Introduction to signal processing
Practical session 1

2021-2022

The goal of this practical activity is to (numerically) investigate the notions of "exact recovery", "stability" and "robustness to noise" introduced during the first two lectures. For the sake of interpretability, we will restrict our attentions to image signal, see *e.g.* Figure 1.

Consider the following linear measurement problem: let $\mathbf{s} \in \mathbb{R}^n$ be a signal –a vectorized image here– and $\mathbf{M} \in \mathbb{R}^{m \times n}$ a linear sensing operator. Then, the observed signal $\mathbf{y} \in \mathbb{R}^m$ reads

$$\mathbf{y} = \mathbf{M}\mathbf{s}. \tag{1}$$

We also assume that $m < n$, *i.e.*, the signal is partially observed, and want to leverage the fact that $\mathbf{s}$ is known to lie in a lower dimensional subspace $\mathcal{S}_{\text{low}} \subset \mathbb{R}^k$ (with $k \ll n$) to design efficient decoders. More precisely, we assume the existence of a subspace (represented by a matrix $\mathbf{V} \in \mathbb{R}^{n \times k}$) such that for all considered signals $\mathbf{s}$, there exists a vector of k scalars $\boldsymbol{\alpha} \in \mathbb{R}^k$ such that

$$\mathbf{s} = \mathbf{V}\boldsymbol{\alpha}. \tag{2}$$

We eventually consider the noisy setting where the observation $\mathbf{y}$ is corrupted by an additive (potentially random) noise vector $\mathbf{n} \in \mathbb{R}^m$ following[1]

$$\mathbf{y}_\varepsilon = \mathbf{y} + \mathbf{n}. \tag{3}$$

Instructions. This lab session is **not graded**. Since some python code is provided with the announcement, you are asked to implement your lab with Python 3 and the numpy library.

[1] Note that, compared to the lecture, we subscript the observation vector $\mathbf{y}$ by "ε" to emphasize the presence of noise in the notation.

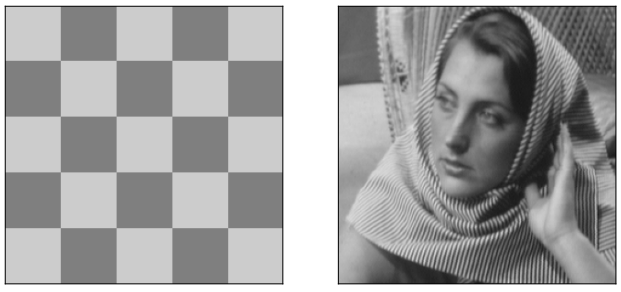

Figure 1

1 Preliminary questions

Prior to address the core of the practical session, let us implement some useful functions that will easier image manipulation.

As mentioned in the introduction, we will consider (square) image as signal. A black and white square image is commonly represented in signal processing as a matrix $\mathbf{S} \in [0,1]^{p \times p}$ where

- a 0 corresponds to a black pixel,
- a 1 corresponds to a white pixel,
- p represents the height and width of the image.

Questions.

1. Implement a code snippet that allows to visualize a black and white image.
2. Create and visualize an image where all pixels are constant and equal to 0.5. What do you observe?
3. Same question but set one pixel to 0.6. What happens?
4. Comment.

All the tools for inference introduced in the lecture sessions target signals that are represented by vectors. We therefore need a transformation to map images to vectors.

Questions.

5. How to represent an image $\mathbf{S} \in [0,1]^{p \times p}$ as a vector?
6. What will be the dimension n of the subsequent vector?
7. Implement the function "`image_to_vector`" that takes a square image as input and returns the vectorized image as output.
8. Conversely, implement the function "`vector_to_image`" that takes a vector as input and returns a square image as output.
9. Check that your code is consistent, that is, the output of "`vector_to_image`" with a vectorized image as input is equal to the initial image.

Remark. In the sequel, we keep using the notations of this section: p will refer to the height/width of a square image and n to the size of its vector representation. Besides, we will also make the following *abuse of notation*: the term "image" will refer to its vector representation.

2 On chess-like images

Let us return to the notion of "chess like" images seen during the lecture class and recalled in Figure 1. In this section, you can set $p = 5$.

Preliminary questions.

10. How to mathematically represent the subspace of chess-like images?
11. What is the dimension k of the latter subspace?

12. Propose an orthogonal basis of this subspace. Denote $\mathbf{V} \in \mathbb{R}^{n\times k}$ its matrix representation and evaluate it.

13. Implement a function that, given k scalars, returns the corresponding element of the subspace.

In the rest of the section, the signal of interest $\mathbf{s}^\star$ will be a chess-like image. The subscript * stands for "quantity we want to infer". Similarly, the notation $\hat{\mathbf{s}}$ will refer to "an estimate of $\mathbf{s}^\star$".

2.1 On exact recovery with pixel picking

Take some time to remember the setup of the chess-like image given in the lecture. In particular, recall that we have used a linear sensing operator that consisted in extracting some well-chosen pixels from the image.

14. Let $\mathbf{M}$ be the sensing matrix that extracts the first two pixels of the image. What are the dimensions m, n of $\mathbf{M}$?

