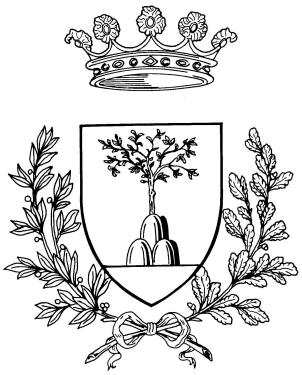
Università degli Studi di Ferrara

Facoltà di Scienze Matematiche, Fisiche e Naturali
Corso di Laurea in Matematica, indirizzo Matematica Applicata



Total Variation image denoising from Poisson data: Split Bregman and Alternating Extragradient methods

Laureando:
Davide Taviani

Relatore:
Chiar.ma Prof.ssa
Valeria Ruggiero

Anno Accademico 2010-2011

Contents

Nell'ambito multidisciplinare dell'elaborazione delle immagini, un problema di rilievo è costituito dalla individuazione delle tecniche numeriche per rimuovere il fenomeno chiamato rumore o *noise*, ossia per ricostruire immagini degradate da effetti a carattere aleatorio. Questo problema è anche noto con il nome di *denoising*.

Il tipo di *noise* che ha avuto più attenzione in letteratura è quello di lettura, cioè dovuto all'amplificazione, da parte di un sistema elettronico, del segnale in uscita dal sistema di sensori, che trasformano i segnali da analogici a digitali. Per questo rumore, di tipo additivo, il modello più accreditato è quello di una distribuzione Gaussiana.

Un altro tipo di *noise* è quello fotonico, che trae origine dalle fluttuazioni nel numero di fotoni che arrivano sul rivelatore, e che può essere descritto statisticamente attraverso una distribuzione multivariata di Poisson.

In alcuni casi, come ad esempio nella radiografia digitale, nella tomografia ad emissione, nella microscopia confocale a fluorescenza e nell'astronomia con telescopi ottici o a infrarossi, il rumore fotonico è prevalente su quello additivo. Questo fatto porta alla necessità di generalizzare l'approccio usato per eliminare il rumore Gaussiano al rumore Poissoniano, riconducendo il problema alla minimizzazione di una funzione convessa, fortemente non lineare, nel suo dominio.

Nel caso dei regolarizzatori che preservano i bordi, come per il funzionale di *Total Variation*, poiché la funzione è anche non differenziabile, viene introdotto un grado di ulteriore complessità nella ricerca di un metodo numerico efficiente per la risoluzione del problema.

Scopo della tesi è descrivere il problema di denoising di immagini degradate da rumore Poissoniano e analizzare dal punto di vista teorico e numerico i metodi che rappresentano lo stato dell'arte per regolarizzazione con la funzione *Total Variation*.

Nel primo capitolo si descrive il processo di formazione delle immagini, evidenziando l'origine delle varie perturbazioni che possono degradarle. Particolare attenzione si è data alla differenziazione fra le caratteristiche del rumore di tipo Gaussiano e quello di tipo Poissoniano. Si è inoltre mostrato come il problema di ricostruzione di immagini sia un problema inverso mal posto nel senso di Hadamard, e quindi potenzialmente di difficile risoluzione; per questo motivo, nell'ambito della teoria Bayesiana, è stato mostrato come l'approccio di massima verosimiglianza e le tecniche di regolarizzazione consentano di ricondurre il problema alla soluzione di un problema di minimo vincolato.

Nel secondo capitolo sono stati analizzati dal punto di vista teorico i metodi di tipo Bregman e Split Bregman, mettendo in evidenza gli aspetti implementativi in termini di complessità computazionale. Tali metodi possono essere ricondotti al metodo alle direzioni alternate dei moltiplicatori, ma possono essere interpretati anche come l'algoritmo di Douglas-Rachford, un particolare metodo *proximal point*, applicato alla formulazione duale del problema di minimo. Dai noti risultati teorici su questi metodi, è possibile ricavare la convergenza dei metodi PIDAL e PIDSplit+, due metodi Split Bregman appositamente progettati per ricostruire dati affetti da rumore Poissoniano con regolarizzazione data dalla

Total Variation.

Nel terzo capitolo viene esaminata la formulazione primale-duale del problema di minimo, che porta a risolvere un problema di sella per una funzione convessa-concava. Esprimendo il problema variazionale come una particolare disequazione variazionale monotona su un dominio esprimibile come prodotto cartesiano dei domini delle variabili primale e duale, si è considerata la classe dei metodi extragradiente, adattando tale approccio al problema in esame.

In particolare viene descritto il metodo AEM, in cui ogni passo esegue una serie di aggiornamenti successivi di tipo Gauss-Seidel delle variabili primale e duale mediante opportuni passi di gradiente proiettato.

Nel quarto capitolo vengono infine svolte alcune simulazioni numeriche, sia considerando problemi test simulati, ovvero problemi per i quali si ha a disposizione l'immagine priva di rumore, che problemi reali. Per la prima classe di questi problemi test sono stati studiati e confrontati il metodo AEM e il metodo PIDSplit+, sia in termini di accuratezza che di efficienza numerica. Particolare enfasi è stata data all'analisi della velocità di convergenza, sia per quanto riguarda le prime iterazioni, che rispetto al comportamento asintotico.

Per quanto riguarda i problemi di tipo reale, invece, è stato dapprima stimato il parametro di regolarizzazione, usando sia il principio di discrepanza, recentemente enunciato in [1], che analizzando il comportamento delle soluzioni ottenute.

Sulla base dei valori ottenuti per il parametro di regolarizzazione, sono stati confrontati i metodi PIDSplit+ e AEM, sperimentando un criterio di arresto automatico. L'analisi sperimentale ha messo in evidenza un'ottima efficienza del metodo PIDSplit+ per opportune scelte di un parametro del metodo, che deve essere stimato dall'utente. D'altro canto, il metodo AEM ha messo in evidenza un buon comportamento, sia dal punto di vista dell'efficienza che dell'accuratezza dei risultati ottenuti, senza avere la necessità di stimare alcun parametro, poiché la lunghezza del passo da cui dipende ogni iterazione è calcolata adattivamente.

In Appendice A, sono state incluse le definizioni di alcune matrici frequenti nei problemi di elaborazioni di immagini, come le matrici di Toepliz e quelle circolanti, mettendo in evidenza come le operazioni di base con queste matrici possano essere realizzate mediante la Trasformata Discreta di Fourier, efficientemente calcolabile con l'algoritmo Trasformata Veloce di Fourier. Inoltre, sono stati riportati in sintesi i principali risultati teorici sull'ottimizzazione vincolata (condizioni necessarie e sufficienti del primo e secondo ordine) e alcune importanti proprietà delle funzioni convesse e monotone.

Nell'Appendice B, infine, sono stati allegati i codici MATLAB usati.

Chapter 1

Image restoration

Image restoration is one of the most important topics in imaging science. To better understand that, this chapter will cover the principles behind image formation and errors that occur during this process.

1.1 The image acquisition

Images are acquired via an image system which consists in a optical part and a detector; they both have to be taken into account for an accurate modeling of the imaging process.

The optical system

The optical system can be summarized as a device which collects the light coming from an object to form a image in a plane, hence called the *image plane*.

Since most imaging systems are exactly, or at least approximately, isoplanatic, if the light emitted by the object is spatially incoherent, then the following integral describes the linear relationship between the intensity of the object $x(s)$ and that of the image obtained $\tilde{f}(s)$:

$$\tilde{f}(s) = \int H(s - s')\, x(s')\, \mathrm{d}s' \tag{1.1}$$

H is called the *impulse response*.

The expression can also be viewed as

$$\tilde{f}(s) = (H * x)(s)$$

where $*$ denotes the *convolution* operator.

In case of a point light source in s_0, usually expressed with a Dirac δ function centered in s_0, under the hypothesis that the overall energy is conserved, its integral is costant at 1.

Thus, we have that $x(s) = \delta(s - s_0)$ and (1.1) can be rewritten as:

$$\tilde{f}(s) = H(s - s_0)$$

Then the impulse response coincides with the image obtained in case of point light sources. For this reason H is also known as *point spread function* (PSF). The PSF has also a practical meaning, since it represents the error due to the optical system known as *blurring* (see Section (1.2.1)).

In general, we will suppose the PSF to satisfy:

1. $H(s) \geq 0$
2. $\int H(s)\,\mathrm{d}s = 1$

Detection

Along with the optical part, the image system is characterized by another fundamental device, the **detector**, which is located in the image plane and measures the incoming radiation, converting it to electrical signals.

Some example of such detectors are the CCD (*Charged Coupled Device*), often used in astronomy, APS (*Active Pixel Sensor*), which is the most common sensor found in cell phone cameras and DSLRs cameras, and APD (*Avalanche Photo-Diode*), one of the most important for fluorescence microscopy.

