

SEPARATING NONLINEAR IMAGE MIXTURES USING A PHYSICAL MODEL TRAINED WITH ICA

Mariana S.C. Almeida

Luís B. Almeida

**Instituto de Telecomunicações IST
Lisbon, Portugal**

Partially funded by FCT (Portugal)





Mariana S.C. Almeida

Luís B. Almeida

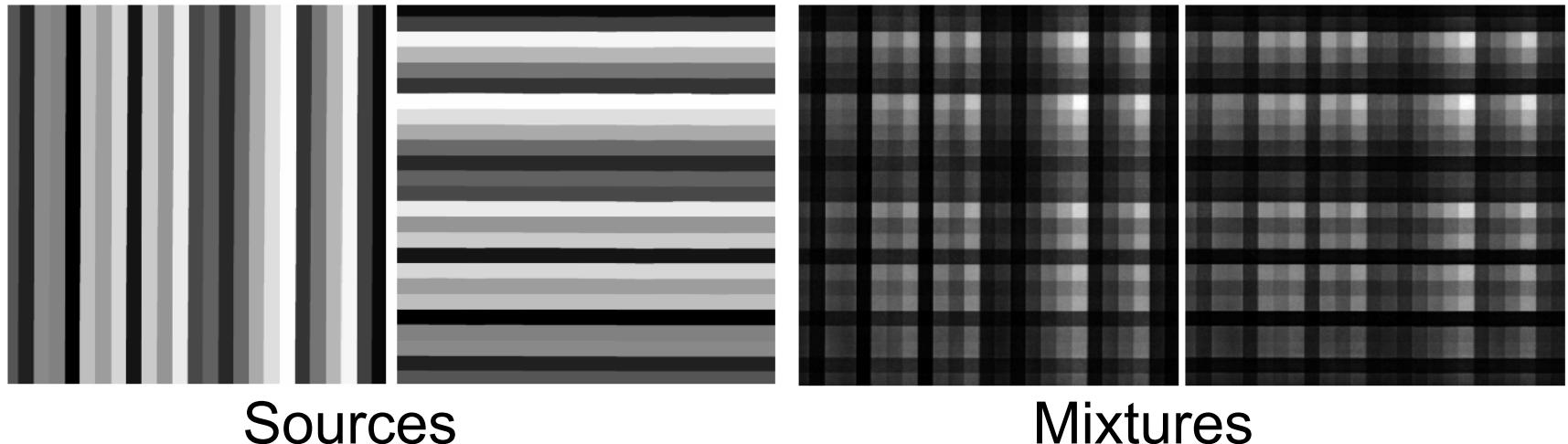
Instituto de Telecomunicações IST
Lisbon, Portugal



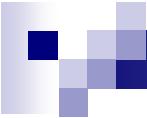
Outline

- Nonlinear mixture of images
- MISEP ICA method (brief review)
- Physical model of the mixing process
- Inverse model for separation
- Experiments and results
- Conclusions

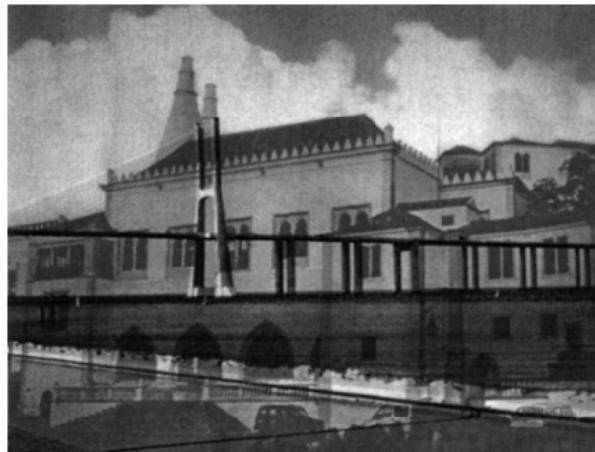
Mixing problem



- Mixture of the front- and back-page images of a document when acquired with a scanner
- The mixture is nonlinear and noisy
- Five different pairs of mixtures were studied



Mixing problem



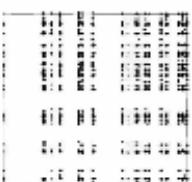
Mixing problem

Separation of nonlinear image mixture

When acquiring an image of a printed document, the image printed on the opposite page often shows through, due to partial transparency of the paper. Here we are dealing with a rather simple case of the effect, because the image on the other page is a uniform gray.

