lions@epfl

EE-556: MATHEMATICS OF DATA: FROM THEORY TO COMPUTATION
LABORATORY FOR INFORMATION AND INFERENCE SYSTEMS
FALL 2018

ÉCOLE POLYTECHNIQUE FÉDÉRALE DE LAUSANNE

INSTRUCTOR:
PROF. VOLKAN CEVHER

HEAD TA:
YA-PING HSIEH

HOMEWORK 3 (FOR WEEKS 10-12)

This laboratory consists of four parts. The first part is devoted to a novel subsampling strategy, which has not been covered in the course, but is easy to understand. In the second part, we consider structure-promoting regularizer functions and examine their respective proximity operators, with an application to image denoising. In the third part, we consider non-smooth composite minimization. We will implement the FISTA algorithm and use it to reconstruct an MRI image from its subsamples. In the last part, we consider the problem of image in-painting, where we use the structure of an image to infer the values of its missing pixels, and study the convergence of variations of the FISTA algorithms.

1 Learning-based Compressive Subsampling

In the lectures, we studied the problem of recovering a vector $\mathbf{x}^\natural \in \mathbb{C}^p$ from a set of linear measurements of the form

$$\mathbf{b} = \mathbf{A}\mathbf{x}^\natural, \tag{1}$$

where $\mathbf{A} \in \mathbb{C}^{n\times p}$ is a known *measurement matrix*. We saw that this problem is ill-posed unless $n \geq p$, whereas surprisingly, accurate reconstruction remains possible even when n is much smaller than p provided that $\mathbf{x}^\natural$ exhibits some structure such as *sparsity* or *compressibility*. In this lab, we will look at various sparse representations and reconstruction algorithms for this problem.

1.1 Problem Statement

In the first part of this lab, we will consider a special form of measurement matrix $\mathbf{A}$ based on *subsampling* in some fixed orthonormal basis. Specifically, we let $\mathbf{\Psi} \in \mathbb{C}^{p\times p}$ be a unitary matrix (i.e., a matrix such that $\mathbf{\Psi}\mathbf{\Psi}^* = \mathbf{I}$, where $(\cdot)^*$ denotes the conjugate transpose), so that the rows of $\mathbf{\Psi}$ form an orthonormal basis for $\mathbb{C}^p$. We then design $\mathbf{A}$ in (1) by taking subsamples:

$$\mathbf{b} = \mathbf{P}_\Omega \mathbf{\Psi}\mathbf{x}^\natural. \tag{2}$$

for some subsampling matrix $\mathbf{P}_\Omega \in \{0,1\}^{n\times p}$ whose rows are canonical basis vectors containing a 1 in one entry and 0 in all other entries. The set $\Omega \subseteq \{1,\dots,p\}$ indexes the entries of $\mathbf{\Psi}\mathbf{x}^\natural$ that are subsampled; for example, if $\Omega = \{1,3\}$ then we have

$$\mathbf{P}_\Omega = \begin{bmatrix} 1 & 0 & 0 & 0 & \cdots & 0 \\ 0 & 0 & 1 & 0 & \cdots & 0 \end{bmatrix}, \tag{3}$$

meaning that we only sample the first and third entries.

In this part of the lab, we seek to answer the following question: *If we are allowed to subsample a given number n of the indices* $\{1,\dots,p\}$*, how should we choose them?* The answer to this question depends on the structure known to be present in $\mathbf{x}^\natural$, the estimation algorithm used, and the performance criterion used in assessing the quality of the resulting estimate.

1.2 Learning Good Indices from Training Data - Linear Case

If we knew the signal $\mathbf{x}^\natural$ perfectly when we were choosing the indices, we might choose the n indices of Ω in order to maximize the total amount of energy captured: $\hat{\Omega} = \arg\max_{\Omega\,:\,|\Omega|=n} \|\mathbf{P}_\Omega \mathbf{\Psi}\mathbf{x}^\natural\|_2^2$. This corresponds to choosing the indices that give the *best n-term approximation* to $\mathbf{x}^\natural$ in the basis specified by $\mathbf{\Psi}$.

Suppose now that we do not know $\mathbf{x}^\natural$, but that we do have access to some number m of *fully sampled training signals* $\mathbf{x}_1,\dots,\mathbf{x}_m$. Here and throughout the lab, we will assume without loss of generality that $\|\mathbf{x}_j\|_2^2 = 1$ for all j, since in general we can always normalize each signal by its magnitude to make this true. How can we choose a *single* set of indices that simultaneously works well for most or all of training signals, and more importantly, on a signal $\mathbf{x}^\natural$ that we have not seen yet? A natural choice is to select the indices that capture the most energy on average:

$$\hat{\Omega} = \arg\max_{\Omega\,:\,|\Omega|=n} \frac{1}{m}\sum_{j=1}^{m} \|\mathbf{P}_\Omega \mathbf{\Psi}\mathbf{x}_j\|_2^2. \tag{4}$$

If $\mathbf{x}^\natural$ is "similar" to the training signals, in the sense that the dominant indices tend to be the same, then we should expect these learned indices to capture the most of the energy in $\mathbf{x}^\natural$. We will make this statement more precise below.

Depending on the estimator and the performance measure, it may not necessarily be the case that maximizing the captured energy corresponds to achieving the best estimation error. However, it turns out that this is indeed the case when we consider the least-squares estimator along with the ℓ_2-error performance measure . Recalling that $\mathbf{A} = \mathbf{P}_\Omega \mathbf{\Psi}$, and letting $(\cdot)^\dagger$ denote the pseudo-inverse and $(\cdot)^*$ the conjugate transpose (also known as the adjoint), the least-squares estimator is given by

$$\hat{\mathbf{x}} = \mathbf{A}^\dagger \mathbf{b} = \mathbf{A}^*(\mathbf{A}\mathbf{A}^*)^{-1}\mathbf{b}, \tag{5}$$

and the squared-error is given by

$$\mathcal{E}(\mathbf{x}, \hat{\mathbf{x}}) = \|\mathbf{x} - \hat{\mathbf{x}}\|_2^2. \tag{6}$$

We saw in the lectures that the estimator in (5) can fail drastically when it comes to recovering *arbitrary* sparse signals. Fortunately, in imaging applications, the sparsity (or compressibility) patterns are often very similar among the images, and the least-squares approach can still work very well.