The detection process introduces the concept of **sampling**, which is a process performed on a uniform grid in the image plane: since it is impossible to store continuous informations, the image is stored as an array of pixels or voxels (depending on the dimension of the problem), and for this reason the equation (1.1) must be somehow discretized; thanks to the mapping provided by geometric optics, the image can be subdivided into pixels (or voxels), so that the convolution equation (1.1) is replaced by the matrix equation

$$\tilde{f} = Hx \tag{1.2}$$

The discrete convolution can assume different forms, depending on the boundary values: if we assume to have 0 in the boundaries, then the matrix H is a block-Toepliz with Toepliz blocks (BTTP) matrix[1], besides, if we assume the boundaries to be periodic (i.e. the first line is replicated as the $(n+1)$-th line, and similarly for every border), the matrix is a block-circulant with circulant blocks (BCCB) matrix[1].

The discrete PSF is normalized to 1: $H^T e = e$, where e denotes an array with all entries equal to 1.

[1] See Section A.1 for a brief overview of such particular matrix structures.

Constraints due to the physical properties of the object can be imposed: a natural constraint is $x_i \geq 0$, $\forall i$.

Furthermore, every entry of H is nonnegative and $He > 0$ (every row and every column of H has at least a nonzero element).

1.2 Errors in the image formation

In the digital signal obtained by the image system, some errors can come at play; in general, the ones which are due to the radiation emitted by the object x itself, the *blurring* and *noise*, are dominant.

1.2.1 Blurring

Blur is an error in the image which occurs during the acquisition, and, mathematically speaking, is connected to backward diffusion processes (such as the inversion of the heat equation), which are notoriously unstable.

In general, blur, along with its origin, can be divided into three main categories, each one with its own PSF:

- *optical blur*, also often called "out-of-focus blur", which is due to the deviation of the image plane from the focus of an optical lens;
- *mechanical blur*, or "motion blur", which arises from the rapid mechanical motion of either the target object or image devices during the image acquisition process;
- *medium-induced blur* which is due to the scattering or optical turbulence of the media through which light rays travel.

The last one affects heavily astronomy and microscopy: for example, spatiotemporal variation of the physical properties of the atmosphere (such as temperature and density) could result in a randomly fluctuating distributions of the index of refraction, causing "optical turbulence"; meanwhile, in microscopy, problems lie in the inhomogeneities in the refractive index of the specimen considered, which lead to a similar effect.

1.2.2 Noise

Noise can be roughly described as an *unwanted* component of the image which is intrinsic to any image system.

There are several kind of noise: some occur naturally, some are induced by sensors, and others are a result from various processes like quantization, transmission of the informations and speckles in coherent light situations.

If the noise is *additive*, a generic model for such process is:

$$f = \tilde{f} + n$$

where $\tilde{f}$ denotes the ideal image as in (1.1), n is the noise, and f the real, noise-affected, observation.

The various noises generally considered are random in nature and therefore their exact values are random variables, which are best described using probabilistic notions.

While there are several image noise models, two of them are the most common in literature: Gaussian and Poisson Noise.

Gaussian noise

Gaussian noise is probably the most frequently occurring noise and for this reason it has been extensively studied.

Gaussian noise retains the properties of Gaussian distribution, and the Central Limit Theorem (CLT) is certainly one of the most important: it states that, given a large number of independent (or nearly so) variables, which are individually negligible compared to the sum, then the distribution of the sum of such variables has a Gaussian distribution. The theorem holds even if the individual random variables do not have a Gaussian distribution themselves or even the same distribution at all.

For this property, Gaussian noise is widely used to model thermal noise, which is due to the thermal agitation of charge carriers, usally electrons, inside a conductor, since the vibration of each electron can be seen as a random variable that fulfills the hypotesis of the CLT.

Gaussian noise, in addition, can be seen as the limit behavior of other noises with a different distribution, such as the *photon noise* previously described.

The density function of univariate Gaussian noise, with mean μ and variance σ^2 is

$$p(x) = \frac{1}{\sqrt{2\pi\sigma^2}}\, e^{-\frac{(x-\mu)^2}{2\sigma^2}}$$

with $-\infty < x < \infty$.

Poisson noise

In all imaging processes where images are obtained by means of the count of particles (photons in general) arriving in the image domain, the noise (also known as *photon noise* or *shot noise*) can be modeled with a Poisson distribution, meaning that the probability of receiving k particles in a given interval of time T is given by:

$$p(k) = \frac{e^{-\lambda}\lambda^k}{k!} \qquad k = 0, 1, 2, ... \tag{1.3}$$

where λ, proportional to T, is the expected value of the occurrencies.

If N is the average number of photons collected, the *signal-to-noise ratio* is $\sqrt{N}$; this explains why photon noise becomes more important when the number of photons collected is small, e.g. in low light conditions.

This statistical model is appropriate to describe data acquired in fluorescence microscopy, emission tomography, optical/infrared astronomy, and so on; even if in all these fields the wavelength of the photons is different, the statistics of the data remains the same, while the image system varies. Namely, the H matrix in the equation (1.2) is a sparse matrix in tomography, and a convolution matrix in microscopy and in astronomy.

As a result of recent development and attention gained by medical application such as PET (*Positron Emission Tomography*) and SPECT (*Single Photon Emission Computerized Tomography*), major advances have been made and then transferred to other fields in which data are affected by Poisson noise.

A minor contribution to the noise can come from a phenomenon known as *background emission*: in astronomy, for example, this emission is due to the sky, while in fluorescence microscopy the auto-fluorescence of both the medium and the sample along with the reflections of the excitation light must be taken into account. This background emission is again a random variable with Poisson distribution and an estimate of its expected value b_i can be obtained by a preprocessing of the detected image. There are some cases in which the background value can be even larger than the signal itself; these images are called *background dominated* and are an important challenge in astronomy.

In general, when the object and the background emission are statistically independent (or it can be reasonably assumed), their overall contribution is a Poisson random variable with the expected value $(Hx + b)_i$.

1.3 Image restoration as an ill-posed inverse problem

If we consider the problem of image restoration to be independent to sensor effects and without inherent noise, we have the equation (1.1); in this case the problem is to determine the original object distribution x, given the recorded image $\tilde{f}$ and some knowledge about the PSF H.

A common method to do so is to employ operathor theory: given spaces of functions in which x and $\tilde{f}$ are defined, considering the operator T:

$$T\{x\} \to \tilde{f}$$

we have that

$$T\{x\} = \int H(s - s')x(s')\,\mathrm{d}s'$$

The problem of image restoration is then to find the inverse transformation T^{-1} such that

$$T^{-1}\{\tilde{f}\} \to x$$

For this reason, most of the problems found in image processing are **inverse problems**.

A problem stands a "good" chance of solution by computation, using a stable algorithm, if it is **well posed** and **well-conditioned**.

A problem is well-posed in the sense of Hadamard if:

- a solution exists
- the solution is unique
- the solution depends continuously on the data, in some reasonable topology

Problems for which T^{-1} does not exist, are said to be *singular*. Even if such inverse operator exists, is unique and depends continuously on the data, it may be *ill-conditioned*, i.e. a trivial perturbation in $\tilde{f}$ may result in nontrivial perturbations in x.

In particular, mathematically speaking, there exists an ϵ, arbitrarily small, such that:

$$T^{-1}\{\tilde{f} + \epsilon\} = x + \delta$$

where $\delta \gg \epsilon$. If this is the case, δ is not arbitrarily small and therefore not negligible.

Summarizing, an ill-posed problem is one in which the inverse transformation may not have a solution or inherent data perturbations, such as noise, may result in undesirable effects in the solution by inverse transformation (*ill-conditioning*).

The ill-posedness of image restoration can be proved by means of Riemann-Lebesgue lemma [2], which, applied to our case, states that:

$$\lim_{\alpha\to\infty} \int H(s - s')\sin(\alpha s')\,\mathrm{d}s' = 0$$

Then

$$\lim_{\alpha\to\infty} \int H(s - s')[x(s') + \sin(\alpha s')]\,\mathrm{d}s' = \int H(s - s')x(s')\,\mathrm{d}s' = \tilde{y} \qquad (1.4)$$

In other words, a sinusoid of infinite frequency can be added to the object distribution x and the result is identical to the image distribution $\tilde{y}$. Then the image restoration problem is an ill-posed problem.

