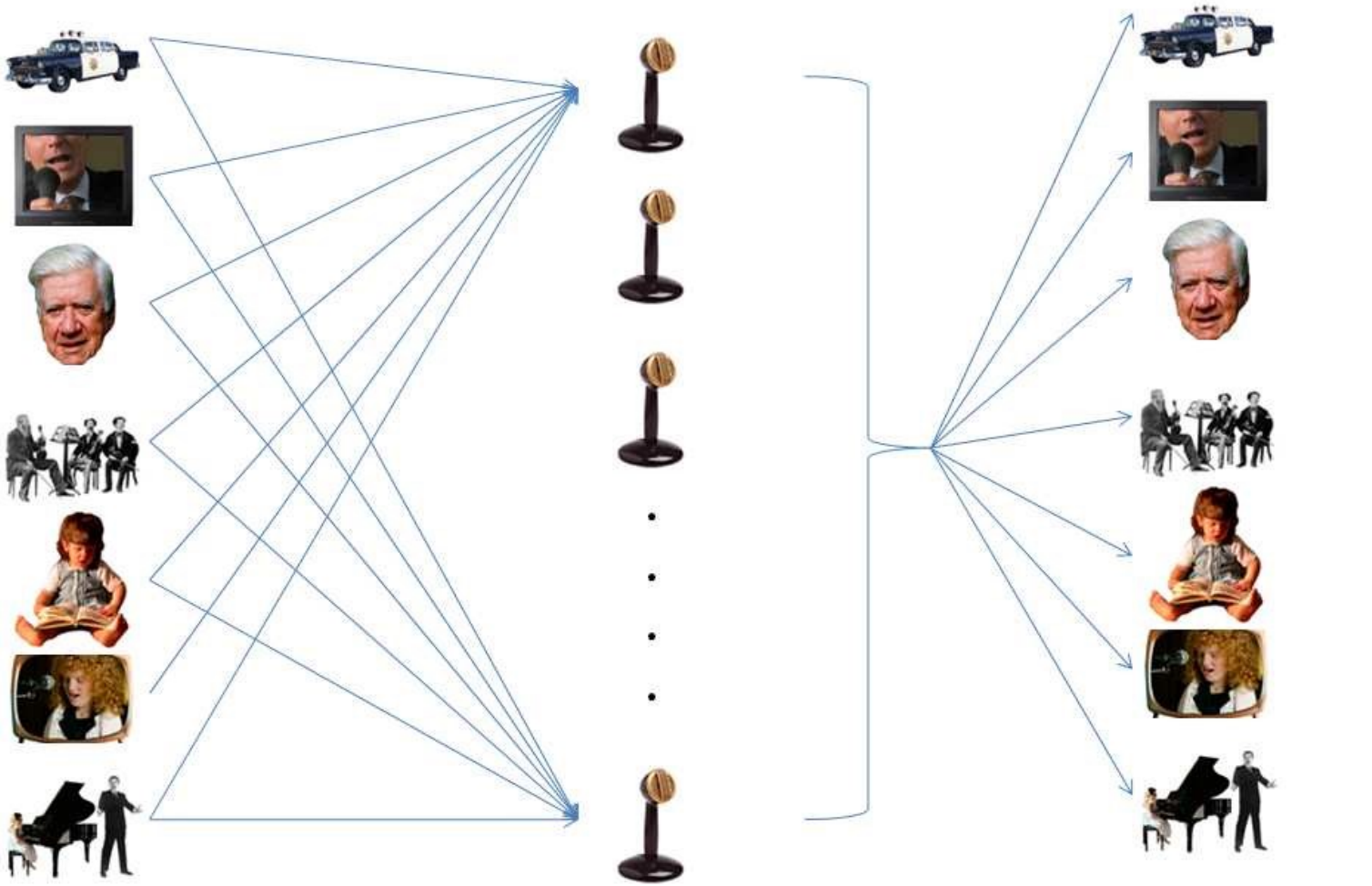
Electroencephalography (EEG)

Event-related analysis

Time-frequency decomposition

Independent component analysis (ICA)

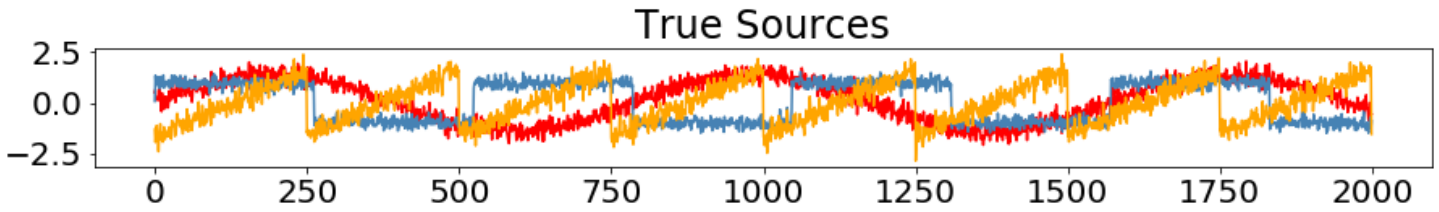
- ICA is a computational method for separating a multivariate signal into additive subcomponents
- ICA is a special case of *blind source separation*
- Example: **cocktail party problem** – many people (sources) talking simultaneously in a room, with an equal number of sound recorders distributed throughout the room
- The goal is to reconstruct each person's individual speech signal (components) from the *mixture* recorded by all microphones
- 'independent component' comes from the fact that ICA maximizes the statistical independence of estimated components, usually by maximizing non-Gaussianity
- $\boldsymbol{S = WX}$
  - $\boldsymbol{S}$ the sources we want to estimate
  - $\boldsymbol{W}$ the 'un-mixing' matrix
  - $\boldsymbol{X}$ the 'raw data' that we measure, assumed to arise from a linear mixture of sources

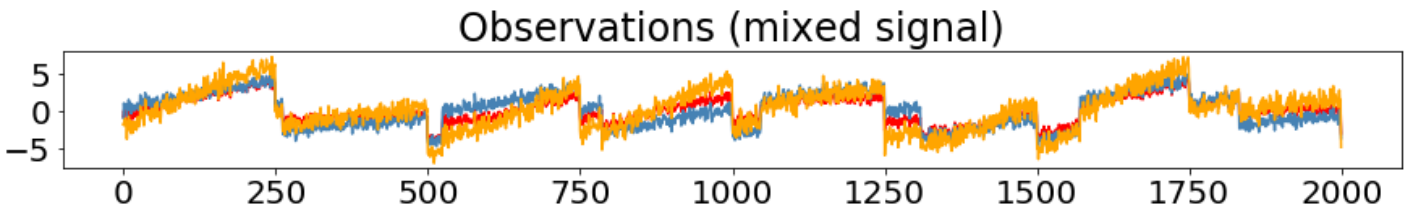


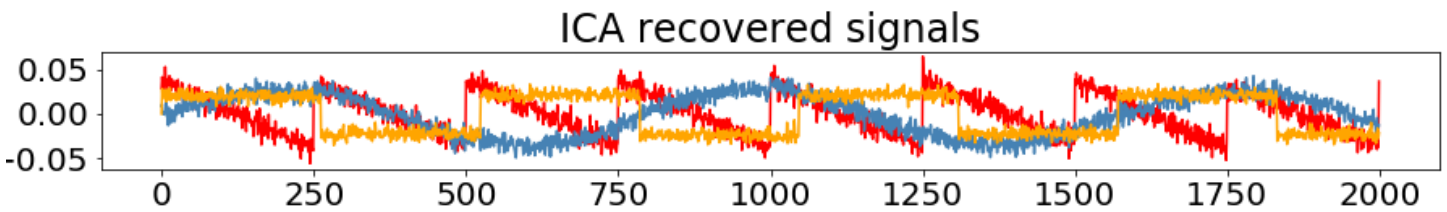
Simple python example

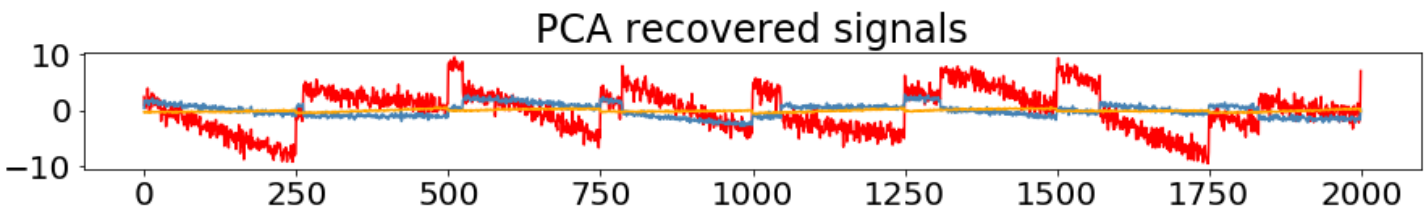
Generate true signal + some noise (*brain activity*)
Mix the true signal into a set of observations (*EEG*)
Run ICA to recover the true signals

```
# Generate sample data
np.random.seed(0)
n_samples = 2000
time = np.linspace(0, 8, n_samples)

s1 = np.sin(2 * time)  # Signal 1 : sinusoidal signal
s2 = np.sign(np.sin(3 * time))  # Signal 2 : square signal
s3 = signal.sawtooth(2 * np.pi * time)  # Signal 3: saw tooth
signal

S = np.c_[s1, s2, s3]
S += 0.2 * np.random.normal(size=S.shape)  # Add noise

S /= S.std(axis=0)  # Standardize data
# Mix data
A = np.array([[1, 1, 1], [0.5, 2, 1.0], [1.5, 1.0, 2.0]])  #
Mixing matrix
X = np.dot(S, A.T)  # Generate observations

# Compute ICA
ica = FastICA(n_components=3)
S_ = ica.fit_transform(X)  # Reconstruct signals
A_ = ica.mixing_  # Get estimated mixing matrix
```







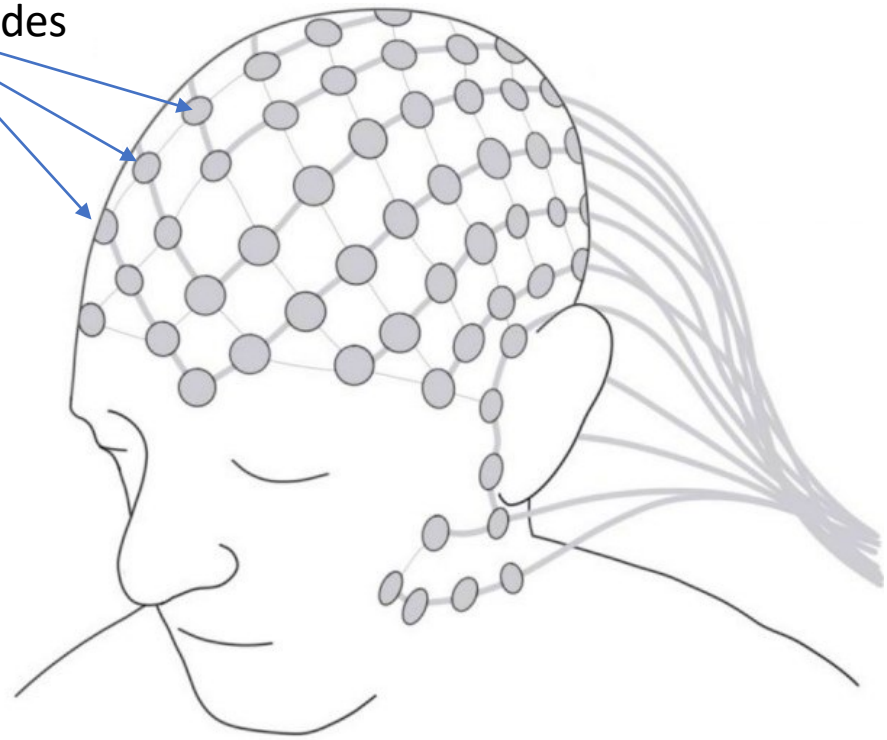


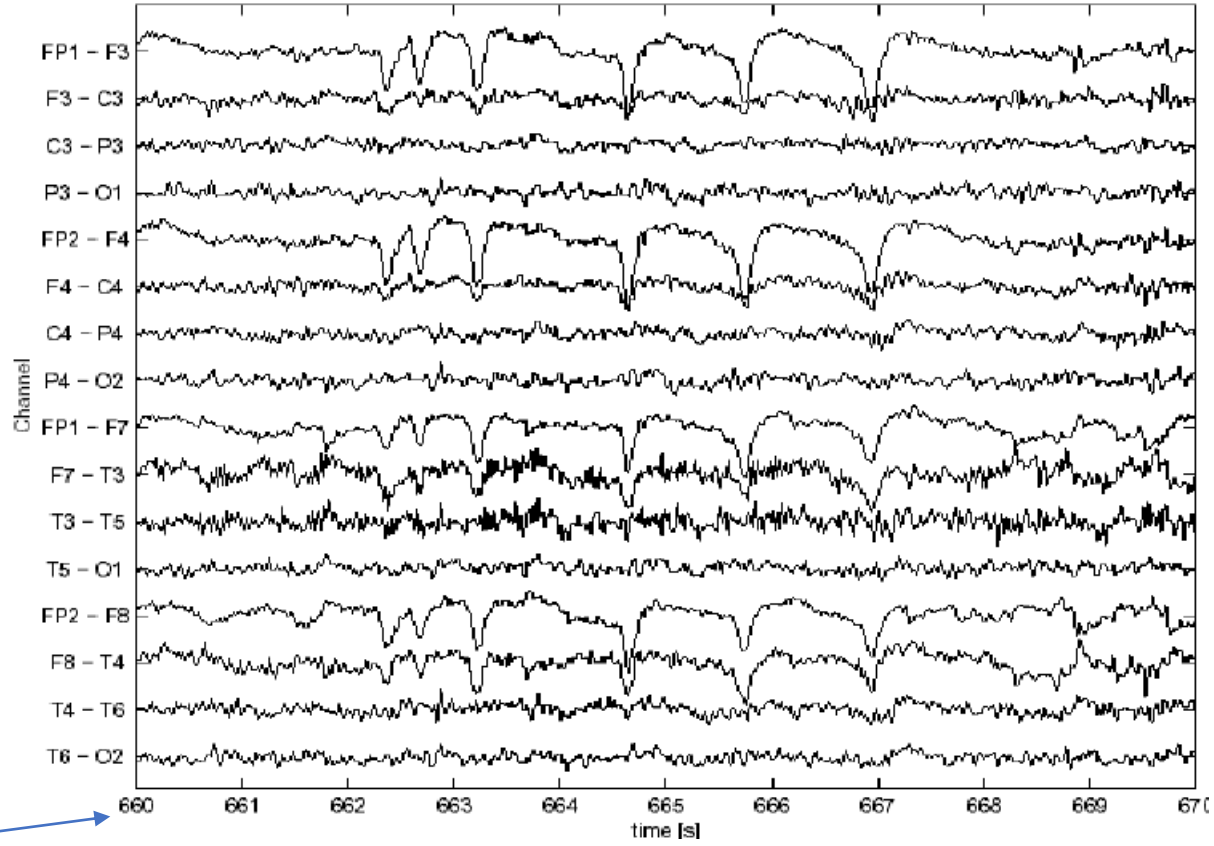
PCA does not recover the true sources but ICA does!