We observe that as occurs in other conditions, as can be observed from the scatter plot on the right, which shows a scatter plot of the intensity of corresponding pairs of points from the two pages of a printed document. The scatter plot of the original images, shown in the lower figure (left side), has corners, and had only a relatively small number of discrete intensity levels (see note 1). The fact that the scatter plot of Fig. 1 is very different, from a parallelogram shape (Fig. 1), the measure was severely disturbed. The fact that the scatter plot becomes more linear in the experimental case (left corner disappears) in the right triangles in both images indicates, yet there are errors in the original document, especially due to the fact that the discrete levels of Fig. 2 become largely blurred at Fig. 1 due to noise in the process. The process leading from the source to the observations involved printing the images on both sides of a sheet of onion skin paper at 1200 dpi, with a black and white laser printer (with the same inherent resolution of 600 dpi), and then scanning both sides of the printed sheet at 100 dpi. The noise is due, at least, to the printing process (including the halftoning in the scanning process) and to the nonuniformity in the onion skin paper, especially in its transparency.

In this example we are dealing with a mixture that results in a linear image, the spectral characteristics of which are such that they actually invert it (i.e., intensity levels black and white through the image). In this case, their will appear in the scanned image, as two clusters of



Separation of nonlinear image mixture

When acquiring an image of a printed document, the image printed on the opposite page often shows through, due to partial transparency of the paper. Here we are dealing with a rather simple case of the effect, because

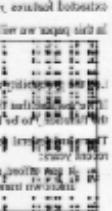
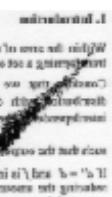
the image on the other page is a uniform gray. We are dealing with the top figure on the right (top-right image) and bottom-left image (bottom-left image) of the original images, shown in the lower figure (left side), corresponding pairs of points from the next page of a printed document.

The scatter plot of the original images, shown in the lower figure (left side), has corners, and had only a relatively small number of discrete intensity levels (see note 1). The fact that the scatter plot of Fig. 1 is very different, from a parallelogram shape (Fig. 1), the measure was severely disturbed. The fact that this scatter plot becomes quite narrow in the upper-right corner, which corresponds to the bottom-left corner in both images, and all around this are empty cells, indicates that, for these intensities, the measure is close to regular. Finally,

the fact that the discrete levels of Fig. 2 become largely blurred at Fig. 1 is due to noise in the process. The process leading from the source to the observations involved printing the images on both sides of a sheet of onion skin paper at 1200 dpi, with a black and white laser printer (with the same inherent resolution of 600 dpi), and then scanning both sides of the printed sheet at 100 dpi. The noise is due, at least, to the printing process (including the halftoning in the scanning process) and to the nonuniformity in the onion skin paper, especially in its transparency.

The number of repetitions were reduced, since the mixture contains the majority of gray areas. The image on the opposite page is a uniform gray, obtained by scanning both faces of the printed document, the images that had been printed in each of its faces, with as little difference from the original image as possible, and can be viewed on the (x_1, x_2, x_3) -all axis.

In this example we are dealing with a mixture that results in a linear image, the spectral characteristics of which are such that they actually invert it (i.e., black and white through the image). The spectral characteristics of the image on the other page is a uniform gray. The image on the opposite page is a uniform gray, obtained by scanning both faces of the printed document, the images that had been printed in each of its faces, with as little difference from the original image as possible, and can be viewed on the (x_1, x_2, x_3) -all axis.