A direct implication of (1.4) is that we choose a small value ϵ_1, there exist a value A such that

$$\int H(s-s')\sin(\alpha s')\,\mathrm{d}s' < \epsilon_1 \qquad \forall \alpha \geq A$$

Therefore

$$\int H(s-s')[x(s') + sin(\alpha s')]\,\mathrm{d}s' = \tilde{y} + \epsilon \qquad \forall \alpha \geq A,\ |\epsilon| < \epsilon_1$$

This shows that if an infinitesimally small value ϵ is chosen and added to the image distribution $\tilde{y}$, this cannot be separate, in the sense of image restoration, from an original object distribution that has an additional component of a sinusoid with frequency α. Since ϵ can be arbitrarily small, a trivial perturbation cannot be distinguished from a finite, nontrivial perturbation in the original object distribution. Then, also the solution of the inverse transformation is ill-conditioned.

1.4 Regularization

As seen before, image restoration is an ill-posed problem and the key to a successful solution of this kind of problems lies in its regularization: assuming the image is affected by noise with a known statistical distribution, we will adopt the Bayesian framework proposed by Geman and Geman [3], which consist in the use of additional *a priori* informations about both the clean signal $\tilde{y}$ and the noise, along with their statistical properties.

Then the computation of the *maximum a posteriori* estimate is reduced to the minimization of the negative-log of the posterior probability density.

1.4.1 Likelihood

Using the previous notation, let f denote the detected signal (an array of integer nonnegative numbers, which represent the number of photons detected) with elements f_i ($i \in S$, which will be implied from this moment on, can be either a single or multiple index, depending on the nature of the problem).

In case of Gaussian noise, f is the realization of a random variable F; its elements F_i are Gaussian random variables with mean $(Hx + b)_i$ and variance σ_i.

Assuming the random variables associated with each pixel are statistically independent, then the probability distribution of F, for given H, x and b, is:

$$p_F(f;x) = \prod_{i \in S} \frac{1}{\sqrt{2\pi\sigma_i^2}} e^{-\frac{(f_i - (Hx+b)_i)^2}{2\sigma_i^2}} \tag{1.5}$$

In case of Poisson noise, on the other side, f is the realization of a Poisson random variable f (with elements F_i) with the expected value $Hx + b$. Assuming again that the random variables associated with each pixel are statistically independent, then the probability distribution of F, for given H, x and b, is:

$$p_F(f;x) = \prod_{i \in S} \frac{e^{-(Hx+b)_i}(Hx+b)_i^{f_i}}{f_i!} \tag{1.6}$$

The *maximum likelihood* (ML) approach consists substantially in the addition of a function of x, the *likelihood*, defined by $L_f^F(x) = p_F(f;x)$, where f is the given detected image. Then a ML estimate is any maximizer of such function.

In case of Gaussian noise, if we take the negative logarithm, the equation (1.5) is reduced to:

$$\phi_0(x) = \sum_{i \in S} \left\{ \log \frac{1}{\sqrt{2\pi\sigma_i^2}} + \frac{(f_i - (Hx+b)_i)^2}{2\sigma_i^2} \right\}$$

Since $\log \frac{1}{\sqrt{2\pi\sigma_i^2}}$ is costant, the minimization of such equations is essentially the weighed least square approximation:

$$\min \frac{1}{2} \left|\left| \frac{f_i - (Hx+b)_i}{\sigma_i} \right|\right|_2^2 \tag{1.7}$$

Considering Poisson noise, however, taking the negative logarithm and using the Stirling formula ($\log f_i! \simeq -f_i + f_i \log f_i$), the equation (1.6) is reduced to:

$$\phi_0(x) = \sum_{i \in S} \left\{ f_i \log \frac{f_i}{(Hx+b)_i} + (Hx+b)_i - f_i \right\} \tag{1.8}$$

where we assume $f_i \log f_i = 0$ if $f_i = 0$.

This function is called the generalized Kullback-Leibler (KL) divergence (also known as Csiszàr I divergence) of x from f, and its minimization is reduced on the nonnegative orthant, since is well-known to be convex, nonnegative and coercive on the nonnegative orthant, so that minimizers exist and are global.

If the equation $Hx+b = y$ has a nonnegative solution x^*, this is also a minimizer of the KL function, since we have

$$\phi_0(x^*) = \sum_{i \in S} \left\{ f_i \log \frac{f_i}{f_i} + f_i - f_i \right\} = 0$$

In general, because of the extreme ill-conditioning of H, a solution could be difficult to compute. Furthermore, f is degraded by noise. Consequently, a minimizer of $\phi_0(x)$ may not provide a sensible solution of the image deblurring problem.

For this reason it is necessary to "regularize" the maximum likelihood approach by a suitable *prior*, which describes the known statistical properties of the unknown solution.

1.4.2 The Bayesian Approach

In the Bayesian approach, the unkown object x is also considered as a realization of a multi-valued random variable, and the *a priori* information is encoded into the given probability distribution $p_X(x)$ of this random variable, the so-called *prior*.

If the probability distribution (1.6) is viewed as a conditional probability of F for a given value of x, i.e. $p_F(f; x) = p_F(f|X = x) =: p_F(f|x)$, then the Bayes formula (hence the name) states:

$$p_X(x|f) = \frac{p_F(f|x)p_X(x)}{p_F(x)}$$

If in this equation we insert the detected value of f, we obtain the *posterior probability distribution* of X,

$$P_f^X(x) = L_f^F(x)\frac{p_X(x)}{p_F(f)}$$

As for the choice of the prior, in general it is assumed that X is either a Gibbs random field or a Markov random field (MRF). Thanks to Hammersley-Clifford Theorem, however, any MRF is equivalent to a suitable Gibbs random field, so we will focus on that.

The Gibbs random field is a random variable with a probability distribution given by

$$p_X(x) = \frac{1}{Z} e^{-\beta \phi_1(x)}$$

where $\phi_1(x)$ is a given function, usually called energy function or **penalty function**, and Z is a normalization constant.

By definition, the *maximum a posteriori* (MAP) estimate is any maximizer of the posterior probability distribution; if we take again the negative logarithm of this function, we can see that a MAP estimate is any x_β^* which minimizes

$$\phi(x) = \phi_0(x) + \beta\phi_1(x) \tag{1.9}$$

where β is the so-called *regularization parameter*.

Functions of the form of (1.9) have been used in several applications, with different form of the penalizing function $\phi_1(x)$.

$$\begin{aligned} &\text{Quadratic Penalization} && \phi_1(x) = \frac{||x||_2^2}{2} \\ &\text{Quadratic Laplacian} && \phi_1(x) = \frac{||\Delta x||_2^2}{2} \\ &\text{Total Variation (TV)} && \phi_1(x) = \sum_{i \in S} ||(\nabla x)_i||_2 \end{aligned} \tag{1.10}$$

where $(\nabla x)_i$ is the discrete approximation of the gradient at x_i.

Different criteria have been proposed for the choice of the regularization parameter β: in case of Gaussian noise, to solve the weighed least squares problem (1.7), a good principle is the so-called *Mozorov discrepancy principle*[4]. Considering Poisson noise, however, a possible way, proposed in [1], is to find the value of β such that

$$D_{KL}(Hx_\beta^* + b, f) = \frac{n}{2} \tag{1.11}$$

where D_{KL} is the same as (1.8), and n is the cardinality of S.

1.5 Formulation of the problem

In this work, we will focus on the image denoising from data corrupted by Poisson noise; the PSF is then the identity matrix ($H = I$) and, if we suppose that there is no *background emission*, the KL divergence (1.8) can be rewritten as:

$$\phi_0(x) = \sum_{i \in S} \left\{ f_i \log \frac{f_i}{x_i} + x_i - f_i \right\}$$

A minimizer x^* for such functional is $x^* = f$, which means that the solution is the detected image itself. Since we know that f is a noisy image, f cannot be a feasible solution for our problem; we overcome this with a penalty function $\phi_1(x)$ (see (1.9)).

The domain of the resulting functional $\phi(x) = \phi_0(x) + \beta\phi_1(x)$ is

$$X = \begin{cases} x \geq \eta > 0 & f_i > 0 \\ x \geq 0 & f_i = 0 \end{cases} \tag{1.12}$$

where

$$\eta = \min\{f_i | f_i > 0\}$$

Such function $\phi(x)$ is convex and coercive and thus the solution of the problem

$$\min_{x \in X} \phi(x) \tag{1.13}$$

is unique[2].

The gradient and Hessian of the ϕ_0 are given by:

$$\nabla \phi_0(x) = 1 - \frac{f}{x}$$
$$\nabla^2 \phi_0(x) = diag\left(\frac{f}{x^2}\right)$$

where the quotient is defined in the Hadamard sense, i.e. pixel by pixel (or voxel by voxel).

We will consider $\phi_1(x)$ as the TV functional as in (1.10). It is important to note that such function is not always differentiable. Such TV functional was has been introduced in [5]. The function (1.13), with $\phi_0(x) = \frac{1}{2}||x - y||_2^2$ is also known as ROF model.