ICA for neuroimaging

- Let $\boldsymbol{X}$ be the input matrix (mixed signal) that we want to separate using ICA
- $\boldsymbol{X}$ has M columns (number of time points) and N rows (number of recording sites)
- Example: EEG recording. M is number of time points, N is number of electrodes

N electrodes





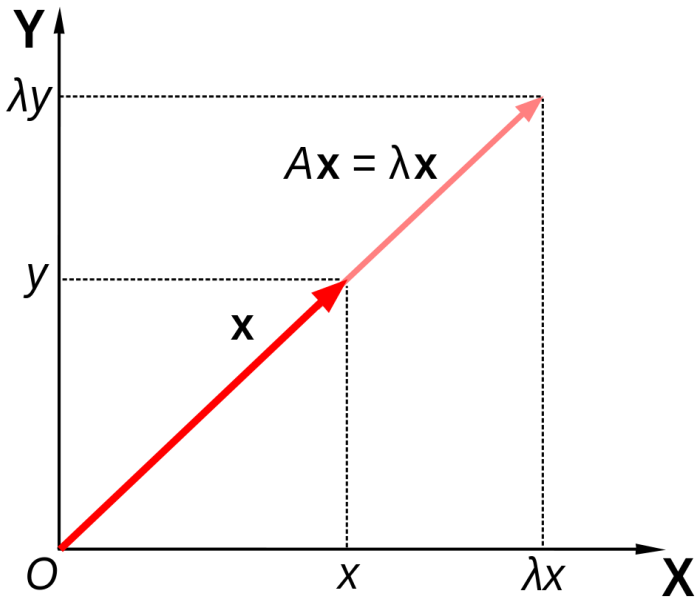
M time points

fastICA algorithm:

- fastICA is an efficient and popular algorithm for ICA
- Input data matrix $\boldsymbol{X}$ must be *prewhitened* before applying the ICA algorithm
- Whitening:
- 1) subtract the mean of each row of the input data: $x_{ij} \leftarrow x_{ij} - \frac{1}{M}\sum_{j'} x_{ij'}$
  - For each $i = 1, \ldots, N$ and $j = 1, \ldots, M$. Each row of $\boldsymbol{X}$ now has mean of 0
- 2) *whitening* the data requires linear transform $\boldsymbol{L}$: $R^{N\times M} \rightarrow R^{N\times M}$ of the centered data, such that components of $\boldsymbol{L}(\boldsymbol{X})$ are uncorrelated and have variance one.
- Hence, if $\boldsymbol{X}$ is the centered data matrix, covariance of $\boldsymbol{L}(\boldsymbol{X})$ is the $(N \times N)$ dimensional identity matrix
- Can whiten using eigenvalue decomposition on covariance matrix of centered $\boldsymbol{X}$
- $E\{\boldsymbol{X}\boldsymbol{X}^T\} = \boldsymbol{E}\boldsymbol{D}\boldsymbol{E}^T$, where $\boldsymbol{E}$ is the matrix of eigenvectors and $\boldsymbol{D}$ is the diagonal matrix of eigenvalues. Whitened data matrix is thus: $\boldsymbol{X} \leftarrow \boldsymbol{D}^{-1/2}\boldsymbol{E}^T\boldsymbol{X}$

Eigenvalue decomposition of covariance matrix

- In linear algebra, *eigendecomposition* is the factorization of a matrix into a canonical form whereby the matrix is represented in terms of its eigenvalues and eigenvectors
- Non-zero vector $\boldsymbol{v}$ of dimension N is an eigenvector of square $N \times N$ matrix A if:
- $\boldsymbol{Av} = \lambda \boldsymbol{v}$, where λ is a scalar termed the eigenvalue corresponding to $\boldsymbol{v}$
- Eigendecomposition: let $\boldsymbol{A}$ be square $n \times m$ matrix with n linearly independent eigenvectors q_i (where $i = 1 \ldots n$), then $\boldsymbol{A}$ can be factorized as $\boldsymbol{A} = \boldsymbol{QVQ}^{-1}$
  - Where $\boldsymbol{Q}$ is the square $n \times n$ matrix whose i^{th} column is the eigenvector q_i of $\boldsymbol{A}$, and $\boldsymbol{V}$ is the diagonal matrix whose diagonal elements are corresponding eigenvalues, $V_{ii} = \lambda_i$
  - Proof: $\boldsymbol{Av} = \lambda \boldsymbol{v} \Rightarrow \boldsymbol{AQ} = \boldsymbol{QV} \Rightarrow \boldsymbol{A} = \boldsymbol{QVQ}^{-1}$



Matrix A acts by stretching the vector $\mathbf{x}$, not changing its direction, so $\mathbf{x}$ is an eigenvector of A

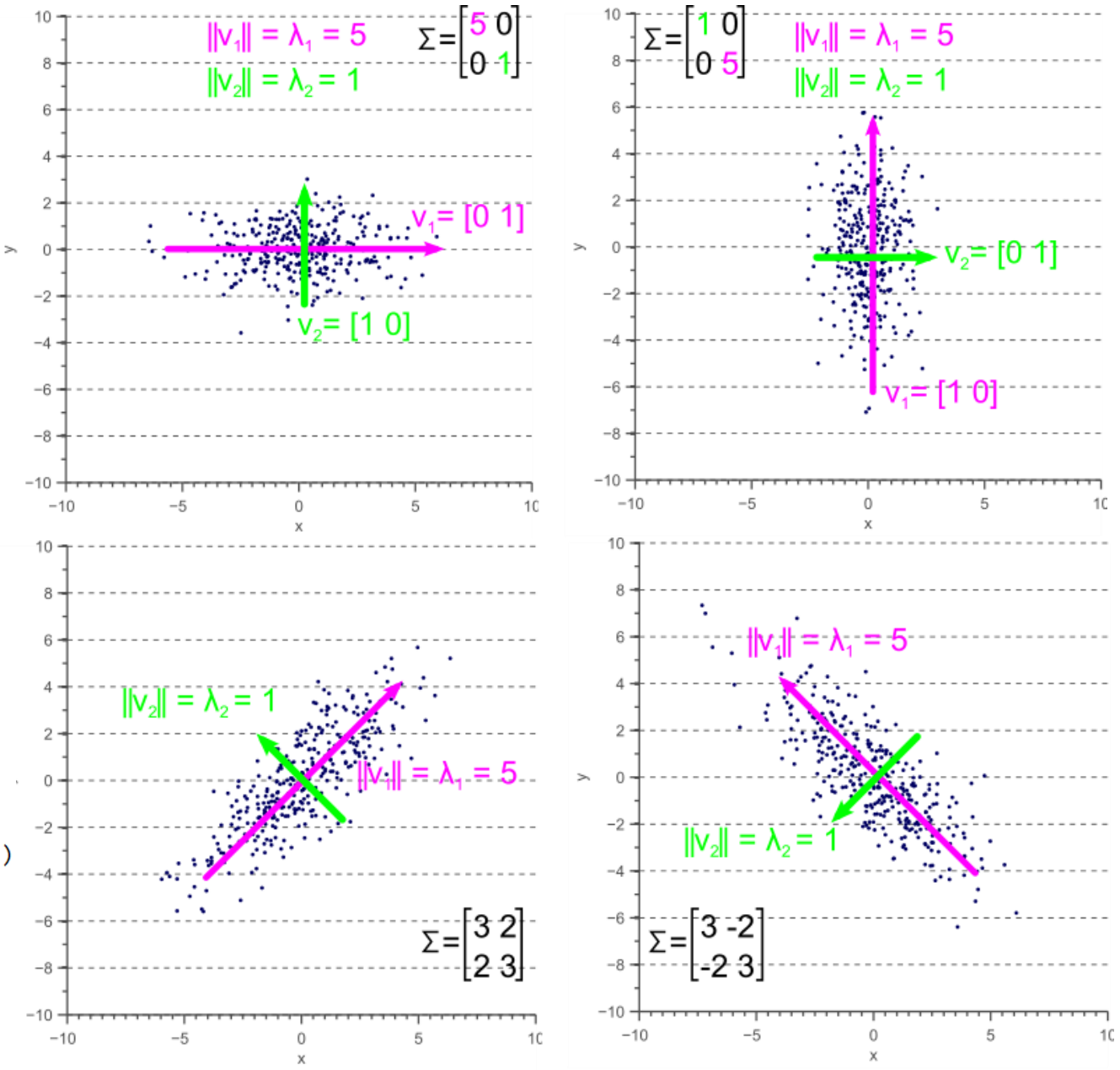
Eigenvalue decomposition of covariance matrix

- a *covariance matrix* $\boldsymbol{\Sigma}$ is a square matrix giving covariance between each pair of elements of a given random vector. Generalizes variance to multiple dimensions
- The first eigenvector of a covariance matrix (eigenvector with largest eigenvalue) is the direction along which dataset has largest variance
  - This eigenvector is also known as the *principle component* of the data
- Example: $\boldsymbol{\Sigma} = \begin{bmatrix} 3 & 2 \\ 2 & 3 \end{bmatrix}$

```
sigma = np.array([[3,2],[2,3]])
w,v = np.linalg.eig(sigma)
```

```
In [139]: w
Out[139]: array([5., 1.])

In [140]: v
Out[140]:
array([[ 0.70710678, -0.70710678],
       [ 0.70710678,  0.70710678]])
```

Eigenvalues represent the variance of the data along the eigenvector directions.

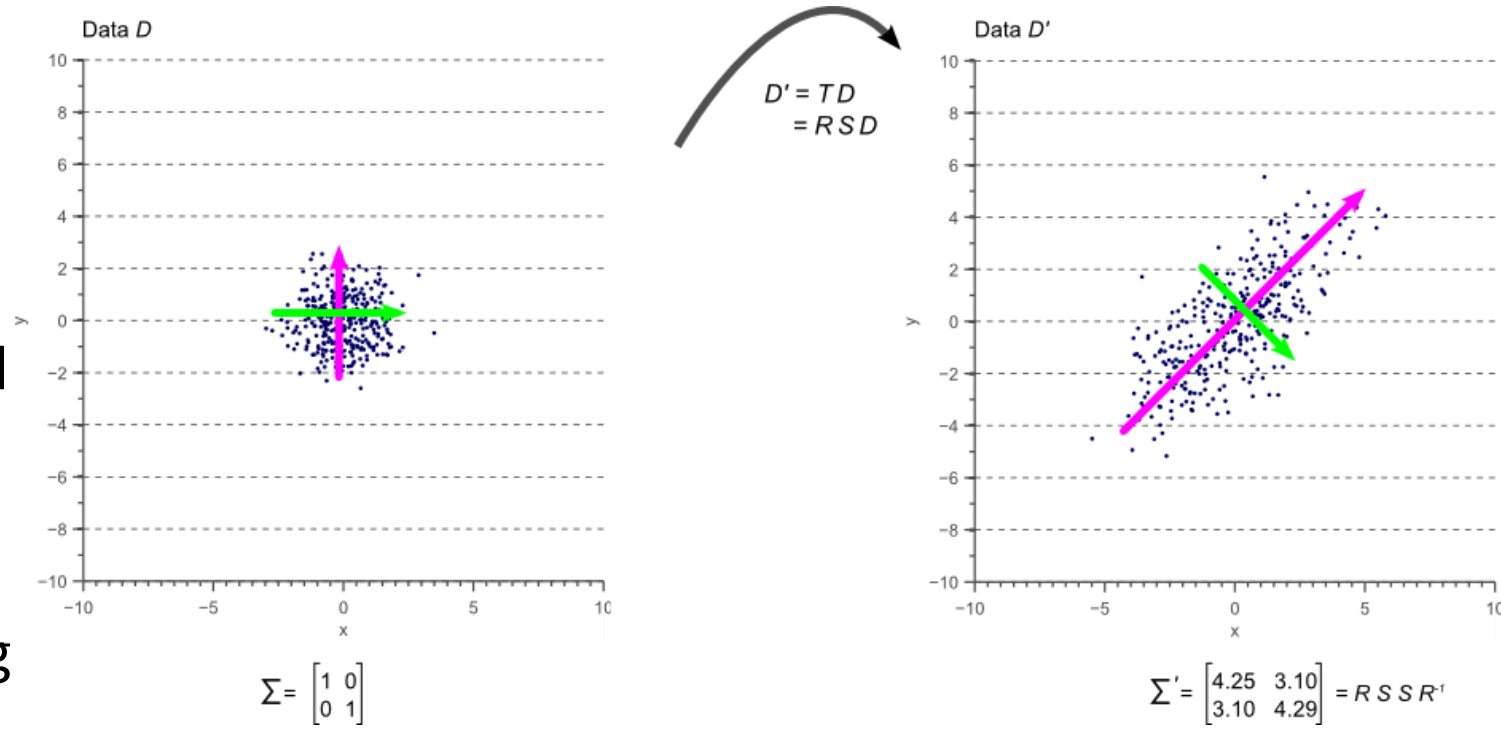


https://www.visiondummy.com/2014/04/geometric-interpretation-covariance-matrix/

Whitening intuition

Why whiten? Basically, reduces complexity of problem ICA needs to solve and helps it converge faster.