Separation of nonlinear image mixture

method 1

To solve the mixing problem, we used the method proposed in reference 10, which is based on the use of a scattering matrix to obtain the intensity distribution in a given point. To do this, we used the following algorithm:

(1) $\text{Intensity} = \frac{1}{3}(I_{11} + I_{22} + I_{33})$

nonlinearity function $\alpha = \text{exp}(a \cdot \text{Intensity})$

and $\alpha = 1/\sqrt{\text{Intensity}}$, and the total nonlinearity value $\beta = \alpha \cdot \text{Intensity}$. After β is calculated, we calculate the total intensity $I_{\text{tot}} = \text{Intensity} + \beta$ and the total nonlinearity $\alpha_{\text{tot}} = \alpha + \beta$.

Introducing the total nonlinearity $\alpha_{\text{tot}} = 1/\sqrt{I_{\text{tot}}}$ into the equation of the scattering matrix

(2) $\begin{pmatrix} I_{11} & I_{12} & I_{13} \\ I_{21} & I_{22} & I_{23} \\ I_{31} & I_{32} & I_{33} \end{pmatrix} = \frac{1}{2} \begin{pmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{pmatrix}$

and applying it to the scattering matrix of the original image, we obtain the total scattering matrix of the mixed image. Its nonlinearity value α_{tot} and the total intensity I_{tot} are used to calculate the total nonlinearity α_{tot} and the total intensity I_{tot} .

Introducing the total nonlinearity $\alpha_{\text{tot}} = 1/\sqrt{I_{\text{tot}}}$ into the equation of the scattering matrix

(3) $\begin{pmatrix} I_{11} & I_{12} & I_{13} \\ I_{21} & I_{22} & I_{23} \\ I_{31} & I_{32} & I_{33} \end{pmatrix} = \frac{1}{2} \begin{pmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{pmatrix}$

and applying it to the scattering matrix of the mixed image, we obtain the total scattering matrix of the original image. Its nonlinearity value α_{tot} and the total intensity I_{tot} are used to calculate the total nonlinearity α_{tot} and the total intensity I_{tot} .

Introducing the total nonlinearity $\alpha_{\text{tot}} = 1/\sqrt{I_{\text{tot}}}$ into the equation of the scattering matrix

(4) $\begin{pmatrix} I_{11} & I_{12} & I_{13} \\ I_{21} & I_{22} & I_{23} \\ I_{31} & I_{32} & I_{33} \end{pmatrix} = \frac{1}{2} \begin{pmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{pmatrix}$

and applying it to the scattering matrix of the original image, we obtain the total scattering matrix of the original image.

method 2

When acquiring an image of a printed document, the image printed on the opposite page often shows through, due to partial transparency of the paper. Here we are dealing with a rather simple case of the effect, because

the image on the other page is a uniform gray. We are dealing with the top figure on the right (top-right image) and bottom-left image (bottom-left image) of the original images, shown in the lower figure (left side), corresponding pairs of points from the next page of a printed document.

The scatter plot of the original images, shown in the lower figure (left side), has corners, and had only a relatively small number of discrete intensity levels (see note 1). The fact that the scatter plot of Fig. 1 is very different, from a parallelogram shape (Fig. 1), the measure was severely disturbed. The fact that this scatter plot becomes quite narrow in the upper-right corner, which corresponds to the bottom-left corner in both images, and all around this are empty cells, indicates that, for these intensities, the measure is close to regular. Finally,

the fact that the discrete levels of Fig. 2 become largely blurred at Fig. 1 is due to noise in the process. The process leading from the source to the

observations involved printing the images on both sides of a sheet of onion skin paper at 1200 dpi, with a black and white laser printer (with the same inherent resolution of 600 dpi), and then scanning both sides of the printed sheet at 100 dpi. The noise is due, at least, to the printing process