In the following chapters, for the TV functional we will use expression

$$\phi_1(x) = \sum_{i=1}^{n} ||A_i x||_2 \tag{1.14}$$

where $n = s \times t$ is the number of pixels of the image, and A_i (for $i = 1, ..., n$) is a $2 \times n$ matrix with null entries except for

$$a_{1,i} = -1 \qquad a_{1,i+1} = 1$$
$$a_{2,i} = -1 \qquad a_{1,i+t+1} = 1$$

where the pixels in the image are ordered column-wise.

[2]See Section A.2.3 for a brief overview on the properties of convex functionals.

In addition, for further reference, we consider the matrix A as the $2n \times n$ matrix defined by

$$A = \begin{pmatrix} A_1 \\ \vdots \\ A_n \end{pmatrix} \tag{1.15}$$

Chapter 2

Bregman Iterations and Split Bregman methods

2.1 Bregman Iterations

In this section, we will introduce the Bregman Iteration and show that it can be used to solve a wide variety of constrained optimization problems.

Then we will introduce the "Split Bregman methods", which we will apply to TV functional for image denoising.

2.1.1 Subgradient and subdifferential

Definition 2.1. *Let $E : U \to \mathbb{R}$ be a real-valued function (not necessarily convex) defined on a convex open set in the Euclidean space $\mathbb{R}^n$: a vector v in that space is called a* ***subgradient*** *[6] at a point $x_0 \in U$ if for any $x \in U$ one has:*

$$E(x) - E(x_0) \geq v^T(x - x_0)$$

The set of all subgradients at x_0 is called the **subdifferential** at x_0 and is denoted $\partial E(x_0)$. The subdifferential is always a convex closed set and if f is convex and finite near x_0, it is nonempty.

The subgradient gives an affine global underestimator of $E(x)$. If $E(x)$ is convex, it has at least one subgradient at every internal point of the domain. Furthermore, if $E(x)$ is convex and differentiable, the gradient vector $\nabla E(x_0)$ is a subgradient of $E(x)$ at x_0.

Theorem 2.1. *Let $E : U \to \mathbb{R}$ be a convex function, $U \subset R^n$.*

$x_0 \in U$ is a global minimum $\iff 0 \in \partial E(x_0)$

Proof.

$$\begin{aligned} x_0 \text{ is a global minimum} &\iff E(x) \geq E(x_0) \qquad \forall x \in U \\ &\iff E(x) - E(x_0) \geq 0 \\ &\iff E(x) - E(x_0) \geq 0^T (x - x_0) \\ &\iff 0 \in \partial E(x_0) \end{aligned}$$

$\Diamond$

2.1.2 Bregman distance

Definition 2.2. *The Bregman distance [7] associated with a convex function E at the point v is*

$$D_E^p(u, v) = E(u) - E(v) - p^T(u - v) \tag{2.1}$$

where p is the subgradient of E at v.

If $E(u)$ is strictly convex, $D_E^p(u, v)$ is also strictly convex in u for each v, and as a consequence $D_E^p(u, v) = 0$ if and only if $u = v$.

However, even for a continuously differentiable and strictly convex function, the Bregman distance is not a distance in the usual sense, since in general $D_E^p(u, v) \neq D_E^p(v, u)$, and the triangle inequality does not hold.

It is, however, a measure of closeness in the sense that $D_E^p(u, v) \geq 0$, and $D_E^p(u, v) \geq D_E^p(w, v)$ for w on the line segment between u and v.

2.2 Bregman iteration for unconstrained minimization problem

Let us consider two non-negative energy functionals, E and H. We assume that H is differentiable and such that

$$\min_{x \in \mathbb{R}^n} H(x) = 0$$

and E is not everywhere differentiable.

We consider the following problem:

$$\min_x \{E(x) + \lambda H(x)\} \tag{2.2}$$

where λ is a positive number.

We can modify this problem, iteratively solving:

$$\begin{aligned} x^{(k+1)} &= \underset{x}{\operatorname{argmin}} \left\{ D_E^p \left(x, x^{(k)} \right) + \lambda H(x) \right\} \\ &= \underset{x}{\operatorname{argmin}} \left\{ E(x) - {p^{(k)}}^T \left(x - x^{(k)} \right) + \lambda H(x) \right\} \end{aligned} \tag{2.3}$$

We have $0 \in \partial \left(\left(D_E^p \left(x, x^{(k)} \right) + \lambda H \right)(x^{(k+1)}) \right)$; if $p^{(k+1)} \in \partial E \left(x^{(k+1)} \right)$, we have that

$$p^{(k+1)} = p^{(k)} - \lambda \nabla H \left(x^{(k+1)} \right) \tag{2.4}$$

To simplify, if we denote $q^{(k+1)} = \lambda \nabla H \left(x^{(k+1)} \right)$, we have $p^{(k+1)} + q^{(k+1)} = p^{(k)}$. Then, if we put $p^{(0)} = 0$, we have:

$$\begin{aligned} p^{(1)} &= p^{(0)} - q^{(1)} = -q^{(1)} \\ p^{(2)} &= p^{(1)} - q^{(2)} = -q^{(1)} - q^{(2)} \\ &\vdots \\ p^{(k+1)} &= -\sum_{j=1}^{k+1} q^{(j)} \end{aligned} \tag{2.5}$$

In [7] the authors analyze the convergence of this Bregman iterative scheme, and it is shown that under fairly weak assumption on E e H, we have that

$$\lim_{k \to \infty} H \left(x^{(k)} \right) = 0 \tag{2.6}$$

Theorem 2.2. *Assume that E and H are convex functionals, and that H is differentiable. We also assume that solutions to the sub-problems in (2.3) exist. We then have:*

1. *$H \left(x^{(k+1)} \right) \leq H \left(x^{(k)} \right)$ for every k.*
2. *if x^* is a minimizer of $H(x)$ such that $E(x^*) < \infty$, then*

$$H \left(x^{(k)} \right) \leq H(x^*) + \frac{E(x^*)}{\lambda k}$$

and in particular $\{x^{(k)}\}$ is a minimizing sequence.

This sequence has converging subsequences to a minimizer x^* of $H(x)$. If such minimizer is unique, $x^{(k)} \to x^*$ as $k \to \infty$.

Proof. See [7], Proposition 3.2 and Theorem 3.3. ◊

Note that x^* is not a solution of problem (2.2).

2.3 Bregman iteration for constrained minimization problem

We assume that we have to solve the following problem

$$\min_x E(x) \qquad \text{such that} \qquad Ax = b \tag{2.7}$$

We put $H(x) = Ax - b$, for some linear operator A and vector b. We want to apply the formula (2.2), so we make this into an uncostrained problem, using a quadratic penalty function, thus having

$$\min_x \left\{ E(x) + \frac{\lambda}{2}||Ax - b||_2^2 \right\} \tag{2.8}$$

The Bregman iteration for this problem is

$$\begin{aligned} x^{(k+1)} &= \operatorname*{argmin}_x \left\{ D_E^p \left(x, x^{(k)} \right) + \frac{\lambda}{2}||Ax - b||_2^2 \right\} \\ &= \operatorname*{argmin}_x \left\{ E(x) - {p^{(k)}}^T \left(x - x^{(k)} \right) + \frac{\lambda}{2}||Ax - b||_2^2 \right\} \end{aligned} \tag{2.9}$$

$$p^{(k+1)} = p^{(k)} - \lambda A^T (Ax^{(k+1)} - b) \tag{2.10}$$

In this case, since $H(x) = Ax - b$, H is linear with respect to x, and we have

$$q^{(k)} = \lambda \nabla H \left(x^{(k)} \right) = \lambda A^T \left(Ax^{(k)} - b \right) \tag{2.11}$$

From (2.5) and (2.11), we have $p^{(k+1)} \in \text{ range}(A^T)$.
If $p^{(0)} = 0, x^{(0)} = 0$ and we put $p^{(k)} = \lambda A^T \left(b^{(k)} - b \right)$, with $b^{(0)} = b$, we have

$$p^{(k+1)} = p^{(k)} - \lambda A^T (Ax^{(k+1)} - b)$$

$$\lambda A^T (b^{(k+1)} - b) = \lambda A^T \left(b^{(k)} - b \right) - \lambda A^T (Ax^{(k+1)} - b)$$

$$\lambda A^T b^{(k+1)} - \lambda A^T b = \lambda A^T b^{(k)} - \lambda A^T b - \lambda A^T A x^{(k+1)} + \lambda A^T b$$

$$\lambda A^T b^{(k+1)} = \lambda A^T b^{(k)} - \lambda A^T A x^{(k+1)} + \lambda A^T b$$

$$\lambda A^T b^{(k+1)} = \lambda A^T \left(b^{(k)} - Ax^{(k+1)} + b \right)$$

Then

$$b^{(k+1)} = b^{(k)} + b - Ax^{(k+1)} \tag{2.12}$$

In (2.9) we have:

$$\begin{aligned} -p^{(k)^T}\left(x - x^{(k)}\right) + \frac{\lambda}{2}||Ax-b||_2^2 &= p^{(k)^T}\left(x^{(k)} - x\right) + \frac{\lambda}{2}||Ax-b||_2^2 \\ &= \lambda\left(b^{(k)} - b\right)^T\left(Ax^{(k)} - Ax\right) + \frac{\lambda}{2}||Ax-b||_2^2 \end{aligned}$$