EXERCISE 1.1: (0 POINTS; SELF STUDY) - *Prove that the estimator in* (5) *is equivalent to expanding* $\mathbf{b}$ *to a p-dimensional vector by placing zeros in the entries corresponding to* $\Omega^c = \{1, \dots, p\} \backslash \Omega$*, and then applying the adjoint* $\mathbf{\Psi}^*$*; in other words, prove that*

$$\hat{\mathbf{x}} = \mathbf{\Psi}^* \mathbf{P}_\Omega^T \mathbf{b}. \tag{7}$$

(HINT: By definition, a unitary matrix has $\mathbf{\Psi}^* = \mathbf{\Psi}^{-1}$*. What can we say about* $\mathbf{P}_\Omega \mathbf{P}_\Omega^T$*? Try computing it in MATLAB or Python for various choices of* Ω *if you are unsure.)*

It can be shown that the estimator in (5) gives an estimation error of

$$\|\mathbf{x}^\natural - \hat{\mathbf{x}}\|_2^2 = \|\mathbf{x}^\natural\|_2^2 - \|\mathbf{P}_\Omega \mathbf{\Psi} \mathbf{x}^\natural\|_2^2, \tag{8}$$

which reveals that the estimation error for the least-squares estimator is determined solely by the amount of energy captured in the indices within Ω. This provides a justification for the index set selection rule given in (4).

EXERCISE 1.2: (3 POINTS) - *Show that the optimization problem in* (4) *can be solved by sorting. (HINT: First try to express the objective function in terms of* $\langle \boldsymbol{\psi}_i, \mathbf{x}_j \rangle$*, where* $\boldsymbol{\psi}_i^T$ *is the i-th row of* $\mathbf{\Psi}$*.)*

1.3 Learning Good Indices from Training Data - From Linear to Non-Linear

We have shown that the more energy is captured in some signal $\mathbf{x}^\natural$, the better the linear estimator in (5) performs in terms of the squared ℓ_2-error defined in (6). However, this intuition does not apply to the more interesting and general case of non-linear estimators. In this case, the estimation becomes

$$\hat{\mathbf{x}} = \Delta(\mathbf{P}_\Omega \mathbf{\Psi} \mathbf{x}^\natural). \tag{9}$$

where Δ is any non-linear decoder, which forms an estimate $\hat{\mathbf{x}}$ from the ground truth image undersampled in the basis $\mathbf{\Psi}$. Suppose now that, as in the previous case, we do not know $\mathbf{x}^\natural$ but we have access to fully sampled training signals $\mathbf{x}_1, \dots, \mathbf{x}_m$. The mask optimization problem defined in (4) needs then to be adapted to the non-linear case as follows

$$\hat{\Omega} = \underset{\Omega\,:\,|\Omega|=n}{\arg\max}\, \eta(\Omega), \text{ where } \eta(\Omega) := -\frac{1}{m}\sum_{j=1}^m \mathcal{E}(\mathbf{x}_j, \hat{\mathbf{x}}_j) = -\frac{1}{m}\sum_{j=1}^m \|\mathbf{x}_j - \Delta(\mathbf{P}_\Omega \mathbf{\Psi} \mathbf{x}_j)\|_2^2. \tag{10}$$

where η is called a performance measure, and measures how well a mask Ω performs on the training signals. Due to the fact that most non-linear algorithms do not admit a closed-form solution as the one of equation (7) in the linear case, the problem essentially becomes combinatorial. The alternative is to find an *approximate* minimizer, and we will consider the case of a greedy algorithm. To do so, we must first define more clearly the underlying structure of the mask Ω. We consider a set $\mathcal{S}$ of indices of $\{1, \dots, p\}$ that we choose to sample, and the mask takes the form

$$\Omega = \bigcup_{j=1}^n S_j, \quad S_j \in \mathcal{S}. \tag{11}$$

The procedure that the greedy algorithm follows is quite intuitive: it starts from an empty mask, looks at the marginal contribution of each element of $\mathcal{S}$ and picks the one that increase the performance measure the most. Then, having added this first element to the definitive mask, it carries on by testing the marginal contribution of the elements that have not yet been added to this mask, and runs until we have sampled n out of the p possible elements. We then say that the *cardinality constraint* $|\Omega| = n$ *is met*. The detailed procedure is given in Algorithm 1 below.

In the following exercise, we will study more deeply the complexity of this greedy algorithm.

EXERCISE 1.3: (6 POINTS) - *We now want to study the computational complexity of Algorithm 1. In the present case, the reconstruction algorithm Δ will be assumed to have a complexity C_Δ[1]. As the nonlinear reconstruction is the most expensive part of the greedy algorithm, we will consider the other operations (e.g. matrix-vector multiplication, evaluation of the performance measure $\eta, \dots$) to be negligible before it.*

The variables that we consider are the reconstruction complexity C_Δ, the number of elements that we will sample n, the number of training elements m, the space dimension p.

1. *What is the computational complexity of the greedy Algorithm 1 ? Explain the steps of your computation. How does adding k unique elements instead of 1 at each iteration could improve this complexity ?*

2. *This greedy algorithm can be greatly accelerated by carrying computations in parallel (several computations run simultaneously on different processes). The **for** loop on the second line of Algorithm 1 is an excellent example of computation that can be parallelized: the results of each iteration do not depend on each other, which means that they can be computed at the same time. How does parallelization improve the complexity of the greedy algorithm?*

   The complexity in the parallel case will correspond to calculating the complexity assuming that there is infinitely many available processes, and that communication will be instantaneous.