- Covariance matrix can be represented as a function of its eigenvalues and eigenvectors $\boldsymbol{\Sigma = VLV^{-1}}$
  - This is also known as the *singular value decomposition* (SVD)
- Can think of $\boldsymbol{V}$ as a rotation matrix and $\sqrt{\boldsymbol{L}}$ as a scaling matrix, further decomposing to:
- $\boldsymbol{\Sigma = RSSR^{-1}}$ where $\boldsymbol{R = V}$ is a rotation matrix and $\boldsymbol{S = \sqrt{L}}$ is a scaling matrix



- Consider the linear transform $\boldsymbol{T = RS}$. Since $\boldsymbol{S}$ is a diagonal scaling matrix, $\boldsymbol{S = S^T}$. Also, since $\boldsymbol{R}$ is an orthogonal matrix, $\boldsymbol{R^{-1} = R^T}$, therefore $\boldsymbol{T^T = (RS)^T = S^T R^T = SR^{-1}}$
- The covariance matrix can thus be written $\boldsymbol{\Sigma = RSSR^{-1}}$
- In other words, if we apply the linear transform defined by $\boldsymbol{T = RS}$ to whitened data $\boldsymbol{D}$, we obtain rotated and scaled data $\boldsymbol{D'}$ with covariance matrix $\boldsymbol{TT^T = \Sigma' = RSSR^{-1}}$

fastICA algorithm: single component extraction

- Iterative algorithm that finds direction for weight vector $\boldsymbol{w} \in R^N$ that maximizes a measure of non-gaussianity of the projection $\boldsymbol{w}^T\boldsymbol{X}$, where $\boldsymbol{X}$ is pre-whitened data matrix and $\boldsymbol{w}$ is a column vector
- To quantify non-gaussianity, fastICA uses the nonquadratic, nonlinear function $f(u)$ and its first derivative $g(u)$ and second derivative $g'(u)$:
- $f(u) = logcosh(u), g(u) = \tanh(u)\,, g'(u) = 1 - \tanh^2(u)$
- Or, $f(u) = -e^{-\frac{u^2}{2}}$, $g(u) = ue^{-\frac{u^2}{2}}$, $g'(u) = (1-u^2)e^{-\frac{u^2}{2}}$ (more robust)

Algorithm:

1) Randomize initial weight vector $\boldsymbol{w}$
2) Set $\boldsymbol{w}^+ = E\{\boldsymbol{X}g(\boldsymbol{w}^T\boldsymbol{X})^T\} - E\{g'(\boldsymbol{w}^T\boldsymbol{X})\}\boldsymbol{w}$
   where $E\{\dots\}$ means average over all column-vectors of $\boldsymbol{X}$
3) Set $\boldsymbol{w} = \frac{\boldsymbol{w}^+}{\|\boldsymbol{w}^+\|}$
4) If not converged repeat from step 2

fastICA algorithm: multiple component extraction

- Inputs:
  - C, the number of desired components
  - $\boldsymbol{X} \in R^{N \times M}$ the pre-whitened matrix, where each column represents an N dimensional sample
- Outputs:
  - $\boldsymbol{W} \in R^{N \times C}$, the un-mixing matrix, each column projects $\boldsymbol{X}$ onto an independent component
  - $\boldsymbol{S} \in R^{C \times M}$, matrix of C independent components, with M samples (typically time points)

Algorithm:

For p in 1 to C:

$\boldsymbol{w}_p =$ random vector of length N

while $\boldsymbol{w}_p$ has not converged

$$\boldsymbol{w}_p = \frac{1}{M}\boldsymbol{X} g\left(\boldsymbol{w}_p^T \boldsymbol{X}\right)^T - \frac{1}{M} g'\left(\boldsymbol{w}_p^T \boldsymbol{X}\right)\mathbf{1}\boldsymbol{w}_p$$

$$\boldsymbol{w}_p = \boldsymbol{w}_p - \sum_{j=1}^{p-1}\left(\boldsymbol{w}_p^T \boldsymbol{w}_j\right)\boldsymbol{w}_j$$

$$\boldsymbol{w}_p = \frac{\boldsymbol{w}_p}{\|\boldsymbol{w}_p\|}$$

Output:

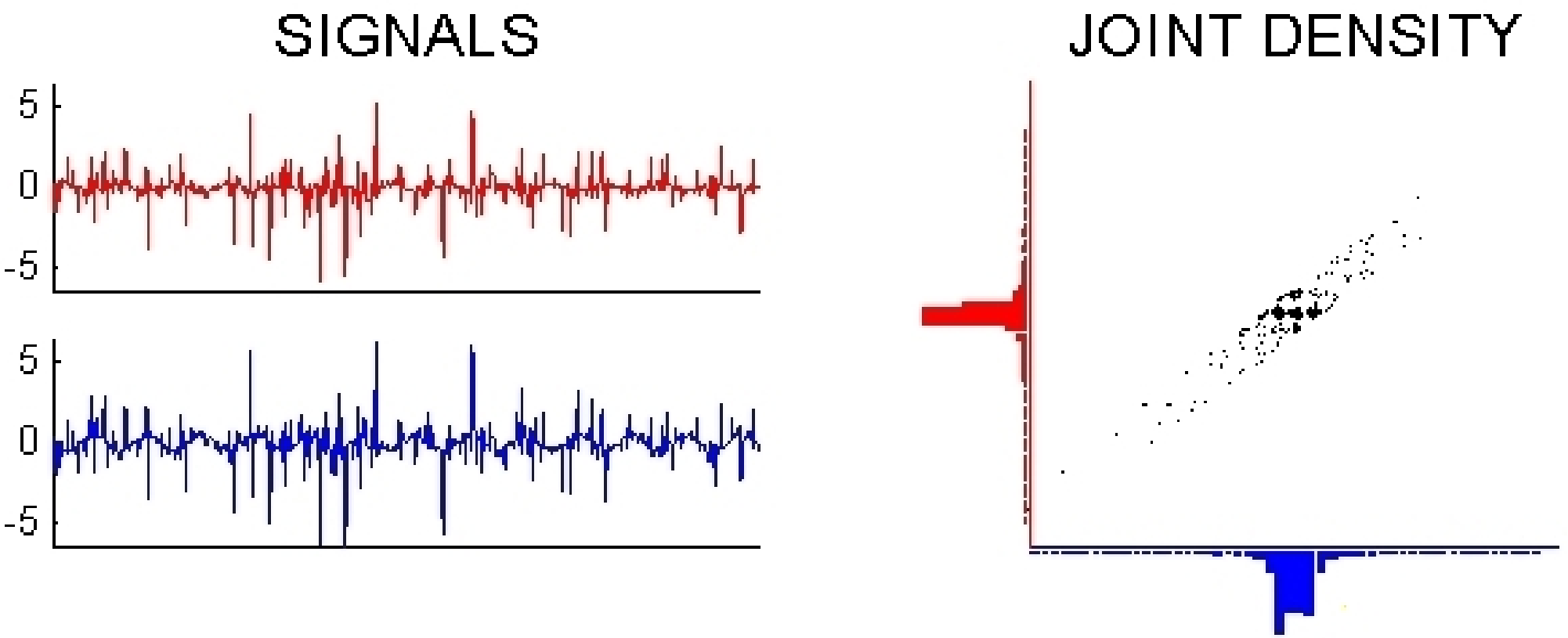
$$\boldsymbol{W} = [w_1, \ldots w_C]$$

$$\boldsymbol{S} = \boldsymbol{W}^T \boldsymbol{X}$$

Where $\mathbf{1}$ is a column vector of 1's of dimension M

http://research.ics.aalto.fi/ica/icademo/

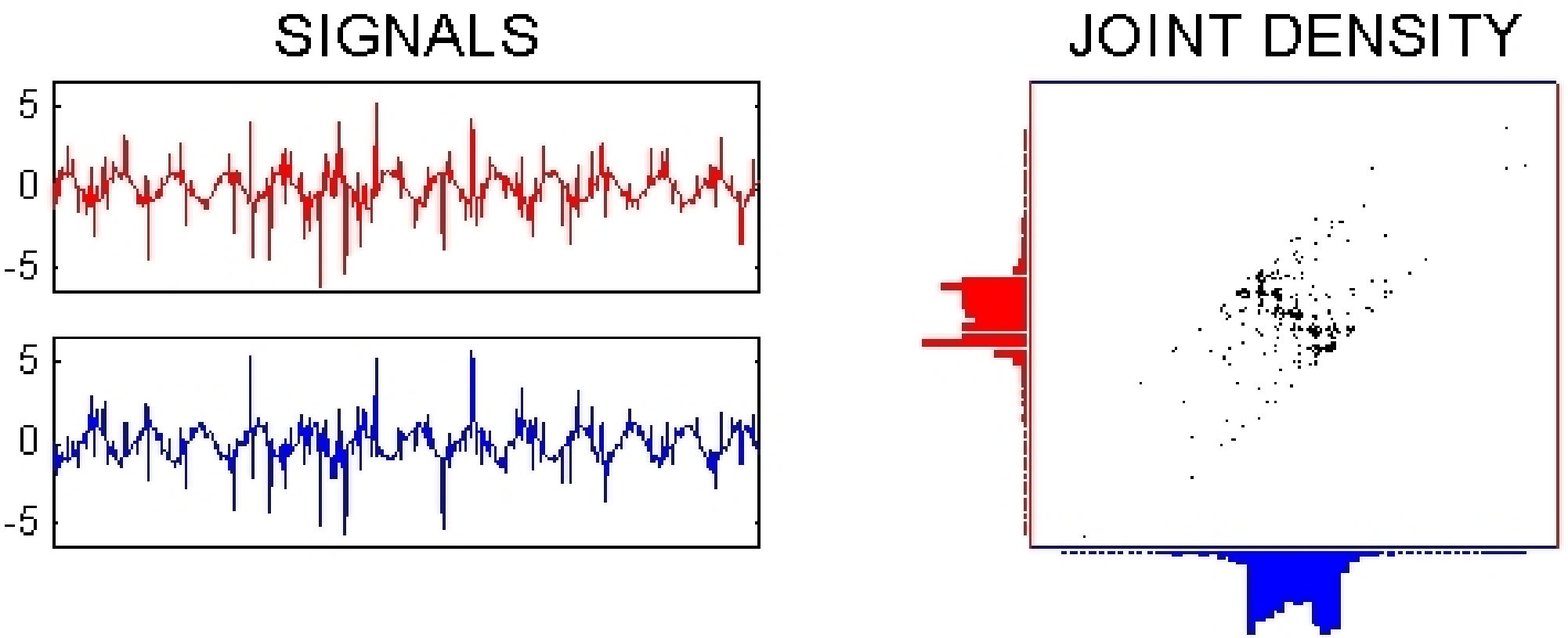
fastICA example



Input signals and density

http://research.ics.aalto.fi/ica/icademo/

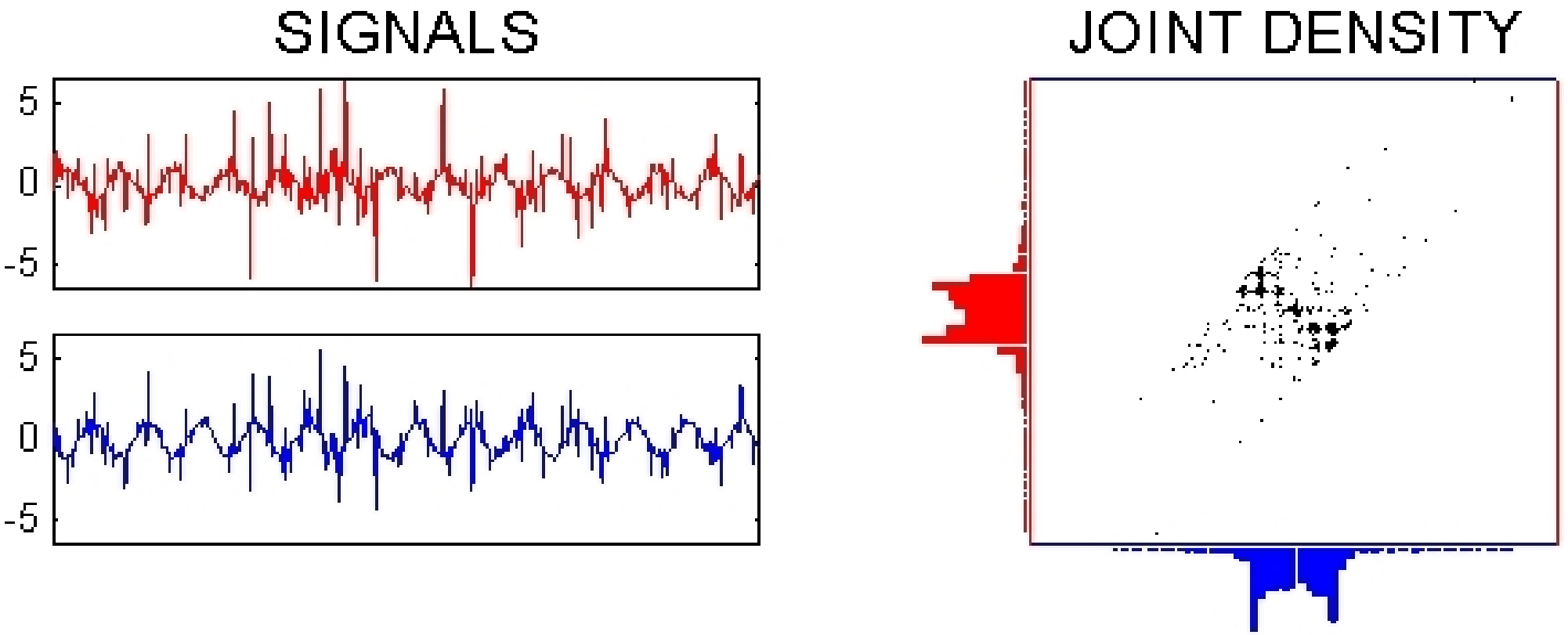
fastICA example



Whitened signals and density

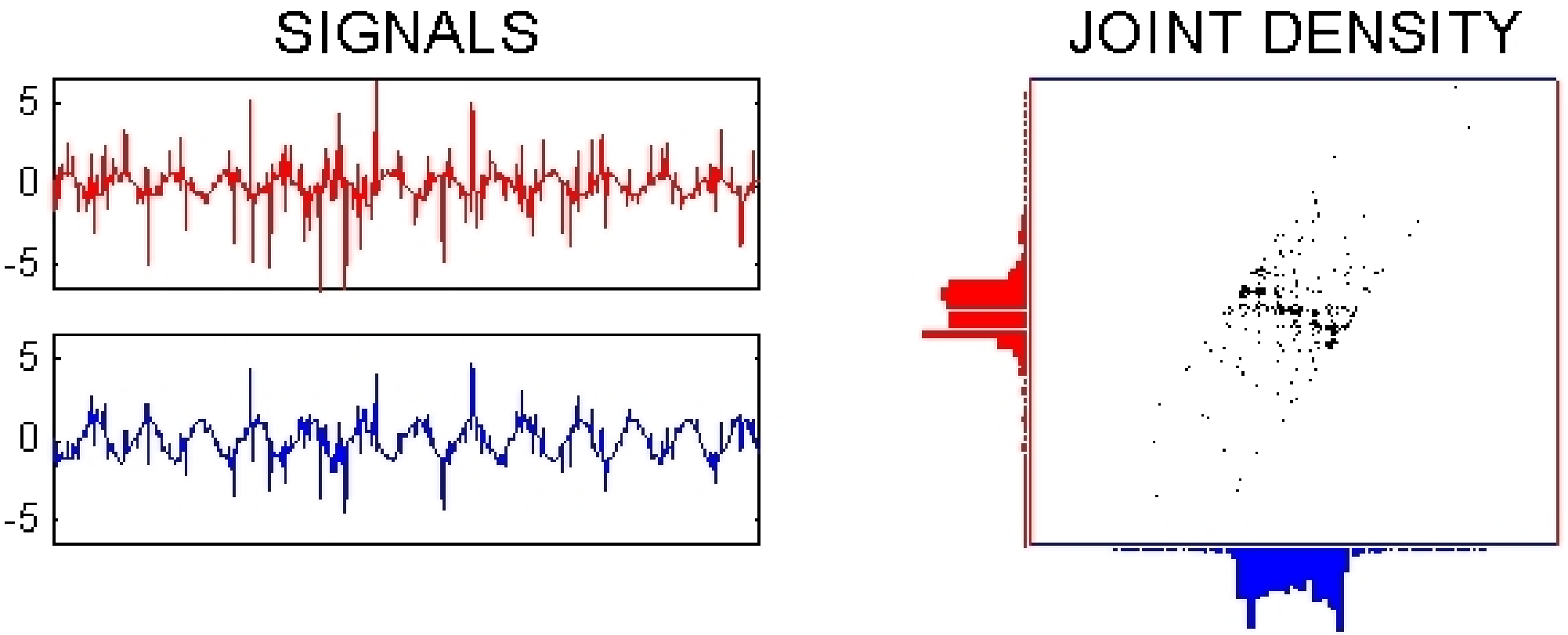
http://research.ics.aalto.fi/ica/icademo/

fastICA example



http://research.ics.aalto.fi/ica/icademo/

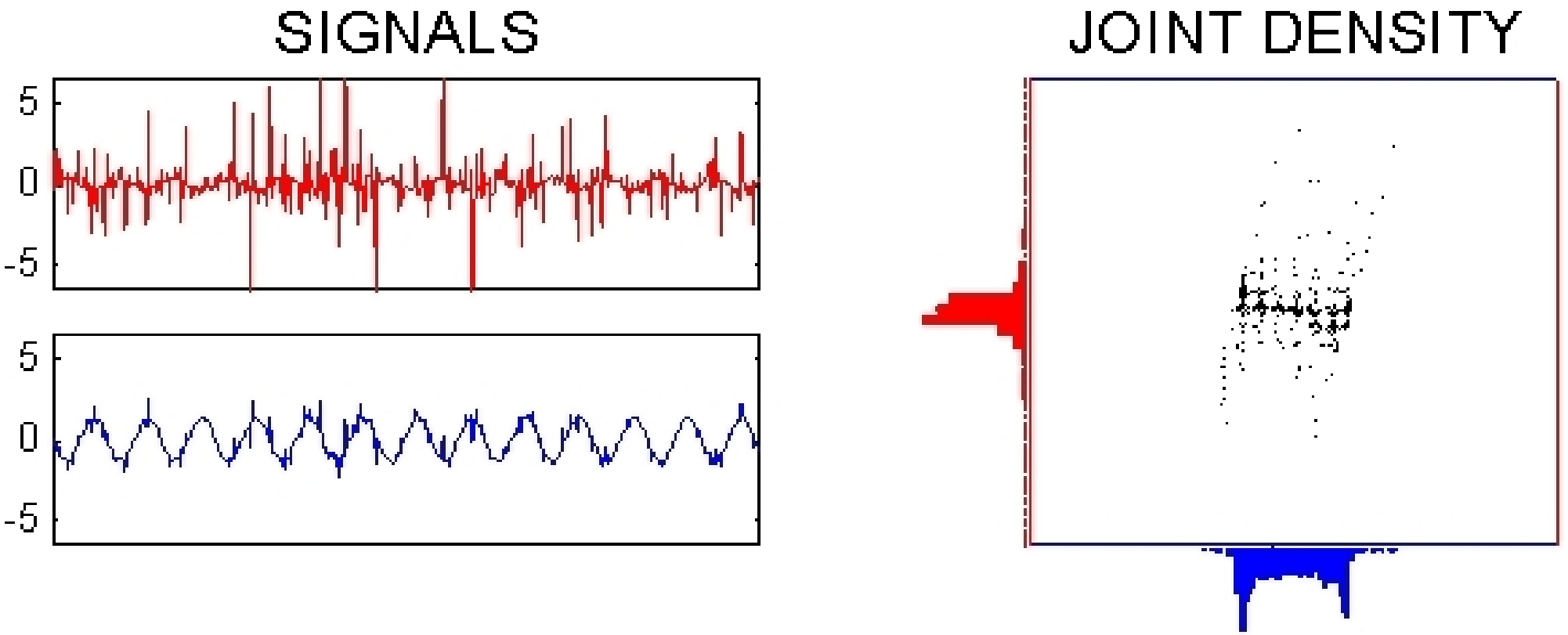
fastICA example



Separated signals after 2 steps of FastICA

http://research.ics.aalto.fi/ica/icademo/

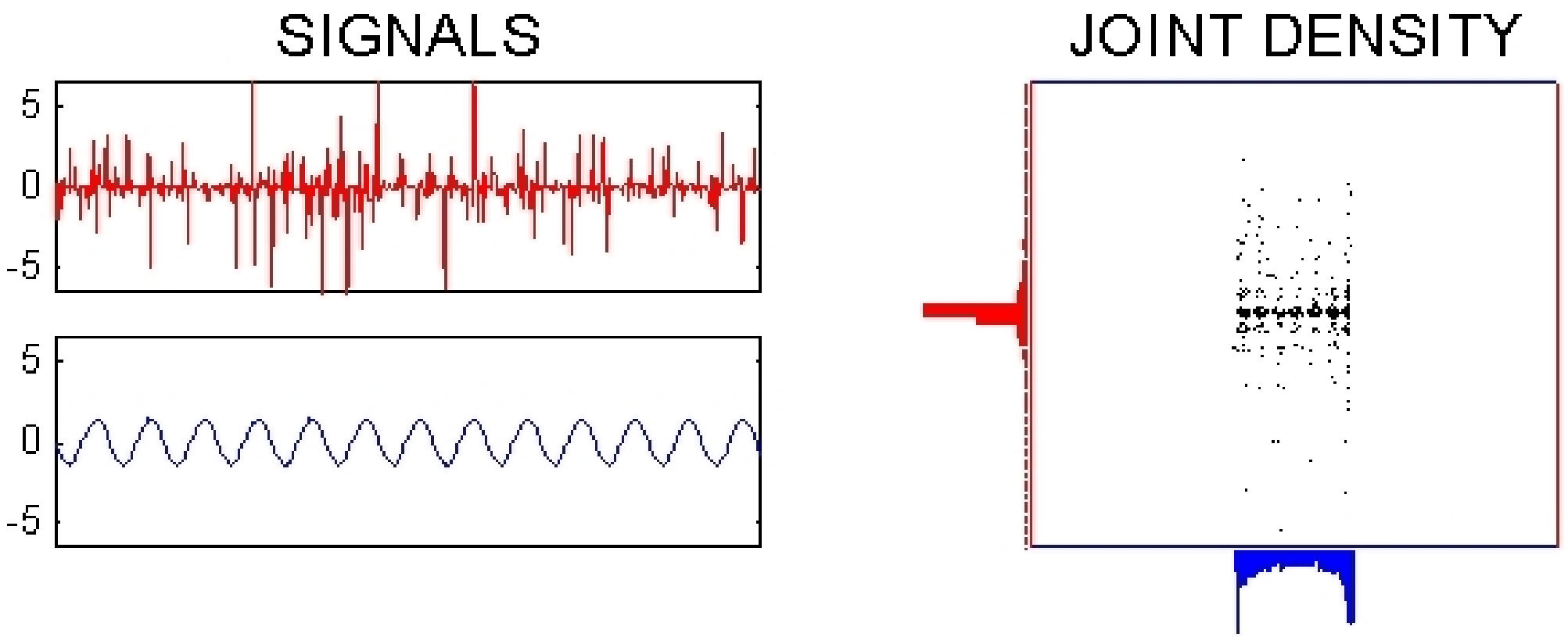
fastICA example



Separated signals after 3 steps of FastICA

http://research.ics.aalto.fi/ica/icademo/

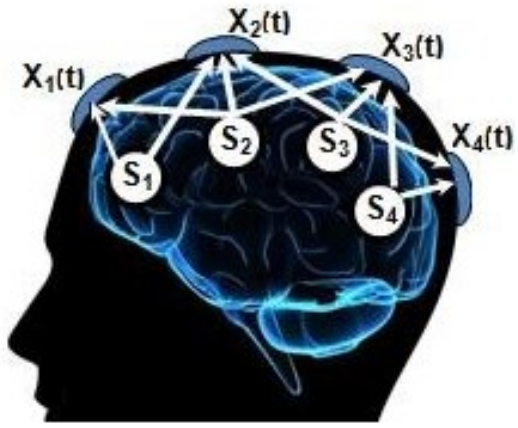
fastICA example

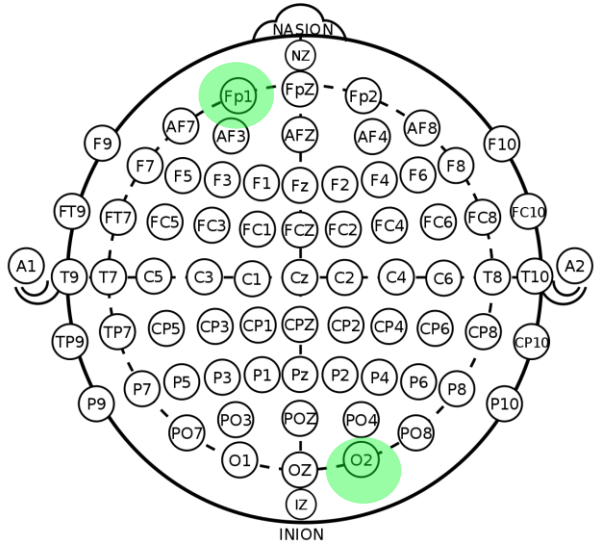


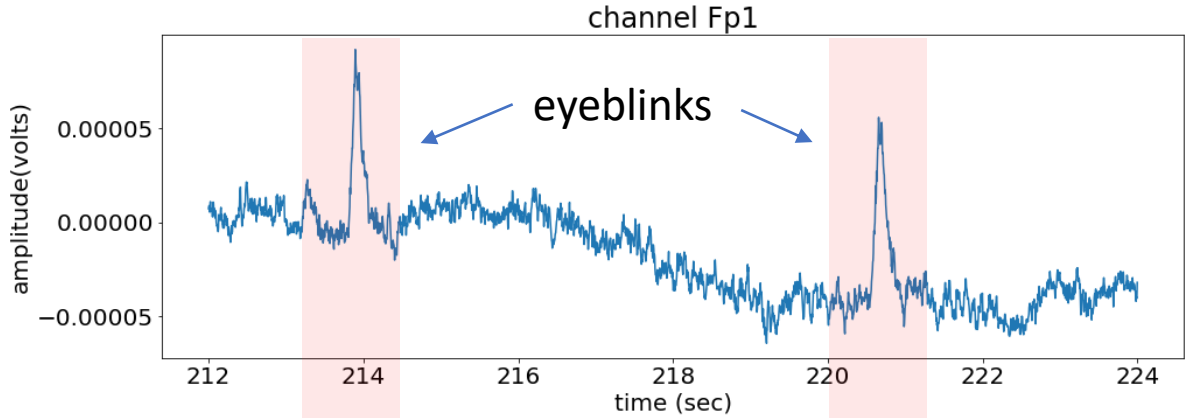
Separated signals after 4 steps of FastICA