Moving on to the gradient we have:

$$\begin{aligned} \nabla\left(\lambda\left(b^{(k)} - b\right)^T\left(Ax^{(k)} - Ax\right) + \frac{\lambda}{2}||Ax-b||_2^2\right) &= -\lambda A^T\left(b^{(k)} - b\right) + 2\frac{\lambda}{2}A^T(Ax-b) \\ &= \lambda A^T\left(b - b^{(k)}\right) + \lambda A^T(Ax-b) \\ &= \lambda A^T\left(Ax - b + b - b^{(k)}\right) \\ &= \lambda A^T\left(Ax - b^{(k)}\right) \\ &= \nabla\left(\frac{\lambda}{2}\left|\left|Ax - b^{(k)}\right|\right|_2^2\right) \end{aligned}$$

Then the Bregman iteration can be written as

$$\begin{aligned} x^{(k+1)} &= \underset{x}{\operatorname{argmin}}\left\{E(x) + \frac{\lambda}{2}\left|\left|Ax - b^{(k)}\right|\right|_2^2\right\} \\ b^{(k+1)} &= b^{(k)} + b - Ax^{(k+1)} \end{aligned} \tag{2.13}$$

Since $H(x) = Ax - b$, we have that (2.6) is now

$$\lim_{k\to\infty} Ax^{(k)} = b \tag{2.14}$$

where the convergence is in the 2-norm sense.

Theorem 2.3. *Let $E : \mathbb{R}^n \to \mathbb{R}$ be convex; let $A : \mathbb{R}^n \to \mathbb{R}^m$ be linear. Consider the algorithm* (2.13) *and suppose that some iterate x^* satisfies $Ax^* = b$.*

Then x^* is a solution to the original constrained problem (2.7)*.*

Proof. Let x^* and b be such that $Ax^* = b$ and

$$x^* = \underset{x}{\operatorname{argmin}} \left\{ E(x) + \frac{\lambda}{2}||Ax - b||_2^2 \right\} \tag{2.15}$$

Let $\hat{x}$ be a true solution to (2.7). Then $Ax^* = b = A\hat{x}$, which implies that

$$||Ax^* - b||_2^2 = ||A\hat{x} - b^*||_2^2 \tag{2.16}$$

Because x^* satisfies (2.15), we have

$$E(x^*) + \frac{\lambda}{2}||Ax^* - b||_2^2 \leq E(\hat{x}) + \frac{\lambda}{2}||A\hat{x} - b||_2^2 \tag{2.17}$$

Finally, note that (2.16)-(2.17)) together imply:

$$E(x^*) \leq E(\hat{x})$$

Because $\hat{x}$ satisfies the original optimization problem, this inequality can be sharpened to an equality, showing that x^* solves (2.7). $\Diamond$

This theorem shows that, provided that the algorithm converges in the sense of (2.14), the iterates $x^{(k)}$ will get arbitrarily close to a solution of the original constrained problem. Furthermore, the proof does not use the linearity of A.

There are some advantages with these Bregman iteration techniques over traditional penalty functions: first of all, Bregman iterations converge quickly when applied to certain types of functions, especially for problems where E contains an L1- regularization term. The second advantage is that the value of λ in (2.2) is constant, and thus we can choose a good value for it, which minimizes the condition number of the sub-problems, also avoiding numerical instabilities that occur as $\lambda \to \infty$, that arise when using penalty methods.

2.4 Split Bregman methods

We will now solve

$$\min_x \{|\Phi(x)| + H(x)\} \tag{2.18}$$

where H and $|\Phi|$ are convex functionals, and Φ is differentiable. The key to our methods is to decouple the L1 and L2 portion of the energy in (2.18). We then consider the problem

$$\min_{x,d} \{|d| + H(x)\} \qquad \text{such that} \qquad d = \Phi(x) \tag{2.19}$$

which is clearly equivalent to the previous one. Then, we proceed to convert (2.19) to an unconstrained problem

$$\min_{x,d} \left\{ |d| + H(x) + \frac{\lambda}{2} ||d - \Phi(x)||_2^2 \right\} \tag{2.20}$$

If we put $E(x,d) = |d| + H(x)$ and we define $A(x,d) = d - \Phi(x)$ we can see that we have to solve the same problem as in Section 2.3.

Then, the Bregman iteration is:

$$\begin{aligned} \left(x^{(k+1)}, d^{(k+1)}\right) &= \underset{x,d}{\operatorname{argmin}} \left\{ D_E^p \left(x, x^{(k)}, d, d^{(k)}\right) + \frac{\lambda}{2} ||d - \Phi(x)||_2^2 \right\} \\ &= \underset{x,d}{\operatorname{argmin}} \left\{ E(x,d) - {p_x^{(k)}}^T \left(x - x^{(k)}\right) + \right. \\ &\quad \left. + {p_d^{(k)}}^T \left(x - x^{(k)}\right) + \frac{\lambda}{2} ||d - \Phi(x)||_2^2 \right\} \end{aligned} \tag{2.21}$$

$$p_x^{(k+1)} = p_x^{(k)} - \lambda \left(\nabla \Phi \left(x^{(k+1)}\right)\right)^T \left(\Phi \left(x^{(k+1)}\right) - d^{(k+1)}\right)$$

$$p_d^{(k+1)} = p_d^{(k)} - \lambda \left(d^{(k+1)} - \Phi \left(x^{(k+1)}\right)\right)$$

After having applied the simplification (2.13), we have the Bregman iteration:

$$\left(x^{(k+1)}, d^{(k+1)}\right) = \underset{x,d}{\operatorname{argmin}} \left\{ |d| + H(x) + \frac{\lambda}{2} \left|\left| d - \Phi(x) - b^{(k)} \right|\right|_2^2 \right\} \tag{2.22}$$

$$b^{(k+1)} = b^{(k)} + \left(\Phi \left(x^{(k+1)}\right) - d^{(k+1)}\right) \tag{2.23}$$

In this case we put

$$\begin{aligned} p_x^{(k+1)} &= \lambda \nabla \Phi \left(x^{(k+1)}\right)^T b^{(k+1)} \\ p_d^{(k+1)} &= \lambda b^{(k+1)} \end{aligned}$$

In order to implement this algorithm, we must be able to solve the problem (2.22); having split the L1 and L2 components of this functional, we can perform

this minimization efficiently, by iteratively minimizing with respect to x and d separately:

$$\text{Step 1:} \qquad x^{(k+1)} = \underset{x}{\operatorname{argmin}} \left\{ H(x) + \frac{\lambda}{2} \left|\left| d^{(k)} - \Phi(x) - b^{(k)} \right|\right|_2^2 \right\} \tag{2.24}$$

$$\text{Step 2:} \qquad d^{(k+1)} = \underset{d}{\operatorname{argmin}} \left\{ |d| + \frac{\lambda}{2} \left|\left| d - \Phi\left(x^{(k+1)}\right) - b^{(k)} \right|\right|_2^2 \right\} \tag{2.25}$$

The scheme (2.24-2.25-2.23) is known as **Split Bregman iteration** [8].

Having decoupled x from the L1 portion of the problem, Step 1 is now differentiable. We can proceed by explicitly computing the optimal value of d using the shrinkage operator:

$$d_j^{(k+1)} = shrink\left(\Phi(x)_j + b_j^{(k)}, \frac{1}{\lambda}\right)$$

where the following operator solves $\underset{d}{\operatorname{argmin}} \left\{ |d| + \frac{1}{2\gamma}||d - z||^2 \right\}$

$$d = shrink(x, \gamma) = \frac{z}{|z|} max(|z| - \gamma, 0)$$

2.4.1 Implementation of the Split Bregman iteration

Algorithm 1 Split Bregman iteration

while $\left|\left|x^{(k)} - x^{(k+1)}\right|\right|_2 > tol$ **do**

$x^{(k+1)} = \underset{x}{\operatorname{argmin}} \left\{ H(x) + \frac{\lambda}{2} \left|\left| d^{(k)} - \Phi(x) - b^{(k)} \right|\right|_2^2 \right\}$

$d^{(k+1)} = \underset{d}{\operatorname{argmin}} \left\{ |d| + \frac{\lambda}{2} \left|\left| d - \Phi\left(x^{(k+1)}\right) - b^{(k)} \right|\right|_2^2 \right\}$

$b^{(k+1)} = b^{(k)} + \left(\Phi\left(x^{(k+1)}\right) - d^{(k+1)}\right)$

end while

Examining the Theorem 2.3, it is easy to observe that any fixed point of the split Bregman algorithm is a minimizer of the original problem (2.19), even if the subproblem (2.24) is inexactly solved.