3. *The greedy algorithm can still be further parallelized. What can be parallelized and what is the resulting complexity of the algorithm?*

4. *This algorithm can be made more scalable by using a batch of elements $\mathcal{S}_t \in \mathcal{S}$ instead of the whole set $|\mathcal{S}| = p$. What is the effect of replacing $\mathcal{S}$ with a batch $\mathcal{S}_t$ of cardinality t on the overall complexity? What are the benefits of such a procedure and why could you expect it to produce good results?*

Algorithm 1 Greedy mask optimization

Input: Training data $x_1, \dots, x_m$, reconstruction rule Δ, sampling subset $\mathcal{S}$, maximum cost n
Output: Sampling pattern Ω

while $|\Omega| \leq n$ **do**
  for $S \in \mathcal{S}$ not yet in Ω **do**
    $\Omega' = \Omega \cup S$
    For each j, set $\mathbf{b}_j \leftarrow \mathbf{P}_{\Omega'}\mathbf{\Psi}\mathbf{x}_j$, $\hat{\mathbf{x}}_j \leftarrow \Delta(\mathbf{P}_{\Omega'}\mathbf{\Psi}\mathbf{x}_j)$
    $\eta(\Omega') \leftarrow -\frac{1}{m}\sum_{j=1}^{m} \mathcal{E}(\mathbf{x}_j, \hat{\mathbf{x}}_j)$
  $\Omega \leftarrow \Omega \cup S^*$, where

$$S^* = \underset{S:|\Omega \cup S| \leq n}{\arg\max}\, \eta(\Omega \cup S) - \eta(\Omega)$$

return Ω

1.4 Solution to self-study problem (Exercise 1.1)

First apply the formula in (5) with $\mathbf{A} = \mathbf{P}_\Omega\mathbf{\Psi}$, which implies $\mathbf{A}^* = \mathbf{\Psi}^*\mathbf{P}_\Omega^T$ since $\mathbf{P}_\Omega$ is real:

$$\hat{\mathbf{x}} = \mathbf{\Psi}^*\mathbf{P}_\Omega^T(\mathbf{P}_\Omega\mathbf{\Psi}\mathbf{\Psi}^*\mathbf{P}_\Omega^T)^{-1}\mathbf{b}. \tag{12}$$

Since $\mathbf{\Psi}$ is unitary, we have $\mathbf{\Psi}\mathbf{\Psi}^* = \mathbf{I}_n$, the $n \times n$ identity matrix. It is also the case that $\mathbf{P}_\Omega\mathbf{P}_\Omega^T = \mathbf{I}_k$ (with $k = |\Omega|$), since the rows of $\mathbf{P}_\Omega$ are orthonormal (e.g., try doing the multiplication directly in (3)!). Combining these gives $\mathbf{P}_\Omega\mathbf{\Psi}\mathbf{\Psi}^*\mathbf{P}_\Omega^T = \mathbf{I}_k$, and therefore $\hat{\mathbf{x}} = \mathbf{\Psi}^*\mathbf{P}_\Omega^T\mathbf{b}$.

2 Proximal operators and image denoising

In this and the next parts, by an *image* we mean a grayscale digital image expressed as a matrix, each entry of which represents the intensity of a pixel.

2.1 Wavelets

A widely used multi-scale localized representation in signal and image processing is the wavelet transform.[2] This representation is constructed so that piecewise polynomial signals have sparse wavelet expansions [3]. Since many real-world signals can be composed by piecewise smooth parts, it follows that they have sparse or compressible wavelet expansions.

[1] While this complexity is not constant, and depends on parameters such as the space dimension p, we write it as C_Δ for simplicity
[2] This section is an adaption from *Signal Dictionaries and Representations* by M. Watkin (see `http://bit.ly/1wXDDbG`).

2.1.1 Scale

In wavelet analysis, we frequently refer to a particular scale of analysis for a signal. In particular, we consider dyadic scaling, such that the supports from one scale to the next are halved along each dimension. For images, which can be represented with matrices, we can imagine the highest scale as consisting of each pixel. At other scales, each region correspond to the union of four neighboring regions at the next higher scale, as illustrated in Figure 1.

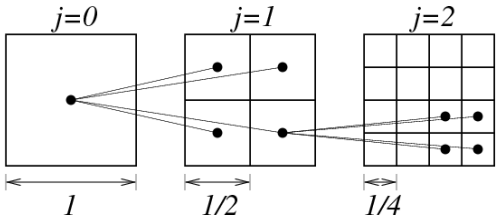

Figure 1: Dyadic partitioning of the unit square at scales $j = 0, 1, 2$. The partitioning induces a coarse-to-fine parent/child relationship that can be modeled using a tree structure.

2.1.2 The Wavelet transform

The wavelet transform offers a multi-scale decomposition of an image into coefficients related to different dyadic regions and orientations. In essence, each wavelet coefficient is given by the scalar product of the image with a wavelet function which is concentrated approximately on some dyadic square and has a given orientation, vertical, horizontal or diagonal. Wavelets are essentially bandpass functions that detect abrupt changes in a signal. The scale of a wavelet, which controls its support both in time and in frequency, also controls its sensitivity to changes in the signal.

An important property of the wavelet functions is that they form a basis $\mathbf{W}$ which is orthonormal, that is $\mathbf{W}\mathbf{W}^T = \mathbf{W}^T\mathbf{W} = \mathbf{I}$, where $\mathbf{I}$ is the identity operator. In the discrete case, $\mathbf{W}$ is just an orthonormal matrix.

2.1.3 Implementation

We provide the codes that help efficiently computing the 2D Wavelet transform $\mathbf{W}$ and its inverse $\mathbf{W}^{-1} = \mathbf{W}^T$. In order to be aligned with the notation used in the rest of the exercise, these transforms, $\mathbf{W}$ and $\mathbf{W}^T$, are recommended to be considered as matrix vector multiplications, i.e. they are, by definition, applied to vectorized images and they output vectorized transforms.