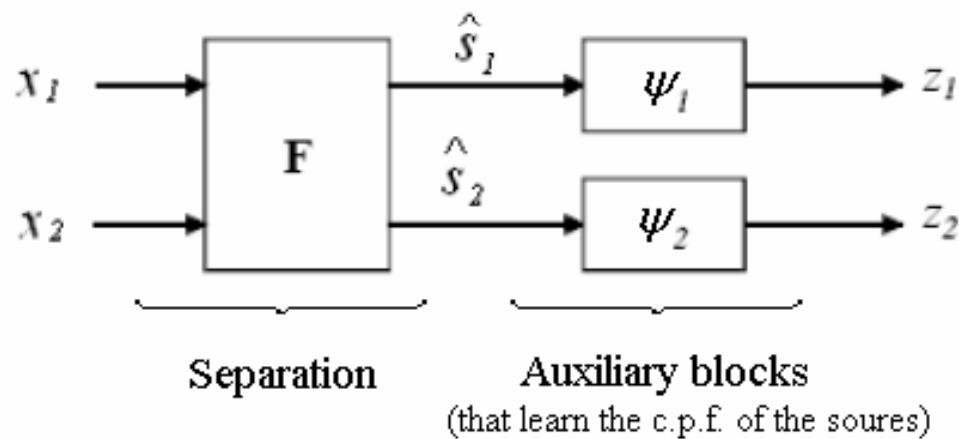
(including the halftoning in the scanning process) and to the nonuniformity in the onion skin paper, especially in its transparency.

The number of repetitions were reduced, since the mixture contains the majority of gray areas. The image on the opposite page is a uniform gray, obtained by scanning both faces of the printed document, the images that had been printed in each of its faces, with as little difference from the original image as possible, and can be viewed on the (x_1, x_2, x_3) -all axis.

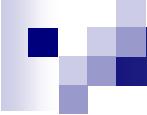
In this example we are dealing with a mixture that results in a linear image, the spectral characteristics of which are such that they actually invert it (i.e., black and white through the image). The spectral characteristics of the image on the other page is a uniform gray. The image on the opposite page is a uniform gray, obtained by scanning both faces of the printed document, the images that had been printed in each of its faces, with as little difference from the original image as possible, and can be viewed on the (x_1, x_2, x_3) -all axis.

MISEP method

- Performs nonlinear ICA by minimizing mutual information



- Generalizes *Infomax* in two directions
 - Allows nonlinear functions in the separation block \mathbf{F}
 - Adaptive output nonlinearities



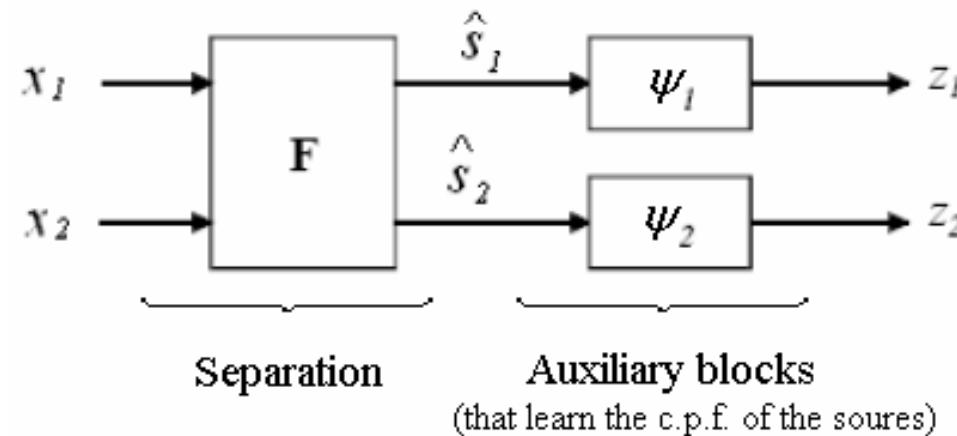
Mixing model

- Based on the *halftoning* process of the printers
 - The printer produces small black dots
 - The gray level is given by the fraction of area covered by the dots
 - We model the dots by a random binary variable
 - With suitable assumptions, a bilinear mixing model can be derived

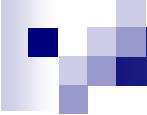
$$\begin{cases} x_1 = \alpha s_1 + \beta s_2 + \gamma s_1 s_2 + \delta \\ x_2 = \alpha s_2 + \beta s_1 + \gamma s_2 s_1 + \delta \end{cases}$$

s_i - sources x_i - mixtures

Inverse (separation) model

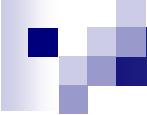


- \mathbf{F} implements the inverse of the physical mixture model
- This inverse can be found algebraically
- Equations not shown here due to complexity
- This inverse has the same four parameters as the physical mixture model
- These four parameters are what needs to be estimated by the MISEP method

