Create an algorithm for Blind Signal Separation (BSS) of 2 and 3 soundwaves.

This task need to do a Blind Signal Separation (BSS) of soundwaves.

Source separation, blind signal separation (BSS) or blind source separation, is the separation of a set of source signals from a set of mixed signals, without the aid of information (or with very little information) about the source signals or the mixing process.
It is most commonly applied in digital signal processing and involves the analysis of mixtures of signals; the objective is to recover the original component signals from a mixture signal. The classical example of a source separation problem is the cocktail party problem, where a number of people are talking simultaneously in a room (for example, at a cocktail party), and a listener is trying to follow one of the discussions. The human brain can handle this sort of auditory source separation problem, but it is a difficult problem in digital signal processing.

And then, i know that Independent component analysis (ICA) is a special case of blind source separation method.

Independent component analysis (ICA) is a statistical and computational technique for revealing hidden factors that underlie sets of random variables, measurements, or signals.

ICA defines a generative model for the observed multivariate data, which is typically given as a large database of samples. In the model, the data variables are assumed to be linear mixtures of some unknown latent variables, and the mixing system is also unknown. The latent variables are assumed nongaussian and mutually independent, and they are called the independent components of the observed data. These independent components, also called sources or factors, can be found by ICA. One of the famous problem “Cocktail Party Problem” — Listening particular One person’s voice in a noisy room, is a common example which is known as an application of ICA algorithm.

So first i look the [6], it's a implementation of a linear mixtures of signals into their underlying independent components.

And then, I have a look at [4]. It use the FastICA to estimating sources from noisy data. This code use the library from sklearn.

FastICA is an efficient and popular algorithm for independent component analysis invented by Aapo Hyvärinen at Helsinki University of Technology.Like most ICA algorithms, FastICA seeks an orthogonal rotation of prewhitened data, through a fixed-point iteration scheme, that maximizes a measure of non-

Gaussianity of the rotated components.
Non-gaussianity serves as a proxy for statistical independence, which is a very strong condition and requires infinite data to verify. FastICA can also be alternatively derived as an approximative Newton iteration.

Algorithm

The generative model of ICA

The ICA is based on a generative model. This means that it assumes an underlying process that generates the observed data. The ICA model is simple, it assumes that some independent source signals s are linear combined by a mixing matrix A.

$$x = As$$

Retrieving the components

The above equations implies that if we invert A and multiply it with the observed signals x we will retrieve our sources:

$$W = A^{-1}$$
$$s = xW$$

This means that what our ICA algorithm needs to estimate is W.

Preprocessing functions

To get an optimal estimate of the independent components it is advisable to do some pre-processing of the data. In the following the two most important pre-processing techniques are explained in more detail.

The first pre-processing step is centering.
This is a simple subtraction of the mean from our input X. As a result the centered mixed signals will have zero mean which implies that also our source signals s are of zero mean. This simplifies the ICA calculation and the mean can later be added back.

```
def center(x):
    mean = np.mean(x, axis=1, keepdims=True)
    centered =  x - mean
    return centered, mean
```

For the second pre-processing technique we need to calculate the covariance.

```
def covariance(x):
    mean = np.mean(x, axis=1, keepdims=True)
    n = np.shape(x)[1] - 1
    m = x - mean

    return (m.dot(m.T))/n
```

The second pre-processing method is called whitening.
The goal here is to linearly transform the observed signals X in a way that potential correlations between the signals are removed and their variances equal unity. As a result the covariance matrix of the whitened signals will be equal to the identity matrix

```
def whiten(x):
    # Calculate the covariance matrix
    coVarM = covariance(X)

    # Single value decoposition
    U, S, V = np.linalg.svd(coVarM)

    # Calculate diagonal matrix of eigenvalues
    d = np.diag(1.0 / np.sqrt(S))

    # Calculate whitening matrix
    whiteM = np.dot(U, np.dot(d, U.T))

    # Project onto whitening matrix
    Xw = np.dot(whiteM, X)

    return Xw, whiteM
```

Implement the fast ICA algorithm

And then take a look at the actual ICA algorithm.
As discussed above one precondition for the ICA algorithm to work is that the source signals are non-Gaussian. Therefore the result of the ICA should return sources that are as non-Gaussian as possible. To achieve this we need a measure of Gaussianity. One way is Kurtosis and it could be used here but another way has proven more efficient.
Nevertheless we will have a look at kurtosis at the end of this notebook. For the actual algorithm however we will use the equations g and g' which are derivatives of $f(u)$ as defined below.

$$f(u) = \log\cosh(u)$$
$$g(u) = \tanh(u)$$

$$g'(u) = 1 - \tanh^2(u)$$

These equations allow an approximation of negentropy and will be used in the below ICA algorithm which is based on a fixed-point iteration scheme:

$$for 1 to number of components c:$$
$$w_p = random initialisation$$
$$whiile w_p not < threshold:$$
$$w_p = \frac{1}{n}(X_g(W^T X) - g'(W^T X)W)$$
$$w_p = w_p - \sum_{j=1}^{p-1}(w_p^t w_j)w_j$$
$$w_p = w_p / ||w_p|| W = [W_1, ..., W_c]$$

So according to the above what we have to do is to take a random guess for the weights of each component. The dot product of the random weights and the mixed signals is passed into the two functions g and g'. We then subtract the result of g' from g and calculate the mean. The result is our new weights vector. Next we could directly divide the new weights vector by its norm and repeat the above until the weights do not change anymore. There would be nothing wrong with that. However the problem we are facing here is that in the iteration for the second component we might identify the same component as in the first iteration. To solve this problem we have to decorrelate the new weights from the previously identified weights. This is what is happening in the step between updating the weights and dividing by their norm.

Reference

1. Signal separation(wikipad)
2. What is Independent Component Analysis?
3. Independent Component Analysis for Signal decomposition
4. Blind source separation using FastICA
5. FastICA
6. Independent Component Analysis (ICA) implementation from scratch in Python