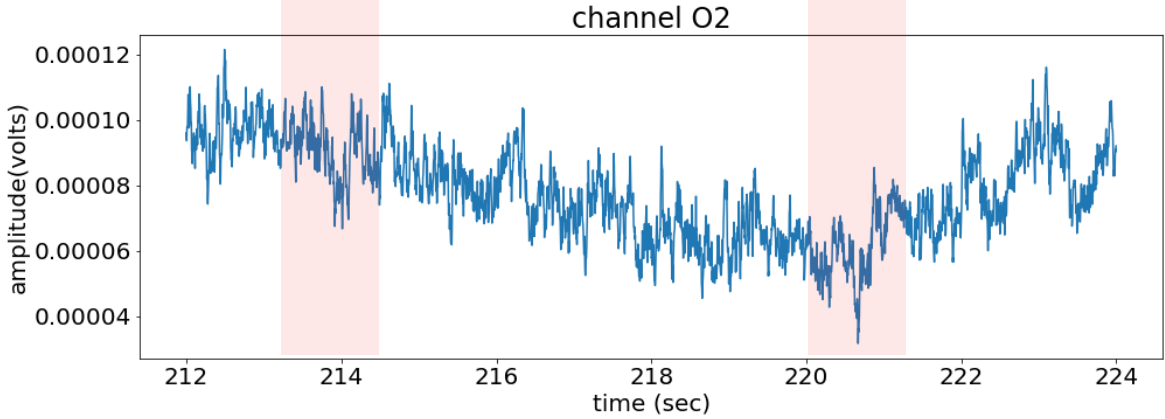
fastICA applied to EEG data

- EEG cap typically features 64 separate recording sites (electrodes)
- each of these electrodes measures a *mixture* of brain sources, therefore ICA is a natural algorithm to apply
- $\boldsymbol{X} \in R^{N \times M}$ is the *raw data*
  - where M is number of time points, N number of electrodes
- $\boldsymbol{W} \in R^{N \times C}$ is the *unmixing matrix*
  - Where N is number of electrodes, C is number of components (typically equal to number of electrodes)





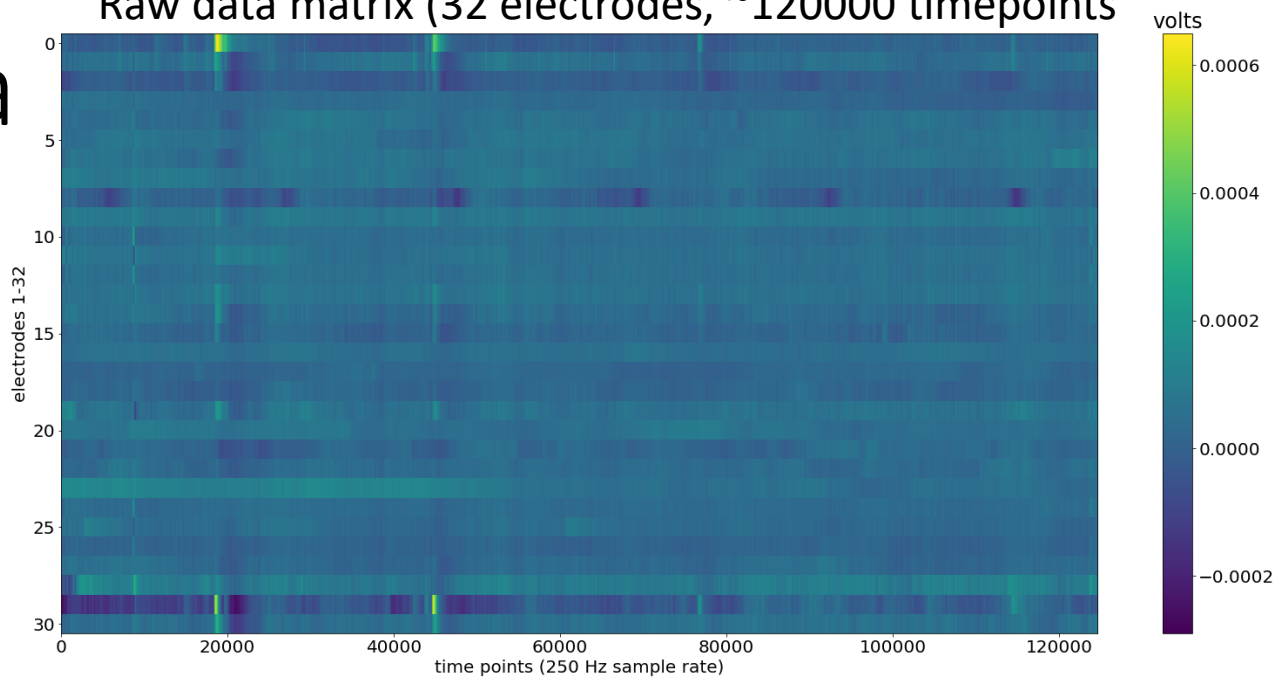




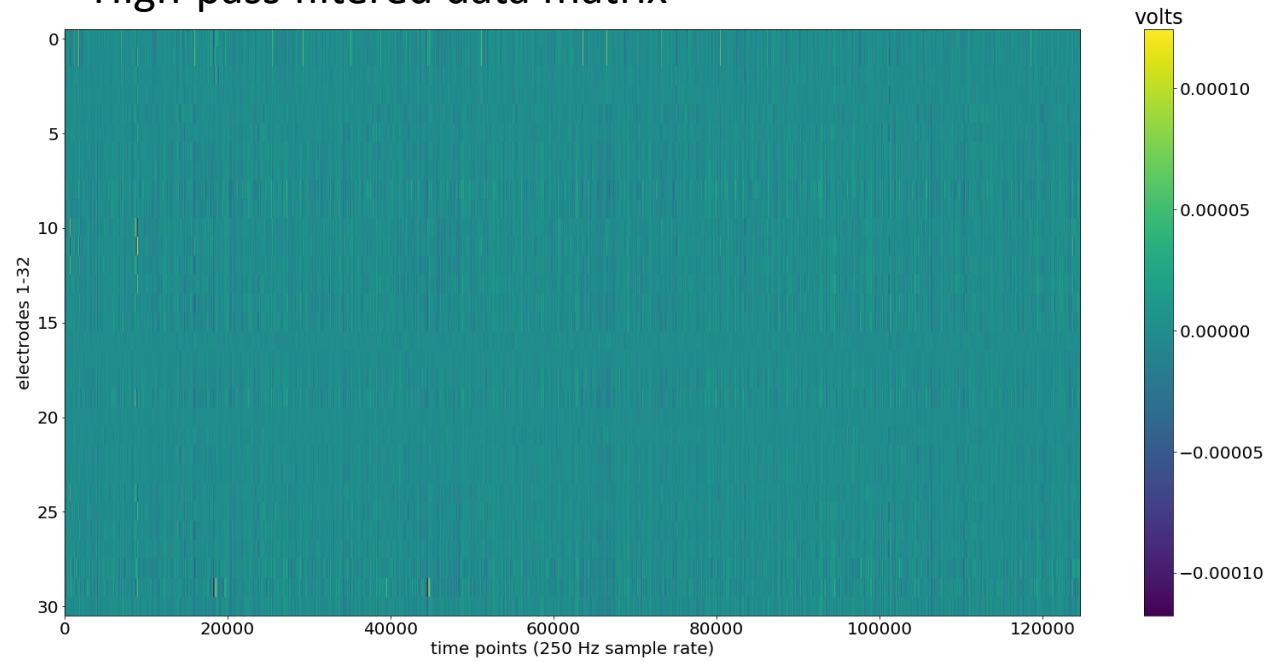
fastICA applied to EEG data

- Some simple pre-processing steps before running fastICA:
- Step 1: remove any ‘bad’ electrodes, electrodes which were poorly connected and contain only noise
  - Here all electrodes seem to be fine
- Step 2: high-pass filter the raw data
  - Remove all frequencies <1 Hz
- Why high pass filter data before running ICA?
  - Removes ‘baseline drift’ which influences ICA towards lower frequencies
  - In EEG, we typically want ICA to focus on frequencies from 3-15Hz (or higher) so low frequencies (<1 Hz) are removed
- These steps are done before we even call fastICA (which will center/prewhiten internally)

Raw data matrix (32 electrodes, ~120000 timepoints



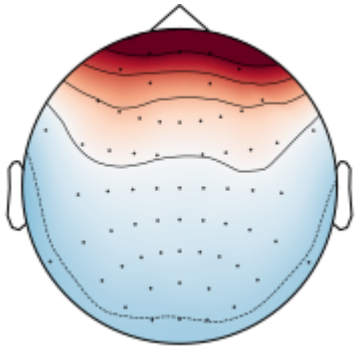
High-pass filtered data matrix



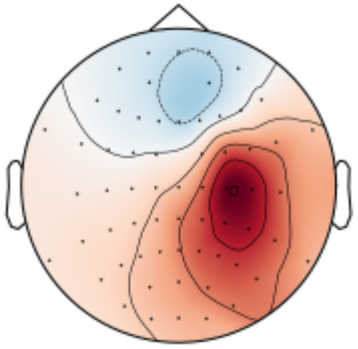
fastICA applied to EEG data : example of 2 components

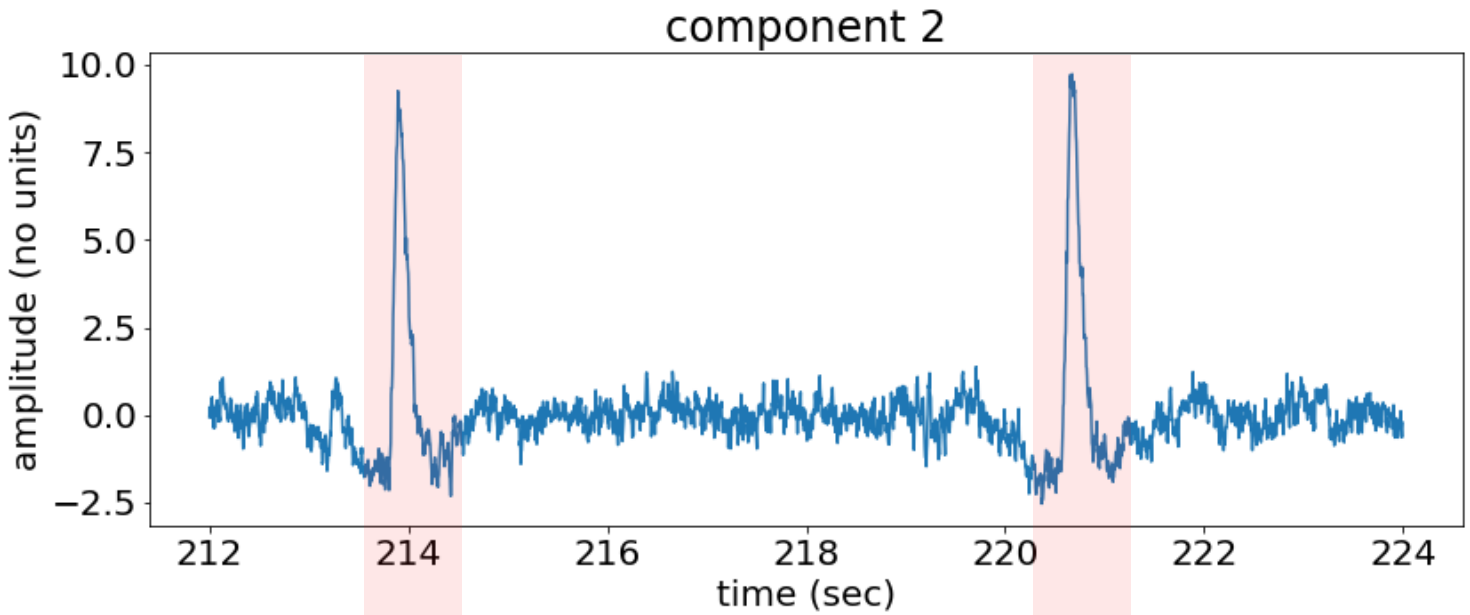
- Example of two components given by ICA decomposition of EEG data.
- a) eye-blink component. Captures signals caused by electromyographic (muscular) activity when the eyes blink.
- b) neuronal component (captures signals coming from brain's parietal lobe, as can be seen by the strong alpha wave from ~217-220 seconds

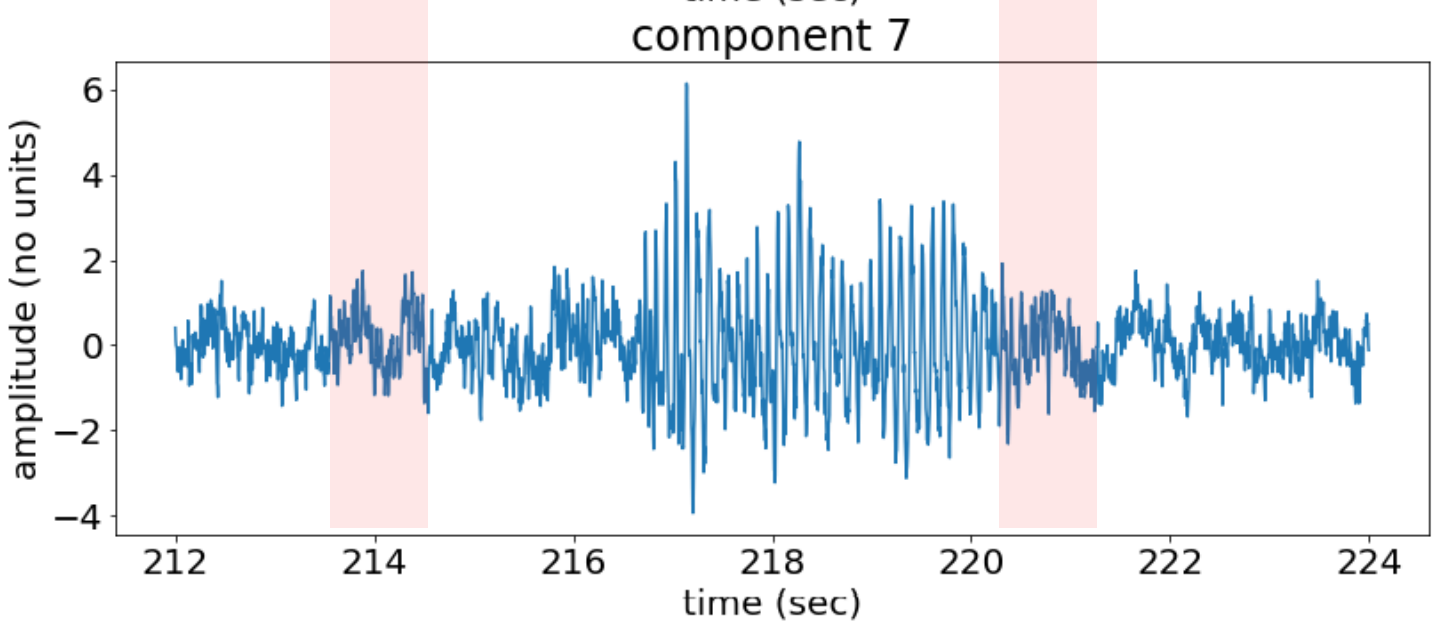
a)