- `mdwt` (`dwt` in Python) computes the 2D Wavelet transform of an image and returns its wavelet coefficients;
- `midwt` (`idwt` in Python) computes the 2D inverse Wavelet transform: given an image of wavelet coefficients it returns the original image;
- As an example, `example_wavelet.m` and `example_wavelet.py` illustrate how to use these functions.
- These codes were written in C and have been precompiled to be used in OS X, Windows and Unix. For installation in Python, please check: https://github.com/ricedsp/rwt. We have tested the Python codes on Anaconda Python 2.7.
- The two functions take as input an $m \times m$ image `X`, a wavelet function definition `wav` and the maximum decomposition level, `level` (e.g., `level = log2(m)`).
- The wavelet function `wav` can be generated using the function `daubcqf`. We suggest using `daubcqf(8)` which generates the Daubechies-8 wavelet.

Remark 1. If the provided codes do not work, directly contact one of the teaching assistants.

2.2 Total Variation

The Total Variation was proposed by Rudin, Osher and Fatemi [5], as a regularizer for solving inverse problems. Much empirical evidence has shown that it is efficient for regularizing images without blurring the sharp edges of the images. Recent researches have been focusing on developing very efficient algorithms for computing its proximal operator. In this homework, we will exploit the approach developed by Chambolle [1].

In order to compute the Total Variation of an image, we first need to define a discrete gradient operator. The gradient $\nabla\mathbf{x}$ is an array in $\mathbb{R}^{(m\times m)\times 2}$ given by $(\nabla\mathbf{x})_{i,j} = \left((\nabla\mathbf{x})^1_{i,j}, (\nabla\mathbf{x})^2_{i,j}\right)$ with

$$(\nabla\mathbf{x})^1_{i,j} = \begin{cases} \mathbf{x}_{i+1,j} - \mathbf{x}_{i,j} & \text{if } i < m, \\ 0 & \text{if } i = m, \end{cases}$$

$$(\nabla \mathbf{x})^2_{i,j} = \begin{cases} \mathbf{x}_{i,j+1} - \mathbf{x}_{i,j} & \text{if } j < m, \\ 0 & \text{if } i = m. \end{cases}$$

Then, the discrete Total Variation (TV) is defined as

$$\|\mathbf{x}\|_{\text{TV}} := \begin{cases} \sum_{1 \le i,j \le m} \|(\nabla \mathbf{x})_{i,j}\|_2 & \text{called } \textit{isotropic} \\ \sum_{1 \le i,j \le m} |(\nabla \mathbf{x})^1_{i,j}| + |(\nabla \mathbf{x})^2_{i,j}| & \text{called } \textit{anisotropic} \end{cases}$$

Even though the isotropic case presents a slightly more difficult computational challenge, it has the advantage of being unaffected by the direction of the edges in images.

One can interpret the Total Variation as the ℓ_1-norm of the absolute values of the gradients evaluated at each pixel. Therefore, in analogy to the ℓ_1-norm of the coefficients of a vector, minimizing the Total Variation subject to some data fidelity constraints, would lead to sparse gradients, which implies an image with many flat regions and few sharp transitions; see Figure 2.

2.3 Image Denoising

In this part of the homework, we leverage the structures described in the previous section to remove the noise from an image. In other words, we want to find an approximation of the noisy image that possesses one of the mentioned structures. These structures can be measured by a proper norm, for example the ℓ_1 norm on the wavelets coefficients, or the TV-norm on the image domain, for which lower values indicate a more evident structure. We can therefore solve the following optimization problems:

$$\min_{\boldsymbol{\alpha} \in \mathbb{R}^p} \Big\{ \underbrace{(1/2)\|\mathbf{y} - \mathbf{W}^T \boldsymbol{\alpha}\|^2_{\text{F}}}_{f(\boldsymbol{\alpha})} + \underbrace{\lambda_1 \|\boldsymbol{\alpha}\|_1}_{g(\boldsymbol{\alpha})} \Big\}, \tag{13}$$

$$\min_{\mathbf{x} \in \mathbb{R}^p} \Big\{ \underbrace{(1/2)\|\mathbf{y} - \mathbf{x}\|^2_{\text{F}}}_{f(\mathbf{x})} + \underbrace{\lambda_{\text{TV}} \|\mathbf{x}\|_{\text{TV}}}_{g(\mathbf{x})} \Big\}, \tag{14}$$

where $\mathbf{y} \in \mathbb{R}^p$, is the vectorized image of length $p = m \times m$ to be denoised, $\mathbf{W}^T$ is the 2D inverse Wavelet transform applied to a vectorized image and returns the vectorized transform image, λ_1 and $\lambda_{\text{TV}} > 0$ are regularization parameters that trade-off approximation accuracy and structure[3].

Remark 2. For the rest of this homework, make sure that all the operators are size consistent. In other words, if an image is given as input, the operators must be able to work on images and return images. In this case you will consider the matrix $\mathbf{W}^T$ as a linear operator: $\mathbf{W}^T : \mathbb{R}^{m \times m} \to \mathbb{R}^{m \times m}$. An alternative is to vectorize all images, and have the operators act on only vectors and return vectors which is the form suggested by the matrix-vector multiplication in (13) where $\mathbf{W}^T$ takes a vectorized image $\boldsymbol{\alpha}$, reshapes it back to a matrix; i.e. the usual 2-D image form, applies the inverse wavelet transform and then again vectorizes it before returning it as the output.

Note that the last problem, (14), corresponds to computing the proximal operator of $g(\mathbf{x}) := \lambda_{\text{TV}} \|\mathbf{x}\|_{\text{TV}}$ at $\mathbf{y}$.

2.3.1 Computation of the proximal operator

Given a convex function $g : \mathbb{R}^p \to \mathbb{R} \cup \{+\infty\}$, we recall the proximal operator of g as the solution of the following convex problem:

$$\text{prox}_g(\mathbf{x}) := \arg\min_{\mathbf{y} \in \mathbb{R}^p} \{ g(\mathbf{y}) + (1/2)\|\mathbf{y} - \mathbf{x}\|_2^2 \}. \tag{15}$$

See Lectures 8 for more details.

EXERCISE 2.1: (3 POINTS) - *Show that problem* (13) *can be transformed into the computation of the proximal operator of the function* $g(\mathbf{x}) := \lambda_1 \|\mathbf{x}\|_1$.