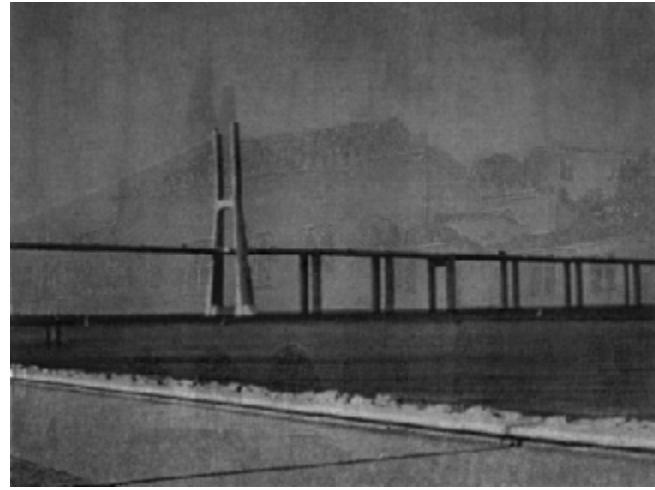
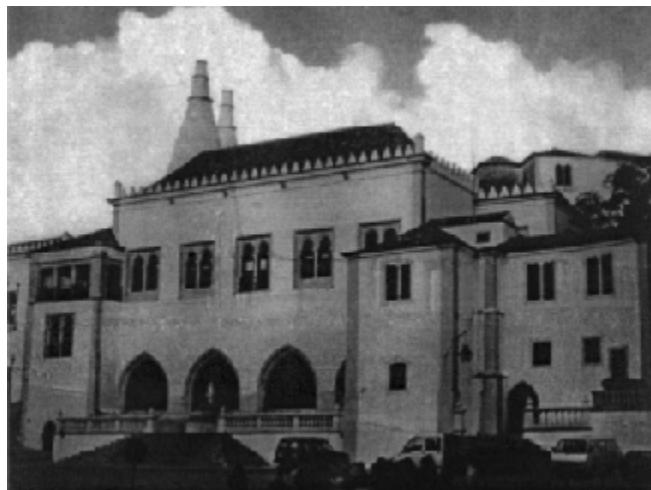
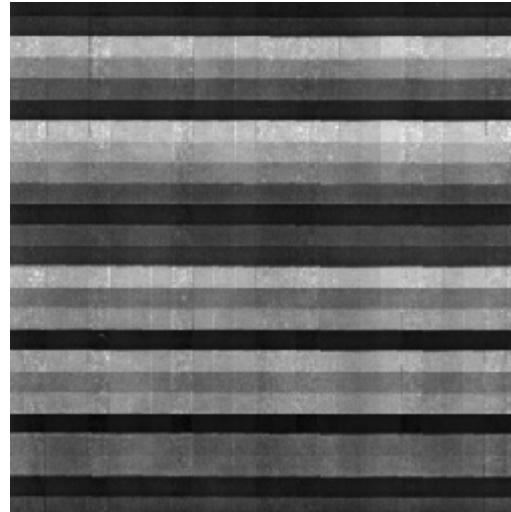
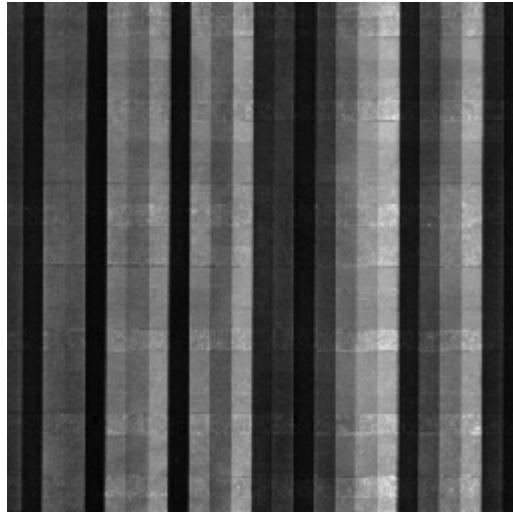


Experiments

- The **F** Block was initialized near the identity function
- Auxiliary blocks were MLPs with 10 hidden units each
- MISEP was applied to each pair of mixtures during 1000 epochs.
 - A separation model was trained for each pair of mixtures
 - Training set: 1000 pairs of pixels, randomly selected



Results (1)



Results (2)

Separation of nonlinear image mixtures

When acquiring an image of a printed document, the image printed on the opposite page often shows through, due to partial transparency of the paper, like it is floating with just a strong out-of-focus effect, because we are reading from the paper with a low-light exposure.