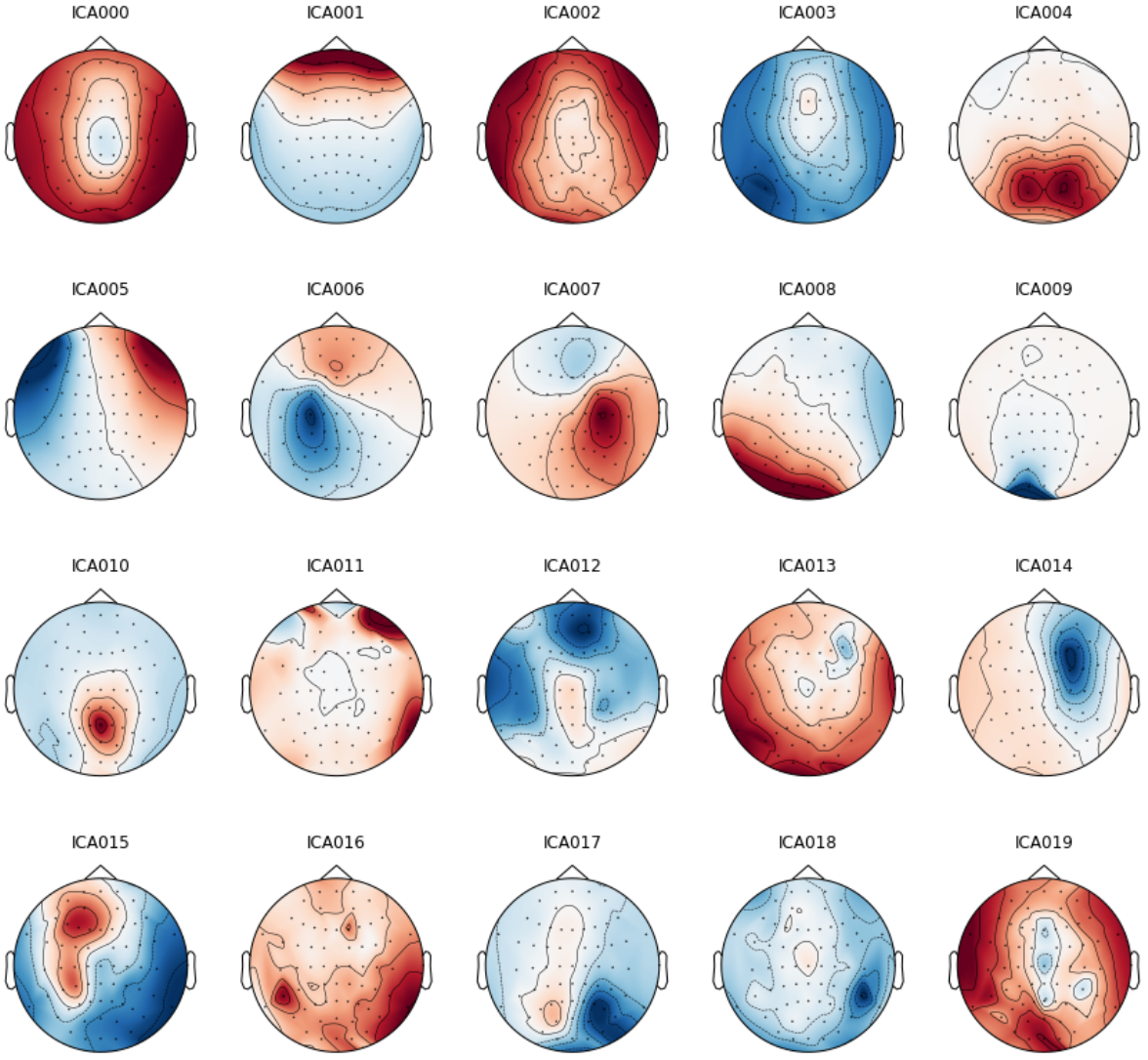
b)

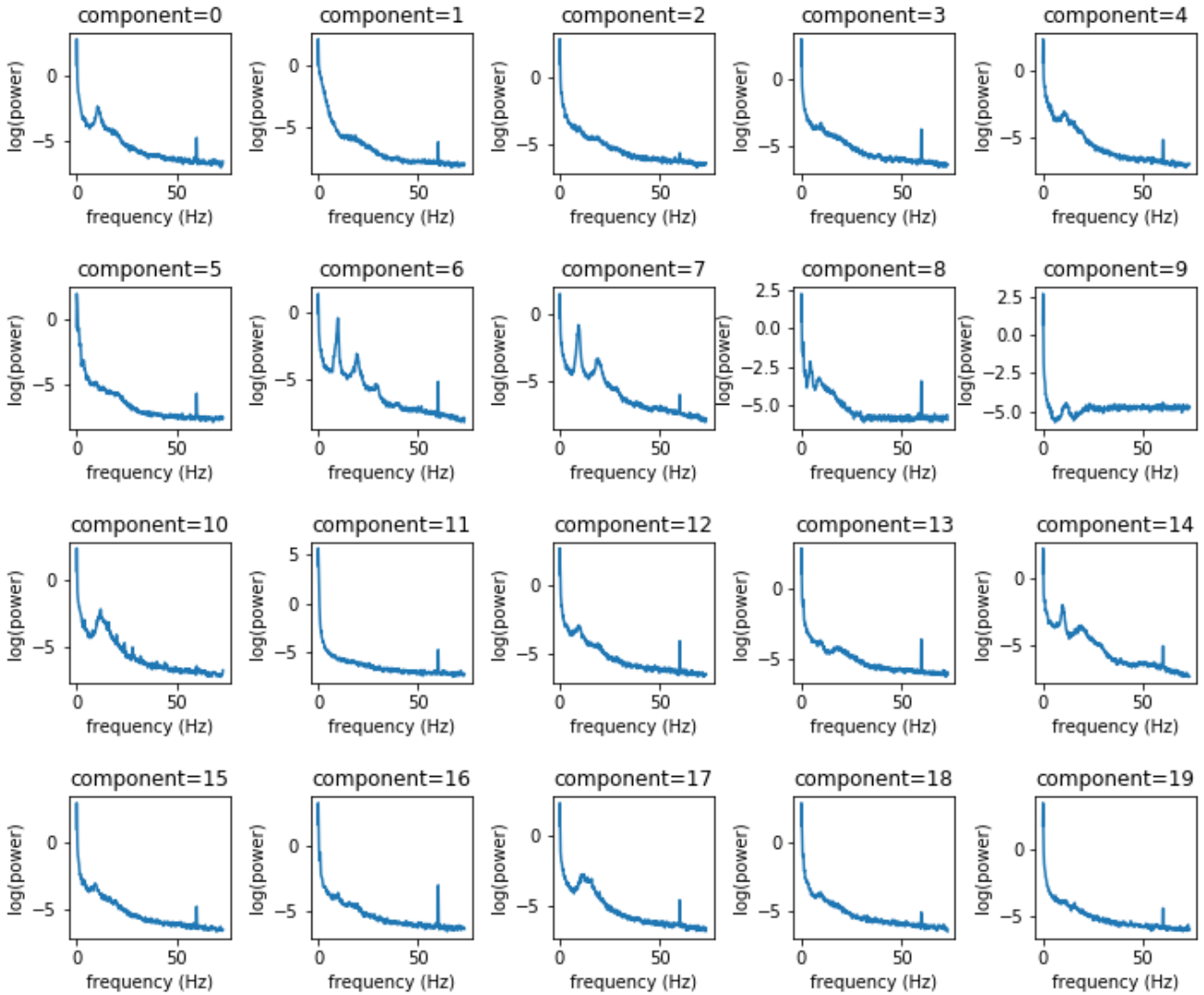






ICA weight maps + power spectrum for a dataset

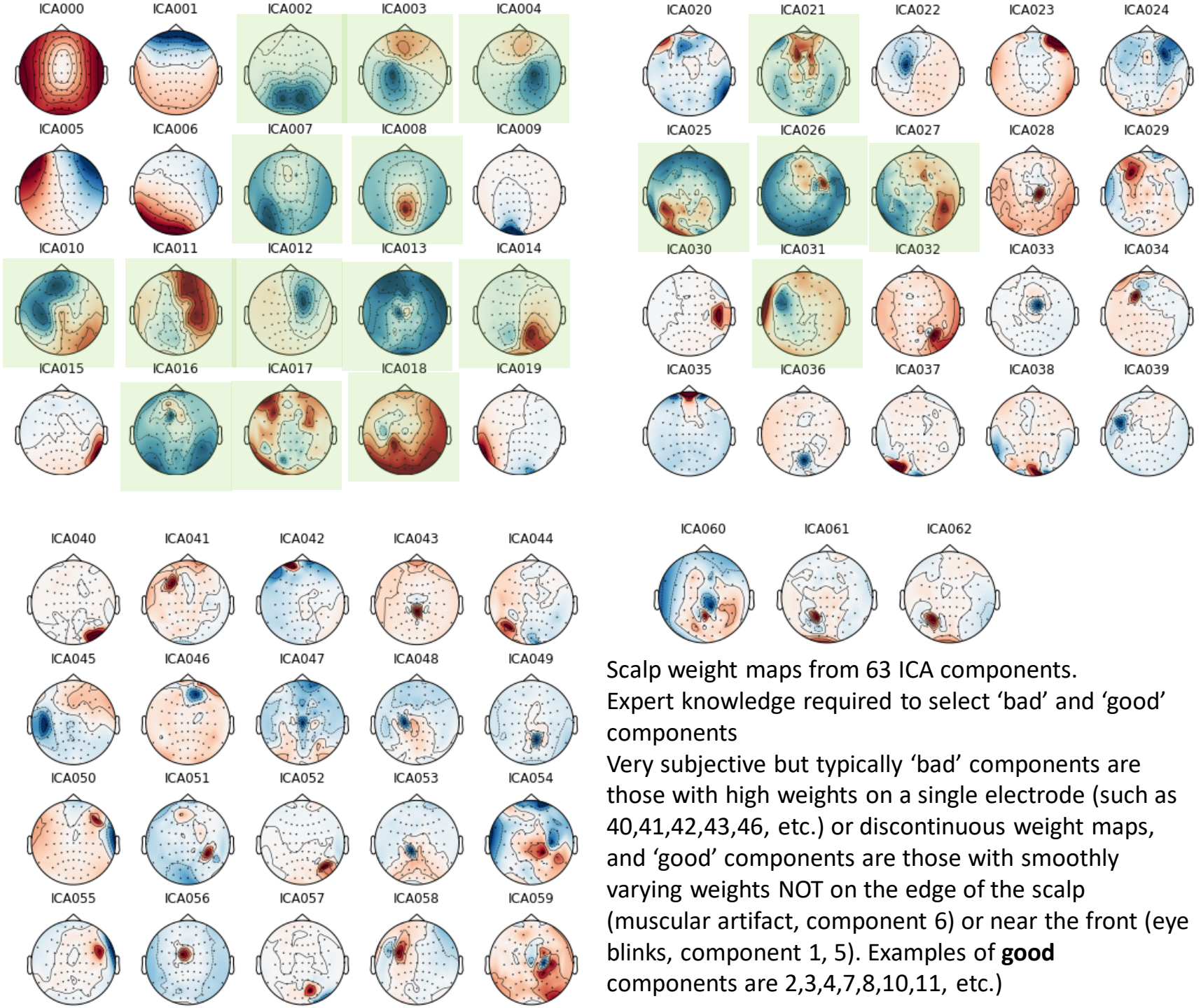




Left = scalp weight maps (shows where component is strongest), **Right** = power spectrum of associated component time series (only the first 20 components of the total 64 are shown here)

ICA denoising

- Can remove noise from EEG by excluding certain components when re-mixing the data:
  - *Step 1*: perform ICA and display all components (scalp weight map + power spectrum, only scalp weight map shown here.
  - *Step 2*: select ‘good’ components using some criteria (expert knowledge or otherwise)
  - *Step 3*: re-mix the data from the source components, while excluding all bad components.
    - Re-mixed data should now contain much less noise and is ready for further processing



Scalp weight maps from 63 ICA components.
Expert knowledge required to select ‘bad’ and ‘good’ components
Very subjective but typically ‘bad’ components are those with high weights on a single electrode (such as 40,41,42,43,46, etc.) or discontinuous weight maps, and ‘good’ components are those with smoothly varying weights NOT on the edge of the scalp (muscular artifact, component 6) or near the front (eye blinks, component 1, 5). Examples of **good** components are 2,3,4,7,8,10,11, etc.)