HINT: Use the fact that the wavelet basis is orthonormal: $\mathbf{W}\mathbf{W}^T = \mathbf{W}^T\mathbf{W} = \mathbf{I}$, *where* $\mathbf{I}$ *is the identity operator.*

EXERCISE 2.2: (7 POINTS) - *Given* $g(\mathbf{x}) := \|\mathbf{x}\|_1$, *show that the proximal function of* $g(\mathbf{x})$ *can be written as*

$$\text{prox}_{\lambda g}(\mathbf{z}) = \max(|\mathbf{z}| - \lambda, 0) \otimes \text{sign}(\mathbf{z}) =: S_\lambda(\mathbf{z}) \tag{16}$$

where the operator max *and* sign *are applied component-wise to the vector* $\mathbf{z}$ *and* $\otimes$ *stands for the component-wise multiplication; i.e.,* $(\mathbf{x} \otimes \mathbf{y})_i = x_i y_i$.

The proximal operator of the ℓ_1-norm has therefore a closed form solution (i.e., exactly and analytically) and this solution corresponds to so-called *soft thresholding* operator. On the other hand it is not possible to compute the proximal operator of a TV-norm, $g(\mathbf{x}) := \|\mathbf{x}\|_{\text{TV}}$, in closed form. An efficient algorithm has been proposed by Chambolle [1]. This algorithm is based on reformulating the original problem as a projection onto a specific closed convex set. This projection can be approximated iteratively by a simple scheme

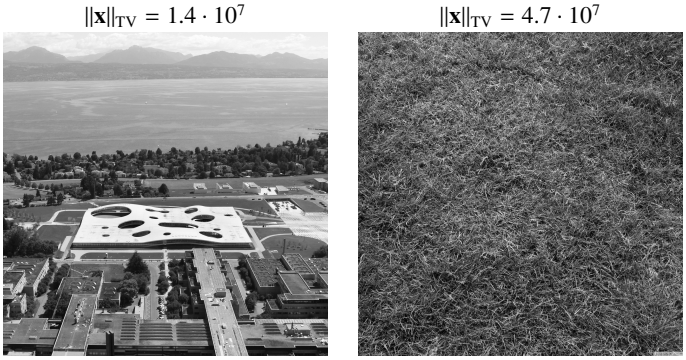

Figure 2: Comparison of Total Variation on two images. The EPFL campus image has fewer edges and more homogeneous regions, which leads to a smaller value for its Total Variation.

and it is guaranteed to converge. The interested student can find more details in [1] and try to implement the algorithm. However, we already provide a Matlab and Python implementations of this algorithm, see `example_tv_prox.m` and `example_tv_prox.py` for an example on how to use it. For Python you can use `denoise_tv_chambolle` function from scikit-image restoration module.

Throughout this homework, we will assess the quality of the estimated images via the Peak Signal-to-Noise Ratio.

Definition 1 (PSNR)**.** *The* Peak Signal to Noise Ratio *(PSNR) is an engineering term for the ratio between the maximum possible power of a signal and the power of corrupting noise that affects the fidelity of its representation. PSNR is most easily defined via the mean squared error (MSE). Given a noise-free $m \times m := N$ monochrome image I and its noisy approximation $\hat{I}$, MSE is defined as:*

$$MSE(I, \hat{I}) := \frac{1}{N} \left\| I - \hat{I} \right\|_F^2 .$$

The PSNR in decibels (dB) is then defined as:

$$PSNR(I, \hat{I}) := 20 \log_{10} \left(\frac{\max(I)}{\sqrt{MSE(I, \hat{I})}} \right)$$

We are now going to use the proximal operators of the ℓ_1-norm and the Total Variation-norm to denoise two images. A denoising example is given in Figure 3.

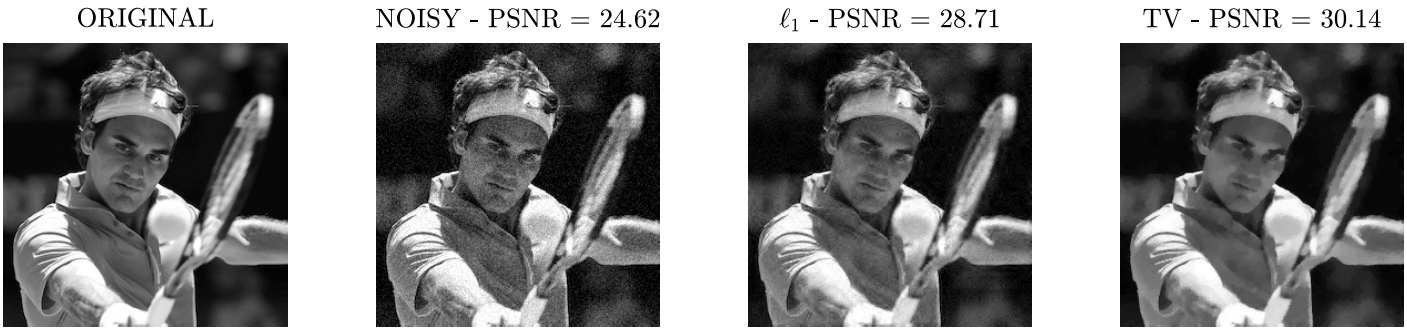

Figure 3: An example of image denoising with the two models described in this homework. The regularization parameter has been set to 15 for both methods.

[3] In Matlab the vectorization operation of an image $\mathbf{x}$ can be performed as `x(:)`.