The mixture that is obtained is called nonlinear, as can be observed from the top figure on the right, which shows a scatter plot of the intensities of corresponding pairs of points from the two pages of a printed document. The scatter plot of the original images, shown in the bottom figure, follows a square, and had only a relatively small number of discrete intensity levels for each image. The fact that the shape of the scatter plot of Fig. 1 is very different from a rectangular shows that the mixture was actually nonlinear. The fact that the scatter plot becomes gray in the upper-right corner (which corresponds to the lighter intensities in both images) indicates that, for those intensities, the mixture is close to singular. Finally, the fact that the discrete levels of Fig. 2 become singular in Fig. 1 is due to noise in the process. The process leading from the source to the observation involved printing the images, as both sides of a sheet of paper at 1200 dpi, with a black and white laser printer (with the inherent resolution of gray levels), and then scanning both sides of the printed sheet at 100 dpi. The noise is due, at least, to the printing process (overlaping, ink bleeding) in the scanning process and to the non-uniformity in the scanner's skin paper, especially in its nonlinearity.

The purpose of separation is to recover, from the mixed images that are obtained by scanning both faces of the printed document, the images that have been printed on each of its faces, with as little noise/noise from the other image.

In the example we are showing without the noise added, images printed text and graphs. The special character is in printed text and graphs in the asymmetric modes for consistency mode (black and white), although, due to the above mentioned noise, there will appear, in the separated images, as two clusters of intensity levels.

The separation of mixtures of two flat images, such as printed text, may be much easier than the separation of grayscale images. In fact, at least in the case of separation that are not too strong, a simple thresholding procedure may yield the desired results. Such a separation can be easily performed by hand with most image processing programs, and would not be hard to automate. In such a case, instead of using general state-of-the-art separation methods might be an overkill, both because it would involve a much larger amount of processing and because it might simply yield worse results. This is an extreme case in which prior knowledge about the sources can strongly simplify the separation process.

In the case of grayscale mixtures, the use of a separator method based on a good model of the visual mixing process would yield better results than the use of a generic nonlinear separation method. A physical model could have a small number of parameters to be estimated, and would thus allow a much more precise estimation. Furthermore, it might yield the inherent ill-conditionedness of nonlinear blind separation, easier to correctly addressed through regularization. The parameters of such a model could be estimated by an independent component analysis algorithm.

Another issue of interest is the definition of separation criteria that are more suited for images of a printed document than statistical independence. In fact, images and/or text from the opposite page of a printed document can easily happen not to be independent from one other. For example, images of landscapes tend to be lighter on the top than on the bottom, showing a correlation between intensities of both. Also, in printed text with regularly spaced lines, the lines from both sides of the paper may happen to be out of sync with each other, or the lines from one side may fall on the opposite of the lines from the other side, also inducing a significant correlation between the lines from both sides of the document. It would be interesting to work on a formal basis on a notion of image non-physics, but this may not be easy to define, and may be even harder to use as a criteria for defining a source separation algorithm.



To see all of materials presented in previous section and with making a simple experiment to zero set initial parameters. When you calculate the standard deviation of your initial vector, it is a good idea to initialize it with a value that is close to zero. This way, the gradient descent of the cost function will converge much faster than if you start with a random vector. It is also good to make sure that the gradient descent does not get stuck in local minima. To do this, you can add some noise to the initial vector. For example, you can add a small amount of noise to the initial vector by doing something like this: $\text{initial_vector} = \text{initial_vector} + \text{np.random.normal(0, 0.01, initial_vector.shape)}$.