Let (x^*, d^*, b^*) be a fixed point of (2.22-2.23). The fixed point satisfies $b^* = b^* + \Phi(x^*) - d^*$, which implies that $d^* = \Phi(x^*)$. This result, combined with (2.22), satisfies the condition of theorem 2.3, showing that (x^*, d^*) is a solution of the constrained problem (2.19).

In [9] and [10] the Bregman iteration is traced back to the *augmented Lagrangian algorithm* discussed by Hestenes in [11], and Powell in [12], with the only difference that in the Bregman iteration the sequence $\{b^{(k)}\}$ is scaled by a factor

λ. The convergence properties of the augmented Lagriangan algorithm are well known in literature, then the convergence of the Bregman iteration can be derived also from that of the augmented Lagrangian algorithm, see [13].

In [9] and [10] it is furthermore observed that the idea to minimize alternatingly with respect to the variables was presented also for the augmented Lagrangian methods in [14] [15].

The resulting algorithm is called the *alternating direction method of multipliers* [16], which is equivalent to the alternating split Bregman algorithm.

If we consider a dual formulation of the problem (2.18), it is possible to observe that the Bregman iteration is equivalent to the *proximal point algorithm* of Rockafellar applied to the dual problem. Furthermore, the split Bregman methods coincides with the *Douglas Rachford splitting method* applied to the dual problem, see [17].

Then the convergence properties of the Bregman iteration and of the split Bregman method can be deduced from that of the proximal point algorithm and of the Douglas Rachford splitting method respectively.

2.5 TV denoising with split Bregman iteration

TV denoising is considered to be one of the best denoising models for both Gaussian and Poisson noise, but also one of the hardest to compute.

In this section, we will show how the Split Bregman thechnique can be used to solve this kind of problems in a simple and extremely efficient way.

2.5.1 Gaussian noise

Anisotropic model

We begin by addressing this anisotropic problem:

$$\min_x \left\{ |D_i x| + |D_j x| + \frac{\mu}{2}||x - f||_2^2 \right\} \tag{2.26}$$

where:

$D_i x$ is the column vector of the forward difference $x_{i+1,j} - x_{i,j}$
$D_j x$ is the column vector of the forward difference $x_{i,j+1} - x_{i,j}$

With the notations of Chapter 1, D_i and D_j are matrices such that

$$PA = \begin{pmatrix} D_i \\ D_j \end{pmatrix}$$

where P is a $2n \times 2n$ permutation matrix that reorders the rows of A (1.15) by considering the odd rows first, and then the even rows.

To apply Bregman splitting, we first replace $D_i x$ by d_i e $D_j x$ by d_j. Thus we have the constrained problem

$$\min_{x,d_i,d_j} \left\{ |d_i| + |d_j| + \frac{\mu}{2}||x - f||_2^2 \right\} \qquad \text{such that} \quad d_i = D_i x \quad \text{and} \quad d_j = D_j x$$

To weakly enforce the constraints, we add penalty function terms:

$$\min_{x,d_i,d_j} \left\{ |d_i| + |d_j| + \frac{\mu}{2}||x - f||_2^2 + \frac{\lambda}{2}||d_i - D_i x||_2^2 + \frac{\lambda}{2}||d_j - D_j x||_2^2 \right\}$$

Finally, we strictly enforce the constraints by applying the Bregman iteration (2.22):

$$\min_{x,d_i,d_j} \left\{ |d_i| + |d_j| + \frac{\mu}{2}||x - f||_2^2 + \frac{\lambda}{2}\left|\left|d_i - D_i x - b_i^{(k)}\right|\right|_2^2 + \frac{\lambda}{2}||d_j - D_j x - b_j^{(k)}||_2^2 \right\}$$

where the proper values of $b_i^{(k)}$ and $b_j^{(k)}$ are chosen through Bregman Iteration.

The resulting scheme is the following [8]:

$$x^{(k+1)} = \underset{x}{\operatorname{argmin}} \left\{ \frac{\mu}{2}||x - f||_2^2|| + \frac{\lambda}{2}\left|\left|d_i^{(k)} - D_i x - b_i^{(k)}\right|\right|_2^2 + \frac{\lambda}{2}\left|\left|d_j^{(k)} - D_j x - b_j^{(k)}\right|\right|_2^2 \right\}$$

$$d_i^{(k+1)} = \underset{d_i}{\operatorname{argmin}} \left\{ |d_i| + \frac{\lambda}{2}\left|\left|d_i - D_i x^{(k+1)} - b_i^{(k)}\right|\right|_2^2 \right\}$$

$$d_j^{(k+1)} = \underset{d_j}{\operatorname{argmin}} \left\{ |d_j| + \frac{\lambda}{2}\left|\left|d_j - D_j x^{(k+1)} - b_j^{(k)}\right|\right|_2^2 \right\}$$

$$b_i^{(k+1)} = b_i^{(k)} + \left(D_i x^{(k+1)} - d_i^{(k+1)} \right)$$

$$b_j^{(k+1)} = b_j^{(k)} + \left(D_j x^{(k+1)} - d_j^{(k+1)} \right)$$

The first subproblem has the following optimality condition:

$$(\mu I - \lambda \Delta) x^{(k+1)} = \mu f + \lambda D_i^T \left(d_i^{(k)} - b_i^{(k)} \right) + \lambda D_i^T \left(d_j^{(k)} - b_j^{(k)} \right) \qquad (2.27)$$

where $\Delta = -D_i^T D_i - D_j^T D_j$.

In order to achieve optimal efficiency, we wish to use a fast iterative algorithm to get approximate solutions to this system. Because the system is strictly diagonally dominant, the most natural choice is the Gauss-Seidel method, which, in this case, can be written component-wise as $x_{r,s}^{(k+1)} = G_{r,s}^{(k)}$, where

$$G_{r,s}^{(k)} = \frac{\lambda}{\mu + 4\lambda}\Big(x_{r+1,s} + x_{r-1,s} + x_{r,s+1} + x_{r,s-1} + d_{i,r-1,s} - d_{i,r,s} + d_{j,r,s-1}^{(k)} + \\ - \; d_{j,r,s}^{(k)} - b_{i,r-1,s}^{(k)} + b_{i,r,s}^{(k)} - b_{j,r,s-1}^{(k)} + b_{j,r,s}^{(k)}\Big) + \frac{\mu}{\mu + 4\lambda} f_{r,s}$$

In this case the value of $x_{\cdot,\cdot}$ are at the step k or $k+1$ according to the order of solution.

Then, using this solver, we have the following algorithm:

Algorithm 2 Split Bregman method for Anisotropic TV

$u^{(0)} = f$

$d_i^{(0)} = d_j^{(0)} = b_i^{(0)} = b_j^{(0)} = 0$

while $\left|\left|x^{(k)} - x^{(k-1)}\right|\right|_2 > tol$ **do**

$x^{(k+1)} = G^{(k)}$

$d_i^{(k+1)} = shrink\left(D_i x^{(k+1)} + b_i^{(k)}, \frac{1}{\lambda}\right)$

$d_j^{(k+1)} = shrink\left(D_j x^{(k+1)} + b_j^{(k)}, \frac{1}{\lambda}\right)$

$b_i^{(k+1)} = b_i^{(k)} + \left(D_i x^{(k+1)} - d_i^{(k+1)}\right)$

$b_j^{(k+1)} = b_j^{(k)} + \left(D_j x^{(k+1)} - d_j^{(k+1)}\right)$

end while

The split Bregman method has the advantage that before the convergence is reached, the intermediate images are smooth; most of the image noise is eliminated during the first 10 iterations.

Isotropic model

The split Bregman technique can be used also with the isotropic TV model with Gaussian noise: in this case we wish to solve

$$\min_x \left\{ \sum_r \sqrt{(D_i x)_r^2 + (D_j x)_r^2} + \frac{\mu}{2} ||x - f||_2^2 \right\}$$

Like we did before, we split the L1 and L2 components by setting $D_i x = d_i$ and $D_j x = d_j$.

The split Bregman formulation then becomes:

$$\min_{x,d_i,d_j} \left\{ ||(d_i, d_j)||_2 + \frac{\mu}{2}||x - f||_2^2 \right\} \qquad \text{such that} \quad d_i = D_i x \quad \text{and} \quad d_j = D_j x$$

where

$$||(d_i, d_j)||_2 = \sum_{r,s} \sqrt{d_{i,r,s}^2 + d_{j,r,s}^2}$$

Note that the d_i and d_j, unlike in the anisotropic case, do not decouple. This changes the way in which these variables must be treated.