EXERCISE 2.3: (10 POINTS) - *Denoising.*

1. *Take a natural image, or better, a picture of you, and load it using the Matlab function* `imread`.
2. *Convert the image to grayscale via the function* `rgb2gray`.
3. *Convert the grayscale image into double, simply by typing* `double(variable_name)`.
4. *Resize the image to* 256×256 *pixels with* `imresize`.
5. *Create a noisy image,* $\mathbf{y}$*, by adding Gaussian noise with standard deviation* $\sigma = 15$*:*
   `y = I_noise_free + 15*randn(size(I_noise_free))`.
6. *Choose values for the regularization parameters* λ_1, λ_{TV} *(positive values).*
7. *Denoise the image by solving problems* (13) *exactly, and problem* (14) *up to a given accuracy.*
8. *Compute the PNSR of the denoised images.*
9. *Visualize the denoised images alongside the original and the noisy image, as in Figure 3.*
10. *Try with very large and very small* λ *values and explain your observations.*
11. *Provide your codes with comments and include the reconstructed images in your report.*

In the previous exercise, the choice of the value of the regularization parameters was arbitrary. You might have noticed that the same value leads to different estimates for the two models and it is not clear which model is better for denosing. This is due to the fact that the two norms give different measures of structures in the image. When doing a simulation like in the previous case, we have access to the noise-free image, and therefore we can assess the performance of each method for a wide range of values for the regularization parameter. This procedure is usually called a parameter sweep and it allows to understand which method can, potentially, obtain the best performance.

EXERCISE 2.4: (4 POINTS) *Using the same image, solve the denoising problems* (13) *and* (14) *on a certain range of the regularization parameter values; e.g., on a range of* 20 *different values. Then plot the obtained PSNR values as a function of the regularization parameter. What are the maximum PSNR values that each of the two regularization methods achieves ? What are the good* λ *values that give these PSNR values ? You are supposed to plot on a meaningful range where you can see the behaviour of the PSNR as a function of the regularization parameter, preferably with the best parameter located near the middle of the x-axis.*

HINT: Matlab functions `linspace` *or* `logspace` *might help to take a meaningful range for the parameter sweep.*

3 Non-smooth composite minimization and compressive MRI

In Compressive Magnetic Resonance Imaging (MRI), we are interested in recovering a structured signal $\mathbf{x}^\natural \in \mathbb{C}^p$ *from measurements* $\mathbf{b} = \mathbf{A}\mathbf{x}^\natural$*, where* $\mathbf{A} = \mathbf{P}_\Omega \mathbf{F} \in \mathbb{C}^{n \times p}$ *is the dimensionality reducing Fourier measurement operator, with* $n \ll p$*. (In MRI imaging, the measurements are taken in Fourier domain). By exploiting the structures commonly found in natural images, such as sparsity over the wavelet coefficients, it is possible to reconstruct MRI images using less number of its Fourier coefficients. This can reduce the scan time and therefore increase the patient comfort. We will reconstruct the original image by solving the following different regularization problem:*

$$\min_{\boldsymbol{\alpha} \in \mathbb{C}^p} \underbrace{\frac{1}{2}\|\mathbf{b} - \mathbf{P}_\Omega \mathbf{F}\mathbf{W}^T\boldsymbol{\alpha}\|_2^2}_{f(\boldsymbol{\alpha})} + \underbrace{\lambda_1 \|\boldsymbol{\alpha}\|_1}_{g(\boldsymbol{\alpha})}, \tag{17}$$

where $\mathbf{W}^T$ *is the inverse 2D Wavelet transform used in the previous parts,* $\mathbf{F}$ *is the 2D Fourier Transform and* $\mathbf{P}_\Omega \in \mathbb{R}^{n \times p}$ *is an operator that selects only few,* $n \ll p := m^2$*, pixels from the vectorized image* $\mathbf{x} \in \mathbb{R}^p$*. In Matlab, the operation* $\mathbf{P}_\Omega \mathbf{x}$ *can simply be implemented as* `x(ind)`*, where* `ind` *is a vector containing the elements of* $\Omega \subseteq \{1, \ldots, p\}$*. Refer again to Remark 2.*

The problem above is unconstrained and the objective function consists of two convex terms: the smooth data fidelity term f *and the non-smooth structure-inducing norm* g *which enforces a sparse solution in the wavelet domain in* (17)*. As we have seen in the previous part, the proximal operators of the function* g *can be computed efficiently. In this homework, we focus on the accelerated proximal-gradient descent, also known as FISTA.*

Remark 3. Note that MRI images are complex-valued and the Fourier transform also introduces complex numbers. Therefore the gradient computation should be done by separating the real and complex parts of the complex numbers.

EXERCISE 3.1: (10 POINTS) -

1. *Write $f(\boldsymbol{\alpha})$ in* (17) *as a function $\tilde{f}$ of $\boldsymbol{\alpha}_R$ and $\boldsymbol{\alpha}_I$ where $\boldsymbol{\alpha} = \boldsymbol{\alpha}_R + \boldsymbol{\alpha}_I i$. Find the gradient, $\nabla \tilde{f}(\boldsymbol{\alpha}_R, \boldsymbol{\alpha}_I)$. (HINT: Check the solutions of Recitation 3 about the gradient of functions with complex variables, and be careful while taking the adjoint of a complex variable.)*

2. *Find the Lipschitz constant of $\nabla \tilde{f}(\boldsymbol{\alpha}_R, \boldsymbol{\alpha}_I)$ either analytically or computationally by power iterations.*

EXERCISE 3.2: (16 POINTS) - *Implement the FISTA algorithm with restart for solving:*

$$\min_{\alpha_R, \alpha_I} \{ f\alpha_R, \alpha_I) + g(\alpha_R, \alpha_I) \}$$

where f and g are convex functions. The algorithm must take as input f, ∇f, g and the Lipschitz constant of ∇f. Other arguments are the algorithm parameters, such as maximum number of iterations and tolerance for stopping criterion. Use exact non-monotonicity test as the restart criterion. Then apply the algorithm to the problem (17) *in order to do the reconstruction of the brain image on the right.*

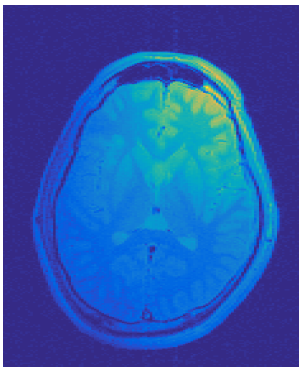

Figure 4: Brain image

- *Take the 256×256 brain image stored in* `brain.mat`*. Take also the indices stored in* `indices_brain.mat` *as the Ω in the problem formulations.*
- *Subsample the image in Fourier domain using the indices in Ω to record the measurement vector $\mathbf{y}$.*
- *Choose manually a good value for the regularization parameter λ_1 (a positive value).*
- *Then solve* (17) *using FISTA.*
- *Provide your codes with comments and include the reconstructed images in your report.*