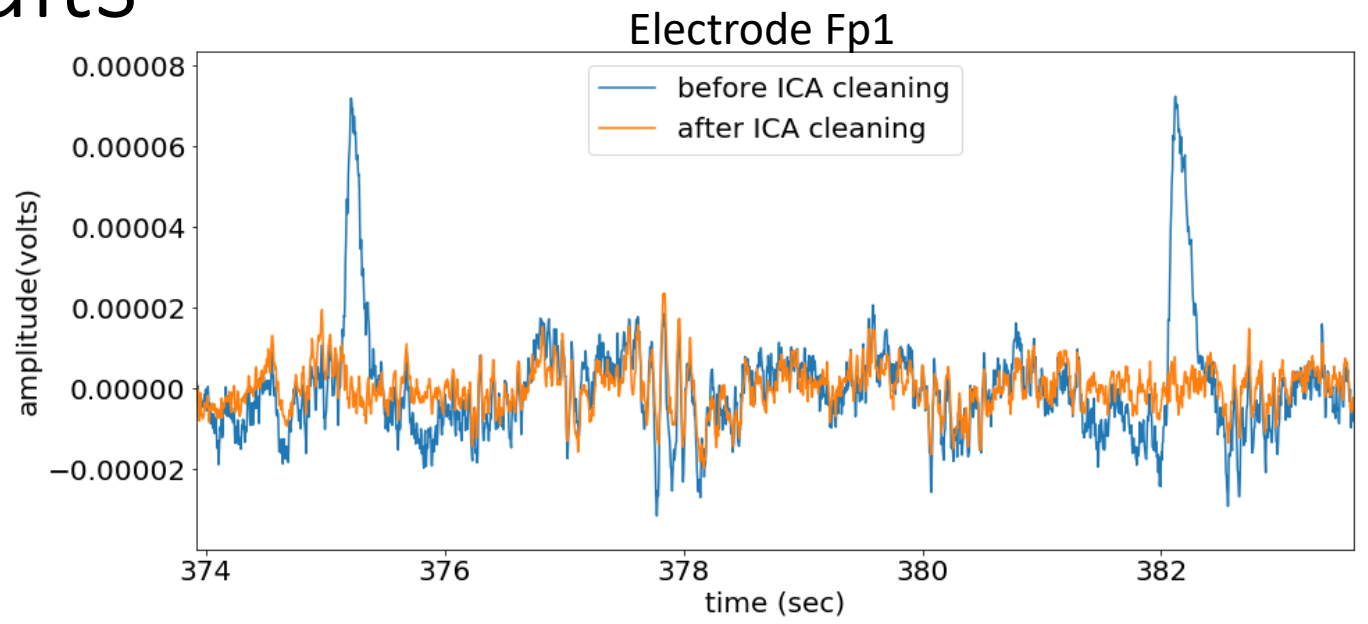
Denoising pipeline + results

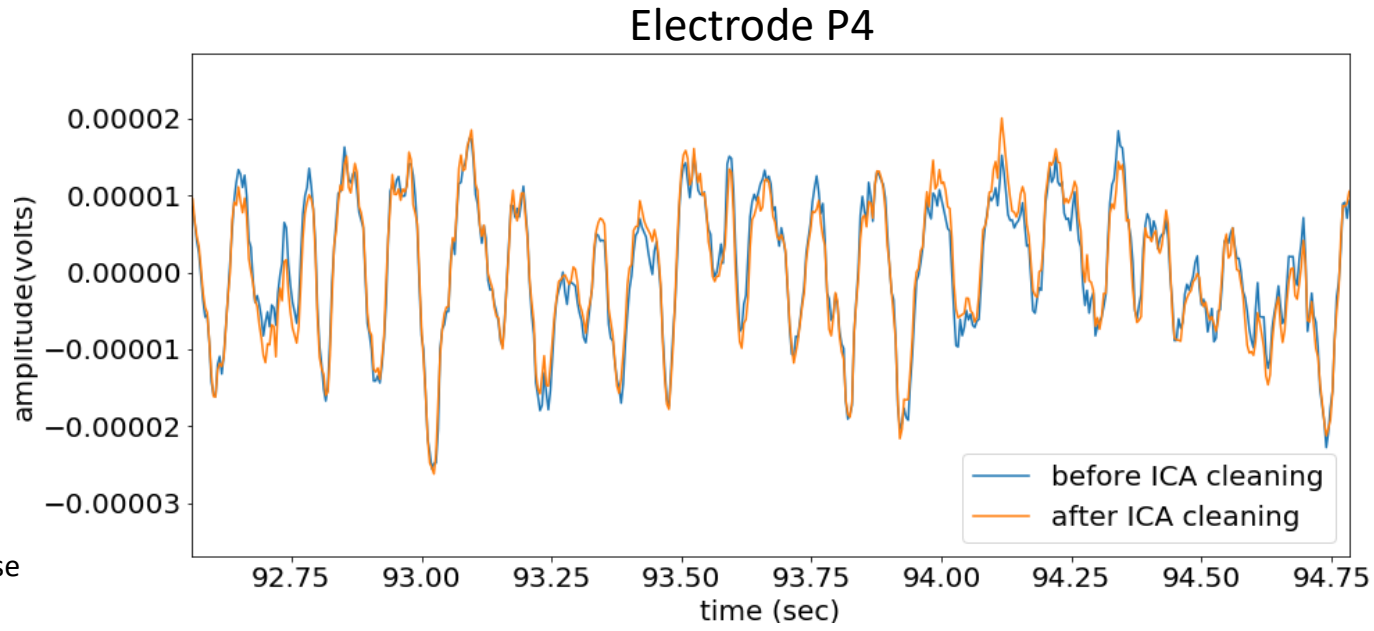
Simple EEG ICA denoising pipeline using the **mne** software package in python:

```
import mne as mne
import matplotlib.pyplot as plt
import numpy as np
plt.rcParams.update({'font.size': 20})

# load two copies of raw data and resample both
# need two copies because we will high-pass filter one before running ICA
highpass_raw = mne.io.read_raw_brainvision('C:/shared/eegdata/outside.vhdr',preload=True)
highpass_raw.resample(250)
raw = mne.io.read_raw_brainvision('C:/shared/eegdata/outside.vhdr',preload=True)
raw.resample(250)

highpass_raw.info['bads'] = ['ECG'] # remove ECG channel (not needed in this example)
highpass_raw.set_montage('standard_1020',raise_if_subset=False) # set electrod locations

highpass_raw.filter(1,124) # apply high pass filter to raw
ica = mne.preprocessing.ICA() # initialize ICA.
ica.fit(highpass_raw) # fit ICA to highpass filtered data
comps = ica.get_sources(raw).get_data() # get the components (not used in this example)

ica.plot_components() # plot weight maps
includes = [0,2,3,4,7,8,10,11,12,13,14,16,18,22,24,25,26,27] # found by visual inspection
ica.apply(raw,include=includes) # now, raw contains full-bandwidth, denoised EEG data
```

```
    ...: ica.apply(raw,include=includes)
Transforming to ICA space (63 components)
Zeroing out 45 ICA components
Out[11]: <RawBrainVision  |  outside.eeg, n_channels x n_times :
64 x 124655 (498.6 sec), ~61.0 MB, data loaded>
```

ICA cleaning affects the signal in some electrodes more than others. Example: Fp1 (electrode near eyes) looks quite different after ICA cleaning because there was a lot of eyeblink contamination in the dataset, however electrode P4 is largely unchanged because it was already quite clean.

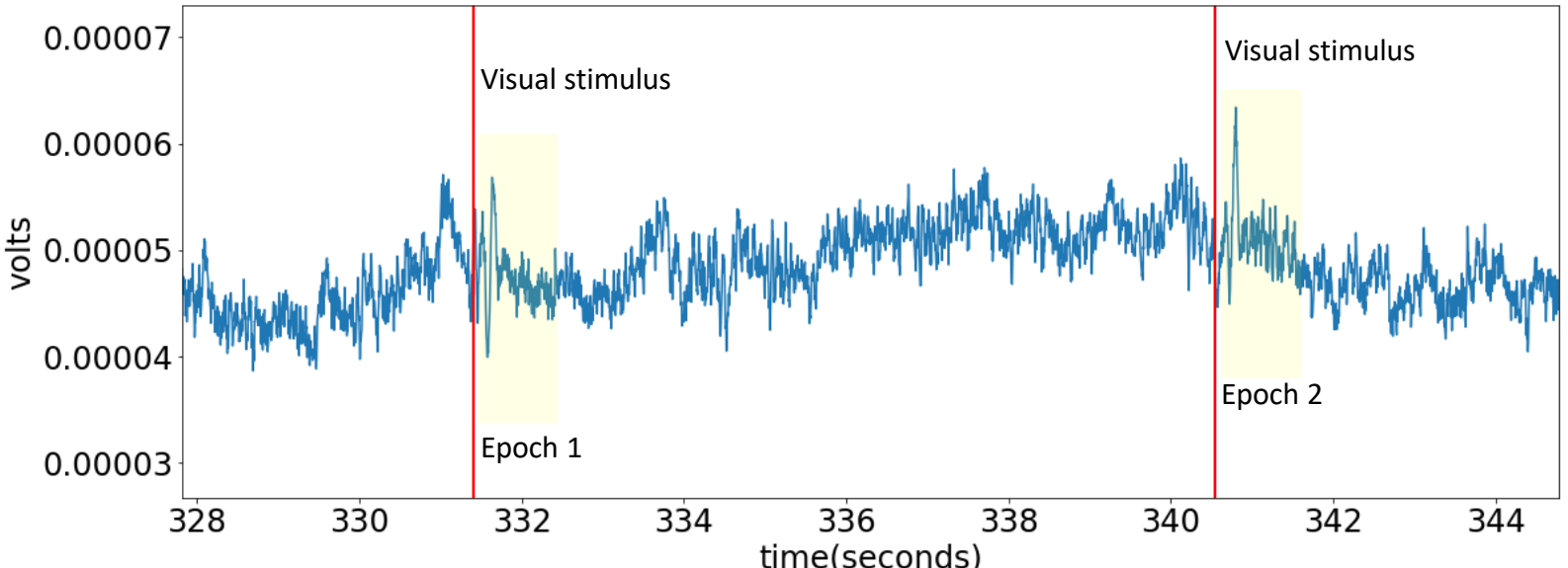




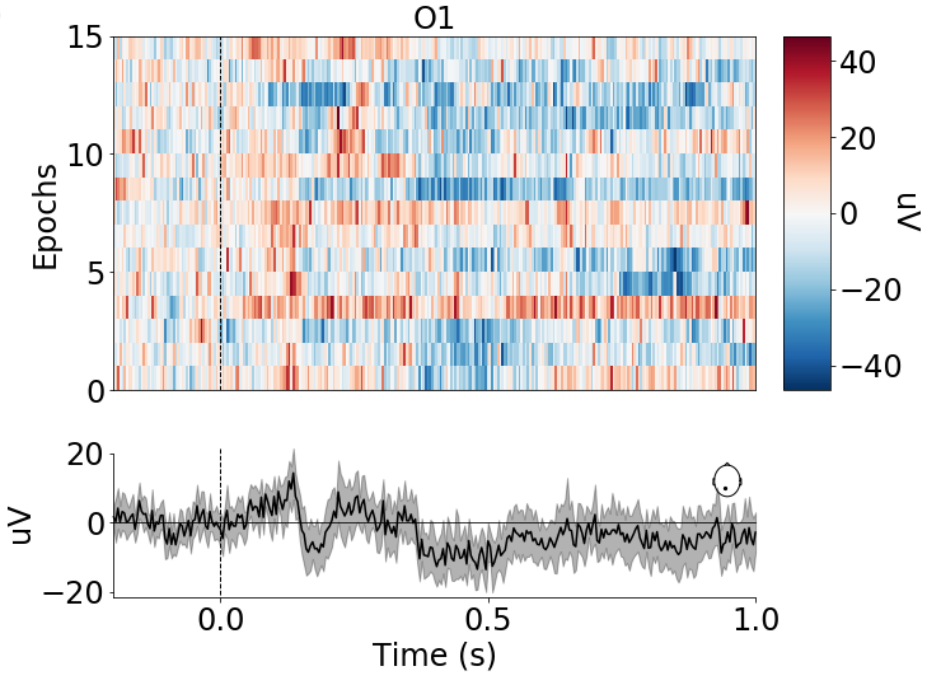
Event-related EEG analysis

- Signal-to-noise ratio of EEG signal is typically quite low
- In most experiments, many *trials* of the same stimulus are performed, and the individual trials are *averaged together* to yield an average response
  - When segmented in time, these trials are called *epochs*
  - Average response has much higher SNR than individual trials
  - a) raw time series showing two trials of visual experiment
  - b) 15 single trials + avg from electrode ‘O1’ (before ICA denoising)
  - c) 15 single trials + avg from electrode ‘O1’ (after ICA denoising)

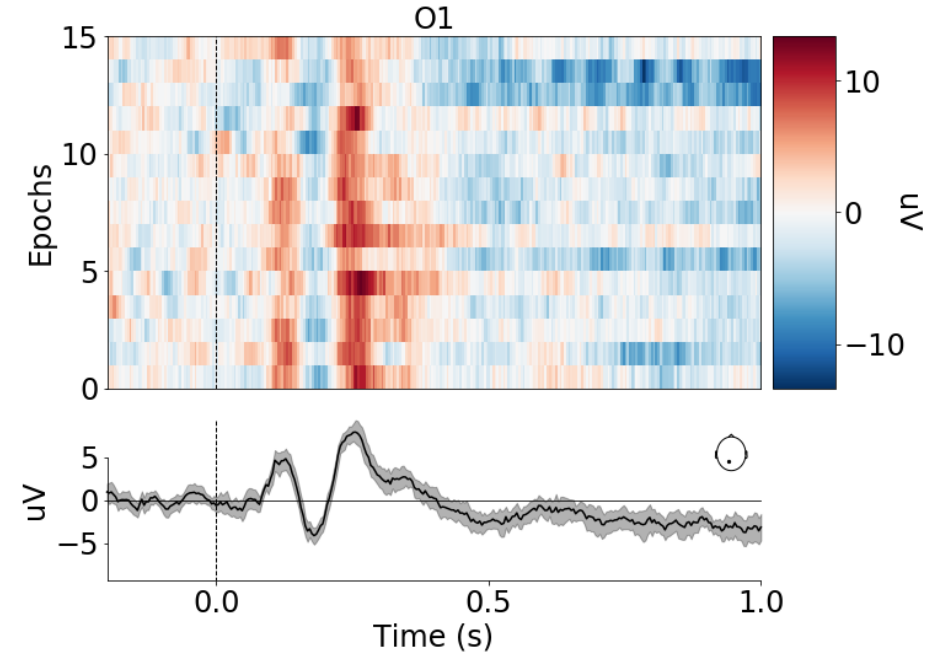
a)



b)