Another good tip is to perform backpropagation on a linear model to validate your implementation. You can do this by writing a simple linear regression function and then writing a test function that checks the output of your function against the expected output. For example, you can write a function like this: $\text{def linear_function(x, w=1, b=0): return w * x + b}$.

Another tip is to perform backpropagation on a linear model to validate your implementation. You can do this by writing a simple linear regression function and then writing a test function that checks the output of your function against the expected output. For example, you can write a function like this: $\text{def linear_function(x, w=1, b=0): return w * x + b}$. Another good tip is to perform backpropagation on a linear model to validate your implementation. You can do this by writing a simple linear regression function and then writing a test function that checks the output of your function against the expected output. For example, you can write a function like this: $\text{def linear_function(x, w=1, b=0): return w * x + b}$.

Another good tip is to perform backpropagation on a linear model to validate your implementation. You can do this by writing a simple linear regression function and then writing a test function that checks the output of your function against the expected output. For example, you can write a function like this: $\text{def linear_function(x, w=1, b=0): return w * x + b}$. Another good tip is to perform backpropagation on a linear model to validate your implementation. You can do this by writing a simple linear regression function and then writing a test function that checks the output of your function against the expected output. For example, you can write a function like this: $\text{def linear_function(x, w=1, b=0): return w * x + b}$.

Another good tip is to perform backpropagation on a linear model to validate your implementation. You can do this by writing a simple linear regression function and then writing a test function that checks the output of your function against the expected output. For example, you can write a function like this: $\text{def linear_function(x, w=1, b=0): return w * x + b}$.

Another good tip is to perform backpropagation on a linear model to validate your implementation. You can do this by writing a simple linear regression function and then writing a test function that checks the output of your function against the expected output. For example, you can write a function like this: $\text{def linear_function(x, w=1, b=0): return w * x + b}$.

Another good tip is to perform backpropagation on a linear model to validate your implementation. You can do this by writing a simple linear regression function and then writing a test function that checks the output of your function against the expected output. For example, you can write a function like this: $\text{def linear_function(x, w=1, b=0): return w * x + b}$.

Another good tip is to perform backpropagation on a linear model to validate your implementation. You can do this by writing a simple linear regression function and then writing a test function that checks the output of your function against the expected output. For example, you can write a function like this: $\text{def linear_function(x, w=1, b=0): return w * x + b}$.

Another good tip is to perform backpropagation on a linear model to validate your implementation. You can do this by writing a simple linear regression function and then writing a test function that checks the output of your function against the expected output. For example, you can write a function like this: $\text{def linear_function(x, w=1, b=0): return w * x + b}$.

Another good tip is to perform backpropagation on a linear model to validate your implementation. You can do this by writing a simple linear regression function and then writing a test function that checks the output of your function against the expected output. For example, you can write a function like this: $\text{def linear_function(x, w=1, b=0): return w * x + b}$.

Another good tip is to perform backpropagation on a linear model to validate your implementation. You can do this by writing a simple linear regression function and then writing a test function that checks the output of your function against the expected output. For example, you can write a function like this: $\text{def linear_function(x, w=1, b=0): return w * x + b}$.

Another good tip is to perform backpropagation on a linear model to validate your implementation. You can do this by writing a simple linear regression function and then writing a test function that checks the output of your function against the expected output. For example, you can write a function like this: $\text{def linear_function(x, w=1, b=0): return w * x + b}$.

Another good tip is to perform backpropagation on a linear model to validate your implementation. You can do this by writing a simple linear regression function and then writing a test function that checks the output of your function against the expected output. For example, you can write a function like this: $\text{def linear_function(x, w=1, b=0): return w * x + b}$.

Another good tip is to perform backpropagation on a linear model to validate your implementation. You can do this by writing a simple linear regression function and then writing a test function that checks the output of your function against the expected output. For example, you can write a function like this: $\text{def linear_function(x, w=1, b=0): return w * x + b}$.

Another good tip is to perform backpropagation on a linear model to validate your implementation. You can do this by writing a simple linear regression function and then writing a test function that checks the output of your function against the expected output. For example, you can write a function like this: $\text{def linear_function(x, w=1, b=0): return w * x + b}$.