To aid you in the preparation of the code, we provide a template to be modified according to your needs: `main_cs_template.m` *and* `main_cs_template.py` *where $\text{prox}_{\gamma g}$ is already given. Note since the MRI images are complex-valued, you need to take the magnitude of the image before plotting and also compute the PSNR values on the magnitude images.*

It is important to use a good regularization parameter to see the benefits of the estimator given in (17)*. Therefore we will do a parameter sweep for this exercise as well.*

EXERCISE 3.3: (4 POINTS) - *Using the same brain image given in Figure 4, solve* (17) *on a certain range of the regularization parameter values; e.g., on a range of 20 different values. Then plot the obtained PSNR values as a function of the regularization parameter. What are the maximum PSNR value that the regularization method achieves ? What are the good λ_1 value that give this PSNR value ? You are supposed to plot on a meaningful range where you can see the behavior of the PSNR as a function of the regularization parameter, preferably with the best parameter located near the middle of the x-axis.*

Remark 4. When performing a parameter sweep for the regularization method such as the one above, it is computationally beneficial to do a warm-start. This procedure consists in using as initial estimate the solution obtained for the previous regularization parameter value. If the two regularization parameter values are close to each other, with such a start the algorithm will converge in fewer iterations. Furthermore, FISTA usually converges faster for large values of the regularization parameter, so it is better to start from the largest value and use warm-start as we decrease the value of the regularization parameter.

We now turn our attention to the linear estimator $\hat{\mathbf{x}} = \boldsymbol{\Psi}^* \mathbf{P}_\Omega^T \mathbf{b}$ (7)*, where $\boldsymbol{\Psi}$ is the Fourier transform and $\mathbf{b} = \mathbf{P}_\Omega \boldsymbol{\Psi} \mathbf{x}^\natural$. The provided indices in Exercise 3.2 work well for nonlinear decoder* (17)*. In the next exercise you'll find a set of indices, Ω, that is optimal for the linear decoder assuming that you can use the whole image itself. But in practice, we don't know those indices a priori, so the performance will be expected to be worse. However if good trained indices are provided, the linear estimator, which is easy to implement, can also be used for reconstruction.*

On the other hand, when properly used, estimator given in (17) *can in general provide better error performance and visual quality in the reconstructions. This is due to fact that they exploit the structures in the images.*

EXERCISE 3.4: (12 POINTS) - *Comparison of different estimators.*

1. *Consider that you don't know anything about the signal and set Ω to be 20% of the pixel locations picked at random. Compare the PSNR and the reconstructions when using the linear decoder and FISTA.*

2. *Now, assuming that you're an oracle, find the best Ω for your image (for this you can use the function* `best_inds` *given to you) Compare the performance of the linear **and** FISTA reconstructions on (i) the best mask you have found, (ii) the mask given to you and (iii) the randomly sampled mask. Do so by plotting the masks and the corresponding reconstructed images with their PSNR values. Comment on the differences.*

3. *Time how long it takes to recover the image for the linear estimator and the nonlinear estimator defined in* (17) *using the brain image.*

4 Image in-painting

Image in-painting consists in reconstructing the missing parts of an image. By exploiting the structures such as sparsity over the wavelet coefficients, it is possible to in-paint images with a large portion of their pixels missing. In this part of the homework, we are going to study the convergence of different methods related to FISTA, as well as different restart strategies. We consider a subsampled image $\mathbf{b} = \mathbf{P}_\Omega \mathbf{x}$*, where* $\mathbf{P}_\Omega \in \mathbb{R}^{n \times p}$ *is again an operator that selects only few,* $n \ll p := m^2$*, pixels from the vectorized image* $\mathbf{x} \in \mathbb{R}^p$*. We can try to reconstruct the original image* $\mathbf{x}$ *by solving similar problems as before:*

$$\min_{\boldsymbol{\alpha} \in \mathbb{R}^p} \underbrace{\frac{1}{2}\|\mathbf{b} - \mathbf{P}_\Omega \mathbf{W}^T \boldsymbol{\alpha}\|_2^2}_{f(\boldsymbol{\alpha})} + \underbrace{\lambda_1 \|\boldsymbol{\alpha}\|_1}_{g(\boldsymbol{\alpha})}, \tag{18}$$

$$\min_{\mathbf{x} \in \mathbb{R}^p} \underbrace{(1/2)\|\mathbf{b} - \mathbf{P}_\Omega \mathbf{x}\|_2^2}_{f(\mathbf{x})} + \underbrace{\lambda_{TV} \|\mathbf{x}\|_{\mathrm{TV}}}_{g(\mathbf{x})}. \tag{19}$$

An example of image in-painting with this model is given in Figure 5.

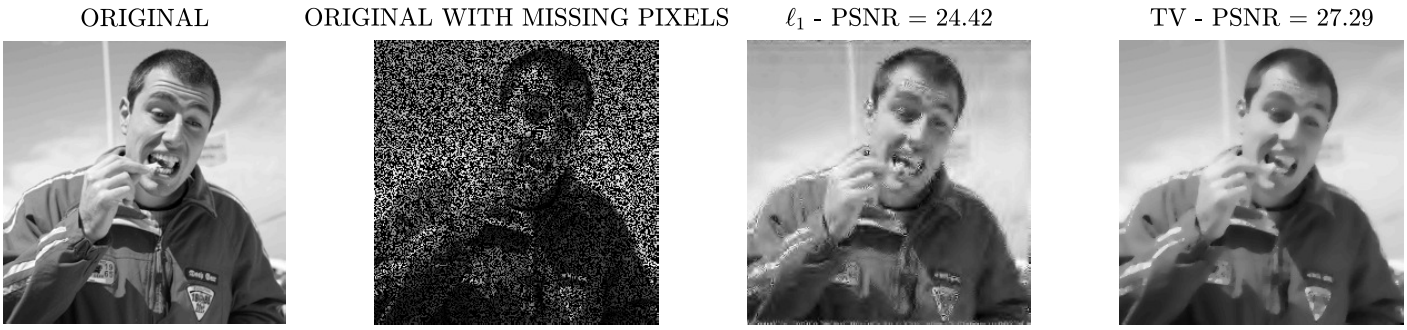

Figure 5: An example of image in-painting with the three models described in this homework. The regularization parameter has been set to 10 for all methods.