Enforcing the constraints as we did in the anisotropic case, adding penalty function terms and applying the Bregman iteration, we have:

$$\min_{d_i,d_j} \left\{ ||(d_i, d_j)||_2 + \frac{\mu}{2}||x - f||_2^2 \frac{\lambda}{2}||d_i - D_i x - b_i||_2^2 + \frac{\lambda}{2}||d_j - D_j x - b_j||_2^2 \right\}$$

The resulting scheme is the following:

$$x^{(k+1)} = \underset{x}{\operatorname{argmin}} \left\{ \frac{\mu}{2}||x - f||_2^2 + \frac{\lambda}{2} \left|\left| d_i^{(k)} - D_i x - b_i^{(k)} \right|\right|_2^2 + \right.$$
$$\left. + \frac{\lambda}{2} \left|\left| d_j^{(k)} - D_j x - b_j^{(k)} \right|\right|_2^2 \right\}$$

$$\left(d_i^{(k+1)}, d_j^{(k+1)}\right) = \underset{d_i,d_j}{\operatorname{argmin}} \left\{ ||(d_i, d_j)||_2 + \frac{\lambda}{2} \left|\left| d_i - D_i x^{(k)} - b_i^{(k)} \right|\right|_2^2 + \right.$$
$$\left. + \frac{\lambda}{2} \left|\left| d_j - D_j x^{(k)} - b_j^{(k)} \right|\right|_2^2 \right\}$$

$$b_i^{(k+1)} = b_i^{(k)} + \left(D_i x^{(k+1)} - d_i^{(k+1)} \right)$$

$$b_j^{(k+1)} = b_j^{(k)} + \left(D_j x^{(k+1)} - d_j^{(k+1)} \right)$$

Despite the fact that d_i and d_j do not decouple as in the anisotropic case, we can still explicitly solve the minimization problem using a generalized shrinkage formula:

$$d_i^{(k+1)} = \max\left(s^{(k)} - \frac{1}{\lambda}, 0\right) \frac{D_i x^{(k)} + b_i^{(k)}}{s^{(k)}}$$
$$d_j^{(k+1)} = \max\left(s^{(k)} - \frac{1}{\lambda}, 0\right) \frac{D_j x^{(k)} + b_j^{(k)}}{s^{(k)}}$$

where

$$s^{(k)} = \sqrt{\left(D_i x^{(k)} + b_i^{(k)}\right)^2 + \left(D_j x^{(k)} + b_j^{(k)}\right)^2} \tag{2.28}$$

If we apply once again the Bregman iteration to this problem, we get the minimization algorithm for the isotropic TV functional:

Algorithm 3 Split Bregman method for Isotropic TV

$x^{(0)} = f$

$d_i^{(0)} = d_j^{(0)} = b_i^{(0)} = b_j^{(0)} = 0$

while $\left|\left|x^{(k)} - u^{k-1}\right|\right|_2 > tol$ **do**

$x^{(k+1)} = G^{(k)}$

$d_i^{(k+1)} = \max\left(s^{(k)} - \frac{1}{\lambda}, 0\right) \frac{D_i x^{(k)} + b_i^{(k)}}{s^{(k)}}$

$d_j^{(k+1)} = \max\left(s^{(k)} - \frac{1}{\lambda}, 0\right) \frac{D_j x^{(k)} + b_j^{(k)}}{s^{(k)}}$

$b_i^{(k+1)} = b_i^{(k)} + \left(D_i x^{(k+1)} - d_i^{(k+1)}\right)$

$b_j^{(k+1)} = b_j^{(k)} + \left(D_j x^{(k+1)} - d_j^{(k+1)}\right)$

end while

2.5.2 Poisson noise

We consider the following denoising problem for data affected by Poisson noise:

$$\min_{x \in X} \left\{ \sum_{i=1}^{n} f_i \log \frac{f_i}{x_i} + x_i - f_i + \beta \sum_{i=1}^{n} ||A_i x||_2 \right\} \tag{2.29}$$

where X is the same as defined in (1.12) and A is the matrix defined in (1.15). We will denote by $\Phi(Ax)$ the term $\sum_{i=1}^{n} ||A_i x||_2$.

PIDAL and PIDSplit+ algorithms

In [9] the authors propose two different versions for the split Bregman iteration. In order to decouple the KL function and the TV, we begin by setting as constraints $w_1 = x$ and $w_2 = x$.

We then add to (2.29) the term $\frac{1}{2\gamma}||x - w_1||_2^2 + \frac{1}{2\gamma}||x - w_2^2||_2^2$

We have to minimize

$$\sum_{i=1}^{n} f_i \log \frac{f_i}{(w_1)_i} + (w_1)_i - f_i + \beta\Phi(Aw_2) + \frac{1}{2\gamma}||x - w_1||_2^2 + \frac{1}{2\gamma}||x - w_2^2||_2^2$$

The algorithm to solve this unconstrained problem is named PIDAL and it is stated as follows:

Algorithm 4 PIDAL

$b_1^{(0)} = b_2^{(0)} = 0$

$w_1^{(0)} = f$

$w_2^{(0)} = f$

for $k = 1, ...$ until a stopping criterion is reached **do**

$$x^{(k+1)} = \underset{x}{\operatorname{argmin}} \left\{ \left|\left| b_1^{(k)} + x - w_1^{(k)} \right|\right|_2^2 + \left|\left| b_2^{(k)} + x - w_2^{(k)} \right|\right|_2^2 \right\}$$

$$w_1^{(k+1)} = \underset{w_1}{\operatorname{argmin}} \left\{ \sum_{i=1}^{n} \left(f_i \log \frac{f_i}{(w_1)_i} + (w_1)_i - f_i \right) + \right.$$
$$\left. + \frac{1}{2\gamma} \left|\left| b_1^{(k)} + x^{(k+1)} - w_1 \right|\right|_2^2 \right\}$$

$$w_2^{(k+1)} = \underset{w_2}{\operatorname{argmin}} \left\{ \beta\Phi(Aw_2) + \frac{1}{2\gamma} \left|\left| b_2^{(k)} + x^{(k+1)} - w_2 \right|\right|_2^2 \right\}$$

$$b_1^{(k+1)} = b_1^{(k)} + x^{(k+1)} - w_1^{(k+1)}$$

$$b_2^{(k+1)} = b_2^{(k)} + x^{(k+1)} - w_2^{(k+1)}$$

end for

The first two steps can be solved explicitly:

$$x^{(k+1)} = \left(\left(w_1^{(k)} - b_1^{(k)} \right) + \left(w_2^{(k)} - b_2^{(k)} \right) \right) / 2$$

$$w_1^{(k+1)} = \frac{1}{2} \left(b_1^{(k)} + x^{(k+1)} - \gamma + \sqrt{\left(b_1^{(k)} + x^{(k+1)} - \gamma \right)^2 + 4\gamma f} \right)$$

For the third step we use a method for the denoising problem with Gaussian noise.

The PIDAL algorithm ensures that the sequence $w_1^{(k+1)}$ is nonnegative and it converges to a nonnegative image, but at every step we have to apply a Split Bregman Method (or another one) to determine $w_2^{(k+1)}$.

In order to simplify the approach, we introduce the indicator function: let $l_{x\geq\eta}$ denote the indicator function of X defined as

$$l_{x\geq\eta}(x) = \begin{cases} 0 & x \geq \eta \\ \infty & \text{otherwise} \end{cases}$$

We then apply the alternating split Bregman algorithm to:

$$\min_x \left\{ \sum_{i=1}^{n} x_i - f_i log(x_i) + \beta\Phi(Ax) + l_{x\geq\eta} \right\} \tag{2.30}$$

We can see this problem as the sum of three different functionals:

$$\begin{aligned} f_1(x) &:= \sum_{i=1}^{n} x_i - f_i log(x_i) \\ f_2(Ax) &:= \beta\Phi(Ax) \\ f_3(x) &:= l_{x\geq\eta} \end{aligned}$$

With this substitution we have now three constraints, $w_1 = x$, $w_2 = Ax$, $w_3 = x$.

The corrisponding split Bregman algorithm is called **PIDSplit+**.

We add to (2.30) the term $\frac{1}{2\gamma}\left(||x - w_1||_2^2 + ||Ax - w_2||_2^2 + ||x - w_3||_2^2\right)$.