Another good tip is to perform backpropagation on a linear model to validate your implementation. You can do this by writing a simple linear regression function and then writing a test function that checks the output of your function against the expected output. For example, you can write a function like this: $\text{def linear_function(x, w=1, b=0): return w * x + b}$.

Another good tip is to perform backpropagation on a linear model to validate your implementation. You can do this by writing a simple linear regression function and then writing a test function that checks the output of your function against the expected output. For example, you can write a function like this: $\text{def linear_function(x, w=1, b=0): return w * x + b}$.

Another good tip is to perform backpropagation on a linear model to validate your implementation. You can do this by writing a simple linear regression function and then writing a test function that checks the output of your function against the expected output. For example, you can write a function like this: $\text{def linear_function(x, w=1, b=0): return w * x + b}$.

Another good tip is to perform backpropagation on a linear model to validate your implementation. You can do this by writing a simple linear regression function and then writing a test function that checks the output of your function against the expected output. For example, you can write a function like this: $\text{def linear_function(x, w=1, b=0): return w * x + b}$.

Another good tip is to perform backpropagation on a linear model to validate your implementation. You can do this by writing a simple linear regression function and then writing a test function that checks the output of your function against the expected output. For example, you can write a function like this: $\text{def linear_function(x, w=1, b=0): return w * x + b}$.

Another good tip is to perform backpropagation on a linear model to validate your implementation. You can do this by writing a simple linear regression function and then writing a test function that checks the output of your function against the expected output. For example, you can write a function like this: $\text{def linear_function(x, w=1, b=0): return w * x + b}$.

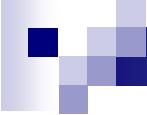
Another good tip is to perform backpropagation on a linear model to validate your implementation. You can do this by writing a simple linear regression function and then writing a test function that checks the output of your function against the expected output. For example, you can write a function like this: $\text{def linear_function(x, w=1, b=0): return w * x + b}$.

Another good tip is to perform backpropagation on a linear model to validate your implementation. You can do this by writing a simple linear regression function and then writing a test function that checks the output of your function against the expected output. For example, you can write a function like this: $\text{def linear_function(x, w=1, b=0): return w * x + b}$.

Another good tip is to perform backpropagation on a linear model to validate your implementation. You can do this by writing a simple linear regression function and then writing a test function that checks the output of your function against the expected output. For example, you can write a function like this: $\text{def linear_function(x, w=1, b=0): return w * x + b}$.

Another good tip is to perform backpropagation on a linear model to validate your implementation. You can do this by writing a simple linear regression function and then writing a test function that checks the output of your function against the expected output. For example, you can write a function like this: $\text{def linear_function(x, w=1, b=0): return w * x + b}$.

Another good tip is to perform backpropagation on a linear model to validate your implementation. You can do this by writing a simple linear regression function and then writing a test function that checks the output of your function against the expected output. For example, you can write a function like this: $\text{def linear_function(x, w=1, b=0): return w * x + b}$.



Objective quality measures

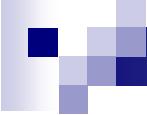
- Three quality measures were computed
 - Q1 – SNR between extracted and original source
 - Q2 – Same as Q1, but compensated for possible nonlinear intensity distortion
 - Q3 – Mutual information between extracted and original source

Results

Quality measure	Physical model	MISEP MLP	Nonl. DSS
Q_1 (dB)	12.4	11.8	11.9
Q_2 (dB)	13.6	13.0	13.3
Q_3 (bit)	2.12	1.98	2.07

MISEP MLP – MISEP using an MLP in the separation block (L.Almeida JMLR 2005)

Nonl. DSS – Nonlinear Denoising Source Separation (M.S.C Almeida ICA 2006)



Conclusions

- A physical model for the mixing process of scanned images was presented
- The inverse model was trained using MISEP (ICA)
- The separation quality is better than those obtained with previous methods for the same data (according to objective quality measures)
- The physical model fits the mixture process well
- MISEP is appropriate for estimating the model parameters