EXERCISE 4.1: (10 POINTS) - *Adapt the FISTA algorithm that you implemented in Exercise 3.2 to solve the problems* (18) *and* (19)*.*

1. *Take a* 1024×1024 *natural image, or better, a picture of you, and randomly and uniformly subsample* 40% *of its pixels (i.e. remove* 60% *of the pixels).*
2. *Perform a parameter sweep over the regularization parameters* λ_1 *and* λ_{TV} *on a meaningful range where you can see the behavior of the PSNR as a function of the regularization parameter.*
3. *Plot the obtained PSNR against the regularization parameter values.*
4. *Provide your codes with comments, and include the reconstructed images.*

EXERCISE 4.2: (15 POINTS) - *Convergence analysis:*

We will now perform convergence analysis, and study the convergence to the optimal solution $\mathbf{x}^*$ *of problem* (18) *as well as to the ground truth. We will again uniformly subsample* 40% *of the pixels of the image, and we will use* $\lambda = 10$*.*

1. *First of all, implement the following variations of FISTA:*
   (a) *ISTA: this is known as the iterative soft thresholding algorithm, where the proximal gradient descent is simply iterated (cf. Lecture 10).*
   (b) *FISTA with no restart*
   (c) *FISTA with fixed iteration restart*
   (d) *FISTA with gradient scheme restart. This procedure, described in greater depth in papers such as [2, 4], has the following restart criterion:*
   $$\langle y^k - x^{k+1}, x^{k+1} - x^k \rangle > 0$$
   The term $y^k - x^{k+1}$ *can be seen as a gradient , and* $x^{k+1} - x^k$ *is the descent direction of the momentum term. Overall this criterion resets the momentum of FISTA to* 0 *when it makes the algorithm go in a bad descent direction.*
2. *As the optimal solution* $\mathbf{x}^*$ *of the problem* (18) *and the corresponding optimal value of the function* $F^* := f(\mathbf{x}^*) + g(\mathbf{x}^*)$ *are not known a priori, they have to be estimated first. To do so, run FISTA with the non-monotonicity test as restart criterion for* 5000 *iterations with tolerance* 10^{-15}*. Then, set the minimum value of* F *as* F^**.*

3. *Study the convergence of ISTA, FISTA without restart, FISTA with fixed iteration restart (with restarts every* $\{25, 50, 100, 200\}$ *iteration) and FISTA with gradient scheme restart over* 2000 *iterations. Adapt the tolerance criterion to stop iterating when* $|F(\mathbf{x}^k) - F^*|/F^* < 10^{-15}$. *Plot a graph* $\log\left(|F(\mathbf{x}^k) - F^*|/F^*\right)$ *against k (cf. the examples in Lecture 10) and comment on the different rates and behaviors observed. HINT: You can use the function* `semilogy` *for the plot.*

4. *Replace then the* F^* *with* $F^\natural := F(\mathbf{x}^\natural)$ *and plot a graph* $\log\left(|F(\mathbf{x}^k) - F^\natural|/F^\natural\right)$ *against k using the same algorithms as in 3. for* 1000 *iterations. Comment on the differences.*

5. *Provide your codes with comments, as well as both convergence plots with discussion on the results.*

5 Guidelines for the preparation and the submission of the homework

Work on your own. Do not copy or distribute your codes to other students in the class. Do not reuse any other code related to this homework. Here are few warnings and suggestions for you to prepare and submit your homework.

- *This homework is due at 9:00AM, 10 December, 2018.*
- *Submit your work before the due date. Late submissions are not allowed and you will get* 0 *point from this homework if you submit it after the deadline.*
- *Questions of 0 point are for self study. You do not need to answer them in the report.*
- *Your final report should include detailed answers and it needs to be submitted in PDF format.*
- *The PDF file can be a scan or a photo. Make sure that it is eligible.*
- *The results of your simulations should be presented in the final report with clear explanation and comparison evaluation.*
- *We provide some MATLAB and Python scripts that you can use to implement the algorithms, but you can implement them from scratch using any other convenient programming tool (in this case, you should also write the codes to time your algorithm and to evaluate their efficiency by plotting necessary graphs).*
- *Even if you use the MATLAB and Python scripts that we provide, you are responsible for the entire code you submit. Apart from completing the missing parts in the scripts, you might need to change some written parts and parameters as well, depending on your implementations.*
- *The codes should be well-documented and should work properly. Make sure that your code runs without error. If the code you submit does not run, you will not be able to get any credits from the related exercises.*
- *Compress your codes and your final report into a single ZIP file, name it as* `ee556hw3_NameSurname.zip`*, and submit it through the moodle page of the course.*

References

[1] CHAMBOLLE, A. *An algorithm for total variation minimization and applications.* J. Math. Imaging Vis. 20 *(2004), 89–97.*

[2] GISELSSON, P., AND BOYD, S. *Monotonicity and restart in fast gradient methods. In* Decision and Control (CDC), 2014 IEEE 53rd Annual Conference on *(2014), IEEE, pp. 5058–5063.*

[3] MALLAT, S. A wavelet tour of signal processing. *Academic press, 1999.*

[4] O'DONOGHUE, B., AND CANDES, E. *Adaptive restart for accelerated gradient schemes.* Foundations of computational mathematics 15, *3 (2015), 715–732.*

[5] RUDIN, L., OSHER, S., AND FATEMI, E. *Nonlinear total variation based noise removal algorithms.* Physica D: Nonlinear Phenomena 60, *1-4 (1992), 259–268.*