15. Implement it.

16. Define the observation vector as in (1).

We are now interested in recovering a chess-like image $\mathbf{s}^\star$ from the few observed pixels $\mathbf{y}$. In particular, two decoders will be compared in this lab:

- A decoder $\mathscr{D}_1$ that use no prior information about the knowledge of $\mathbf{s}$. In particular, $\mathscr{D}_1$ is defined as a solution to the least square problem

$$\begin{aligned} \mathscr{D}_1 \colon \mathbb{R}^m &\to \mathbb{R}^n \\ \mathbf{y} &\mapsto \hat{\mathbf{s}} \in \arg\min_{\mathbf{s}\in\mathbb{R}^n} \|\mathbf{y} - \mathbf{M}\mathbf{s}\|_2^2 . \end{aligned} \tag{4}$$

- A decoder $\mathscr{D}_2$ that leverage the knowledge of $\mathcal{S}_{\mathrm{low}}$. In particular, $\mathscr{D}_2$ is defined as

$$\begin{aligned} \mathscr{D}_2 \colon \mathbb{R}^m &\to \mathbb{R}^n \\ \mathbf{y} &\mapsto \hat{\mathbf{s}} = \mathbf{V}\hat{\boldsymbol{\alpha}} \end{aligned} \tag{5}$$

  where $\hat{\boldsymbol{\alpha}}$ is given by

$$\hat{\boldsymbol{\alpha}} \in \operatorname*{arg\,min}_{\boldsymbol{\alpha}\in\mathbb{R}^k} \|\mathbf{y} - \mathbf{M}\mathbf{V}\boldsymbol{\alpha}\|_2^2. \tag{6}$$

Questions.

17. Implement the two decoders.

18. Apply them to the observed signal $\mathbf{y}$. Visualize and discuss the results.

19. What happens when more than two pixels are observed, *i.e., when* $m > 2$*?*

At this stage, let us introduce a standard metric in signal processing to evaluate the quality of the reconstruction. More precisely, given a signal $\mathbf{s}^\star$ to recover and an estimate $\hat{\mathbf{s}}$, we define the reconstruction signal to noise ratio (SNR) as

$$\mathrm{SNR}_{\mathtt{rec}}(\mathbf{s}^\star, \hat{\mathbf{s}}) \triangleq 10\log\left(\frac{\|\mathbf{s}^\star\|_2^2}{\|\mathbf{s}^\star - \hat{\mathbf{s}}\|_2^2}\right) \tag{7}$$

that is, the ratio (in logarithm) between the "energy"[2] of the signal and the "energy" of the the residual. One easily sees that the closer the residual is to zero, the higher is the SNR. Note also that a SNR is expressed in decibels (dB).

[2] note that we left the notion of energy (purposely) undefined in this lab.

Question.

20. For the two decoders, plot the reconstruction SNR as a function of the number of observed pixels.

2.2 A more general sensing matrix

We now consider a more general sensing operator. Let $\mathbf{M} \in \mathbb{R}^{m \times n}$ be the sensing matrix such that the entries are i.i.d. realizations of a standard normal distribution. In other words, the sensing operation now consists in taking the inner product between our (beloved) image and a random (Gaussian) vectors.

Questions

21. For the two decoders, plot the reconstruction SNR as a function of the number of measurement m. Compare with the previous results.

2.3 About the robustness to noise

We now consider the noisy observation model (3). Here, the noisy additive term $\mathbf{n}$ is assumed to be a vector of m i.i.d. random realizations of a centered Gaussian distribution with standard deviation σ. Note that the parameter σ controls the noise level. The higher σ, the more corrupted $\mathbf{y}$.

To quantify the corruption of the observation, we use a SNR metric which reads

$$\text{SNR}(\mathbf{y}, \mathbf{n}) \triangleq 10 \log \left(\frac{\|\mathbf{y}\|_2^2}{\|\mathbf{n}\|_2^2} \right) \tag{8}$$

that is, the ratio (in logarithm) between the energy of the observation and the energy of the noise.

Questions.

22. Set $\sigma = 0.5$. For the two decoders, plot the reconstruction SNR as a function of the number of measurement m.

23. Set $m = 5$. For the two decoders, plot the reconstruction SNR as a function of the SNR of the observation vector.

24. Comment the results.

3 An invitation to image processing

To conclude this lab, we invite you to test your decoders on a natural image. Although our methodology applies to any image[3], we restrict our attention to a small subset of "*barbara*" proposed in the right side of Figure 1 to lower the computational burden. Using images of higher dimensions would require some implementation tricks that are out of scope of the class.

As mentioned during the lectures, there is no tractable representation of natural images. However, specialists agree that they are "well represented" by the first few coefficients of their discrete cosine transform (DCT). Interestingly, this idea is at the heart of the jpeg standard. Yet, the mathematical definition of the DCT is out of scope of the class and the interested student is referred to the wikipedia page for a start.

Instead we provide you the function `get_subspace_image` that, given the square root of k and the height of the image, return a basis of the latter subset of dimension k.

Questions.

25. Load `imagette.png`. What is the value of p?
26. For several values of m and k, try to infer the (subset of) barbara image from partial measurement. *advice:* start with low values.
27. Discuss the interest of the method.

[3] in particular, your holiday pictures ;p

```
def get_subspace_image(c, p):
    """ returns a basis of the subspace of dimension c**2
    that well approximate natural images

    Parameters
    ----------
    c: int
        square root of the the size of the subspace
    p: int
        square root of the number of pixels

    Returns
    -------
    V: np.ndarray
    """
    import numpy as np
    from scipy.fftpack import dct, idct
    from scipy.fft import dctn, idctn
    n=p**2
    V = np.zeros((n, c**2))
    counter = 0
    for i in range(c):
        for j in range(c):
            e = np.zeros((p, p))
            e[i, j] = 1
            V[:, counter] = np.reshape(idctn(e, type=2), (n,))

            counter += 1
    return V
```