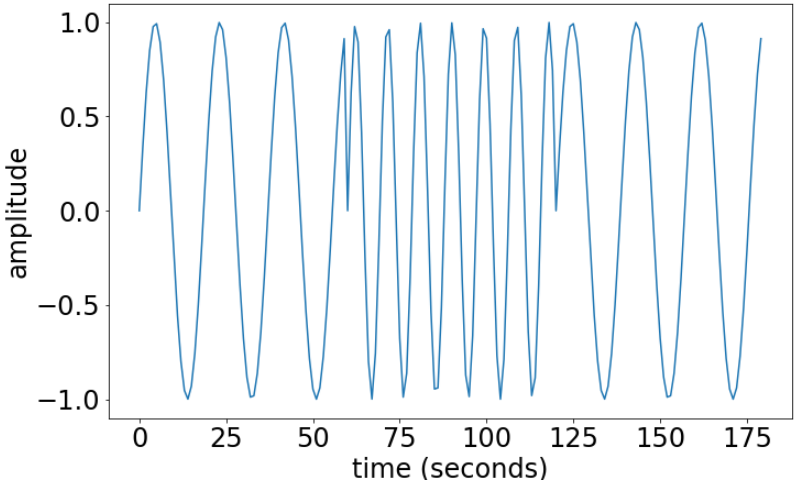
c)



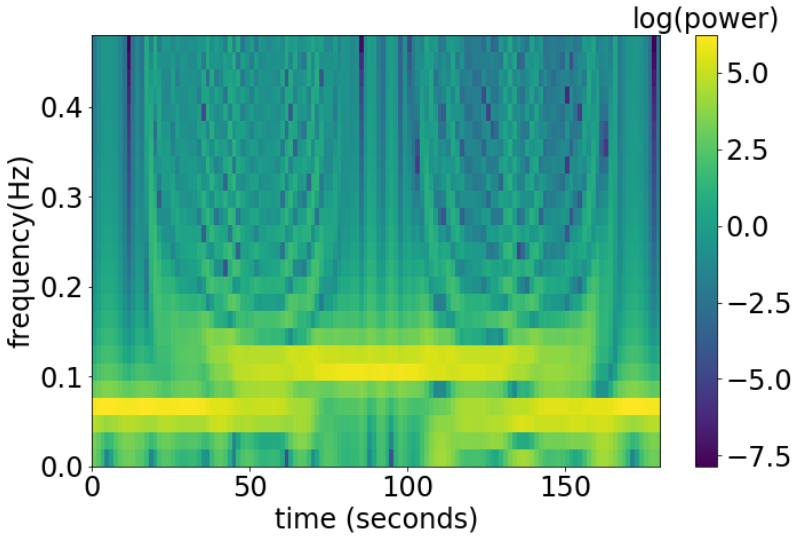
Time-frequency decomposition

- Time frequency analysis refers to techniques that study a signal in both time and frequency domains simultaneously
- Rather than viewing a 1-dimensional function (time series), time frequency analysis studies a two-dimensional function of both time and frequency
- Involves passing a window over the entire signal in discrete steps, computing FFT in the window, and placing result as a column in the time-frequency decomposition
- Time-frequency decomposition simple example:
  - Window size given by `wsize`, step size by `step`

```
import numpy as np
import matplotlib.pyplot as plt
from numpy.fft import fft, ifft, fftshift, fftfreq

signal = np.array([np.sin(np.linspace(0,20,60)),\
                   np.sin(np.linspace(0,40,60)),
                   np.sin(np.linspace(0,20,60))])
signal = signal.flatten()
freqs = fftfreq(50,1)
wsize = 50
step = 1
time_freq = np.zeros([wsize//2,(signal.shape[0]-wsize)//step])
for i in np.arange(0,signal.shape[0]-wsize):
    fft_i = fft(signal[i:i+wsize])
    time_freq[:,i] = np.abs(fft_i[0:wsize//2])


plt.subplot(1,2,1)
plt.plot(signal); plt.xlabel('time (seconds)');plt.ylabel('amplitude')

plt.subplot(1,2,2);
plt.imshow(np.log(time_freq**2),origin='lower',aspect='auto',\
           extent=[0,180,freqs[0],freqs[24]])
plt.xlabel('time (seconds)'); plt.ylabel('frequency(Hz)')

cb = plt.colorbar() ; cb.ax.set_title('log(power)')
```



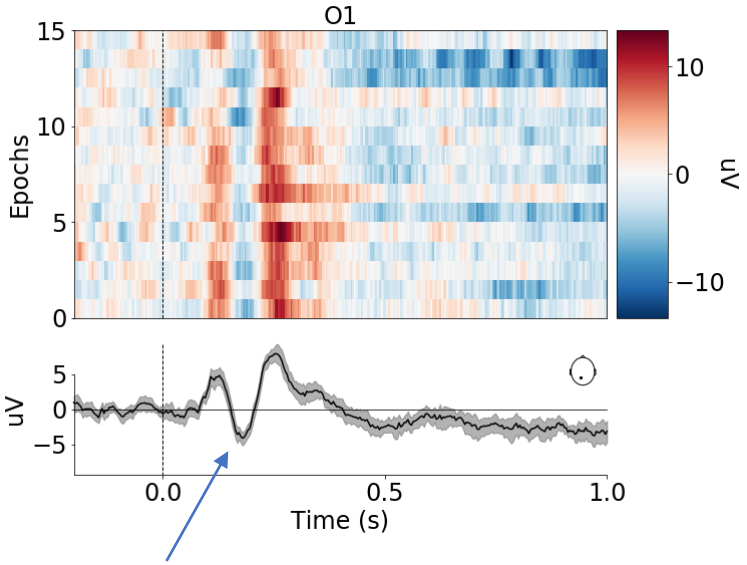
Time-frequency decomposition



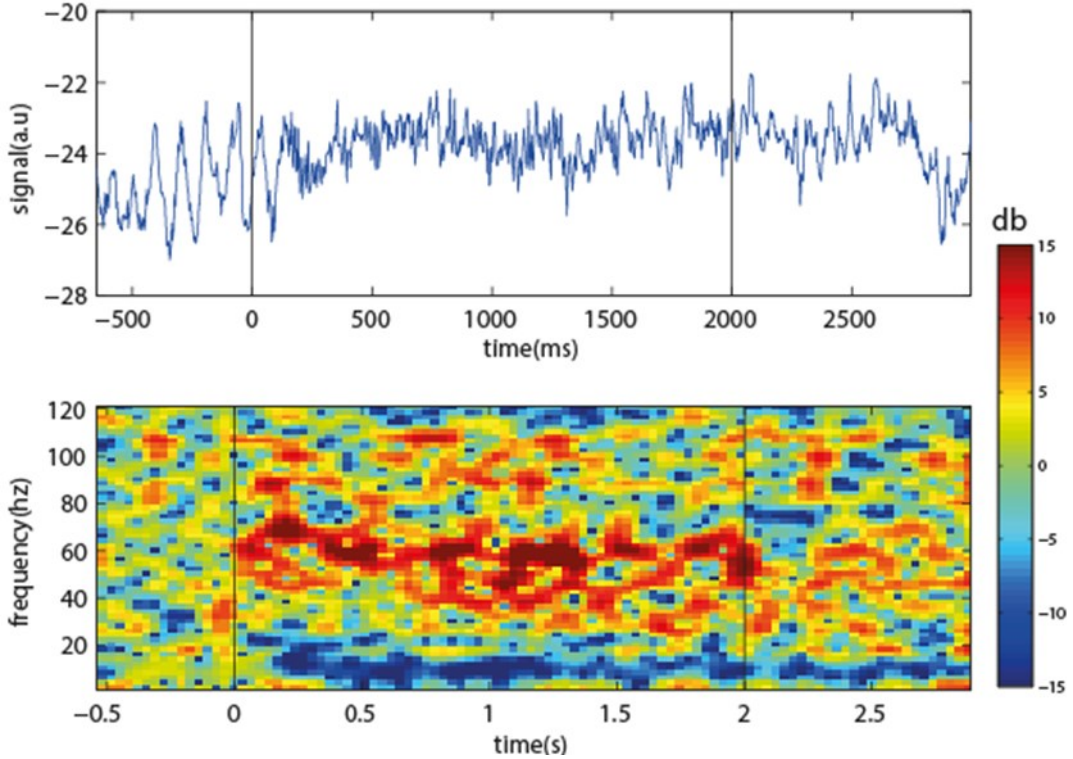
~4 oscillations in first 60 seconds of time series, corresponding to yellow line at ~0.07 Hz in time frequency decomposition. Between 60-120 seconds, 7 oscillations, corresponding to yellow line at ~0.12 Hz

Time-frequency decomposition applied to EEG data

- Time-frequency decomposition allows to capture high-frequency oscillations in human EEG data
- Averaging in time domain will not work. Why? Small fluctuations across multiple trials cancel each other out, smoothing signal and hence removing higher frequencies
- Instead, perform time-frequency decomposition first, *then* take average

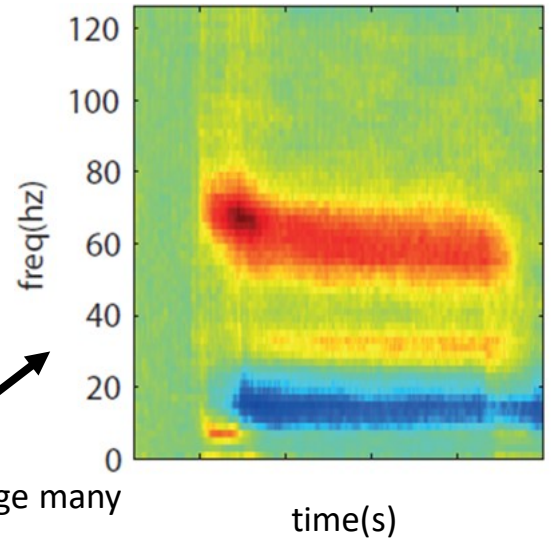


Smoothed due to averaging



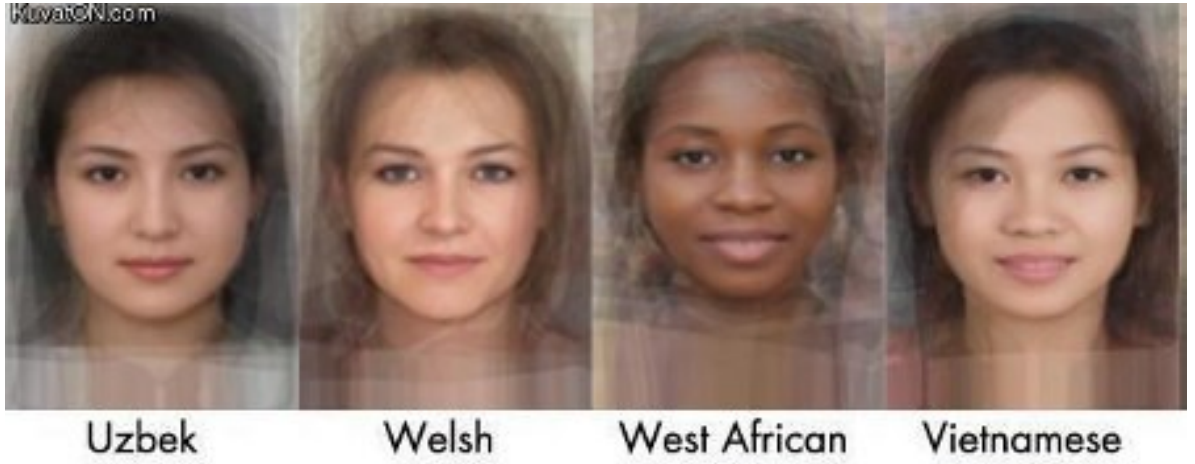
Average many trials

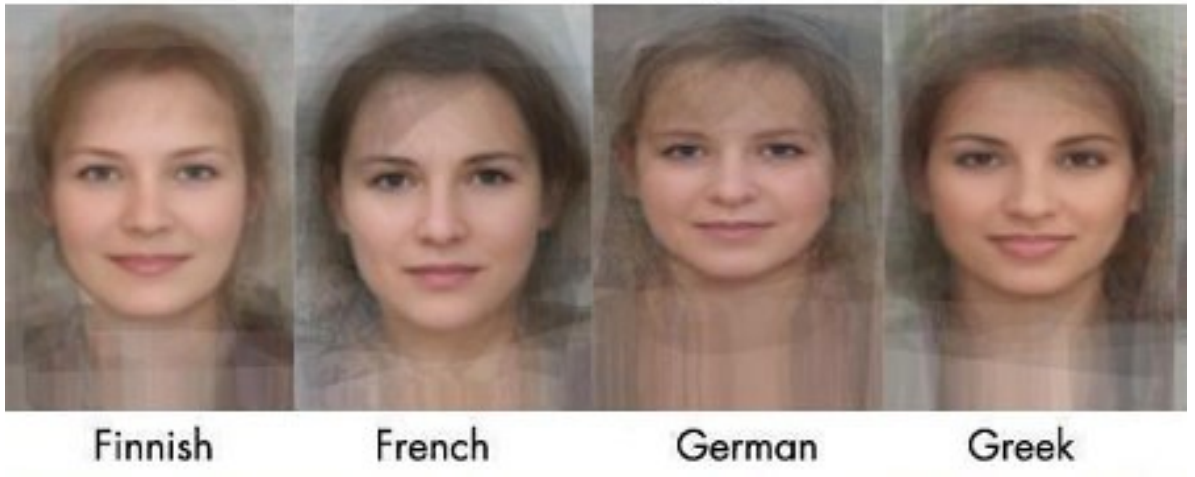
averaged TF decomposition



A final word on ‘averaging’

- Why are all ‘average’ faces attractive?
- Averaging reduces noise and increases signal-to noise ratio
- However, averaging can also smooth and reduce high-frequencies (especially if the images/signals are not properly aligned)
- *‘Why are average faces attractive? The effect of view and averageness on the attractiveness of female faces’*
  - https://link.springer.com/article/10.3758/BF03196599





fastICA applied to BOLD fMRI

Nilearn:
Machine learning for Neuro-Imaging in Python

- Applying ICA to BOLD similar to EEG, except we refer to it as *spatial* ICA rather than *temporal* ICA (as for EEG)
- ICA requires a large number of samples relative to the number of channels.
  - Example: in EEG we typically have 64 channels, and >100,000 time points
- In BOLD fMRI, however, we have typically only 500-1000 time points, and >100,000 voxels (voxels in fMRI are analogous to electrodes in EEG)
- Therefore, we treat each time point in BOLD as a channel, and voxels as time points (this is called spatial ICA)
- For each component, spatial ICA outputs a set of voxel weights (a)