The PIDSplit+ algorithm is then:

Algorithm 5 PIDSplit+

$b_1^{(0)} = b_2^{(0)} = b_3^{(0)} = 0$
$w_1^{(0)} = f$
$w_2^{(0)} = Af$
$w_3^{(0)} = f$
for $k = 1, ...$ until a stopping criterion is reached **do**

$$x^{(k+1)} = \underset{x}{\operatorname{argmin}} \left\{ \left|\left| b_1^{(k)} + x - w_1^{(k)} \right|\right|_2^2 + \left|\left| b_2^{(k)} + Ax - w_2^{(k)} \right|\right|_2^2 + \right.$$
$$\left. + \left|\left| b_3^{(k)} + x - w_3^{(k)} \right|\right|_2^2 \right\}$$

$$w_1^{(k+1)} = \underset{w_1}{\operatorname{argmin}} \left\{ \sum_{i=1}^{n} (w_1)_i - f_i \log\Big((w_1)_i\Big) + \frac{1}{2\gamma} \left|\left| b_1^{(k)} + x^{(k+1)} - w_1 \right|\right|_2^2 \right\}$$

$$w_2^{(k+1)} = \underset{w_2}{\operatorname{argmin}} \left\{ \beta ||w_2||_2 + \frac{1}{2\gamma} \left|\left| b_2^{(k)} + Ax^{(k+1)} - w_2 \right|\right|_2^2 \right\}$$

$$w_3^{(k+1)} = \underset{w_3}{\operatorname{argmin}} \left\{ l_{w_3 \geq \eta}(w_3) + \frac{1}{2\gamma} \left|\left| b_3^{(k)} + x^{(k+1)} - w_3 \right|\right|_2^2 \right\}$$

$$b_1^{(k+1)} = b_1^{(k)} + x^{(k+1)} - w_1^{(k+1)}$$

$$b_2^{(k+1)} = b_2^{(k)} + Ax^{(k+1)} - w_2^{(k+1)}$$

$$b_3^{(k+1)} = b_3^{(k)} + x^{(k+1)} - w_3^{(k+1)}$$

end for

Since $w_3^{(k)} \in X \ \forall k$, we prefer it over $x^{(k+1)}$ as output for the previous algorithm.

A direct consequence of this scheme is that we no longer need to solve a denoising problem for Gaussian noise as inner iteration loop; instead, $w_1^{(k+1)}, w_2^{(k+1)}, w_3^{(k+1)}$ can be computed directly and we get $x^{(k+1)}$ by solving this linear system of equations:

$$x^{(k+1)} = \left(2I + A^T A\right)^{-1} \left(\left(w_1^{(k)} - b_1^{(k)} \right) + A^T \left(w_2^{(k)} - b_2^{(k)} \right) + \left(w_3^{(k)} - b_3^{(k)} \right) \right)$$

$$w_1^{(k+1)} = \frac{1}{2} \left(b_1^{(k)} + x^{(k+1)} - \gamma + \sqrt{(b_1^{(k)} + x^{(k+1)} - \gamma)^2 + 4\gamma b} \right)$$

$$w_2^{(k+1)} = shrink\left(b_2^{(k)} + Ax^{(k+1)}, \gamma \right)$$

$$w_3^{(k+1)} = \begin{cases} b_3^{(k)} + y^{(k+1)} & \text{if } b_3^{(k)} + x^{(k+1)} \geq \eta \\ 0 & \text{otherwise} \end{cases}$$

A very efficient way to compute $\left(2I + A^T A\right)^{-1}$ is to apply the Discrete Fourier Transform as shown in Section A.1.3.

Chapter 3

Alternating Extragradient Method

3.1 Primal-Dual formulation of the image restoration problem

In section 1.4.2, we observed that image restoration, in the Bayesian framework, is obtained by solving the optimization problem (1.9), with $x \in X$, where X is defined by (1.12).

Being under the hypotesis of Poisson noise, and focusing as before on the edge-preserving regularization via Total Variation (1.10), the so-called **primal** formulation of the problem is:

$$\min_{x \in X} \left\{ \sum_{i \in S} \left[f_i \log \frac{f_i}{x_i} + x_i - f_i \right] + \beta \sum_{i \in S} ||A_i x||_2 \right\} \tag{3.1}$$

where A_i is the approximation of the gradient of x at the pixel i as seen in (1.15).

There are a class of methods that require a slightly different formulation of this problem: let us consider, in general, $a \in \mathbb{R}^2$; since the definition of norm in ℓ_2 can be formulated as:

$$||a||_2 = \sup_{z \in \mathbb{R}^2 : ||z||_2 \leq 1} z^T a$$

we have that

$$\sum_{i=1}^{n} ||A_i x||_2 = \sup_{y \in Y} \sum_{i=1}^{n} y_i^T A_i x \qquad \text{with } y_i = \begin{pmatrix} y_1^{(i)} \\ y_2^{(i)} \end{pmatrix} \tag{3.2}$$

where $Y = \left\{ y \in \mathbb{R}^{2n} \mid ||y_i||_2 \leq 1 \, , j = 1, ..., n \right\}$

Putting

$$y = \left(y_1^{(1)}, y_2^{(1)}, y_1^{(2)}, y_2^{(2)}, \ldots, y_1^{(n)}, y_2^{(n)} \right) \qquad A = \begin{pmatrix} A_1 \\ \vdots \\ A_n \end{pmatrix} \qquad x = \begin{pmatrix} x_1 \\ \vdots \\ x_n \end{pmatrix}$$

then (3.2) can be also written as the matrix product: $\sup_{y \in Y} y^T A x$.

The problem (3.1) can now be rewritten as

$$\min_{x \in X} \left\{ \sum_{i \in S} \left[f_i \log \frac{f_i}{x_i} + x_i - f_i \right] + \beta \sup_{y \in Y} y^T A x \right\}$$

obtaining the discrete **primal-dual** formulation of the problem (3.1):

$$\min_{x \in X} \max_{y \in Y} F(x, y) \tag{3.3}$$

with

$$F(x, y) = \left\{ \sum_{i \in S} \left[f_i \log \frac{f_i}{x_i} + x_i - f_i \right] + \beta y^T A x \right\} \tag{3.4}$$

The problem (3.1) has an unique solution $x^* \in X$. Then we can restrict the minimization problem to a closed bounded set $\overline{X} \subset X$, such that $x^* \in \overline{X}$. Since Y is a bounded set and the functional F is convex with respect to x and concave with respect to y, the Minimax Theorem in [6] guarantees the existence of at least one *saddle-point*, that is a point (x^*, y^*) such that

$$F(x^*, y) \leq F(x^*, y^*) \leq F(x, y^*)$$

This class of problems are often found in *Variational Inequality Problems* (VIP).

3.2 Extragradient Methods

As suggested in [18], one of the most effective approaches for a solution of saddle-point problems is the class of extragradient-type methods, whose iterative schemes consist in projection steps, along with a suitable choice of a steplength parameter.

A solution $v^* = (x^*, y^*)$ of (3.3) satisfies also the VIP

$$(v - v^*)^T \Phi(v^*) \geq 0 \qquad \forall v \in X \times Y \tag{3.5}$$

where

$$\Phi(v) = \Phi(x, y) = \begin{pmatrix} \nabla_x F(x, y)^T \\ \\ -\nabla_y F(x, y)^T \end{pmatrix}$$

A solution v^* of (3.5) satisfies the fixed point equation

$$v^* = P_{X \times Y}(v^* - \alpha \Phi(v^*)) \qquad \forall \alpha > 0 \tag{3.6}$$

where P denotes the orthogonal projection operator, defined as

$$y = P_\Omega(x) = \underset{z \in \Omega}{\operatorname{argmin}} \, ||x - z||_2^2$$

Since the domain $X \times Y$ has a separable structure, (3.6) holds if and only if

$$x^* = P_X(x^* - \alpha \nabla_x F(x^*, y^*)) \tag{3.7}$$

$$y^* = P_Y(y^* - \alpha \nabla_y F(x^*, y^*)) \tag{3.8}$$

Both these two projections are fairly easy to compute: (3.7) is a simple thresolding, while in (3.8) the projection onto Y is defined as $(P_Y(y))_k = s_k y_k$, with

$$s_{2i-1} = s_{2i} = \frac{1}{\max\left\{1, \sqrt{y_{2i-1}^2 + y_{2i}^2}\right\}} \qquad i = 1, \ldots, n$$

Due to the structure of the constraints in (3.3), the projection methods are attractive; however, the effectiveness of a projection method as such is deeply related to the choice of the steplength parameter.

Since F is concave with respect to y and convex with respect to x, $\Phi(v)$ is a monotone function (see Theorem A.5). A well known scheme to solve monotone variational inequality problems as (3.5) is the extragradient method defined by the iteration:

$$\begin{aligned} \bar{v}^{(k)} &= P_{X \times Y}\left(v^{(k)} - \alpha_k \Phi\left(v^{(k)}\right)\right) \\ v^{(k+1)} &= P_{X \times Y}\left(v^{(k)} - \alpha_k \Phi\left(\bar{v}^{(k)}\right)\right) \end{aligned} \tag{3.9}$$

where α_k is the steplength.

There are mainly two strategies about the choice of such steplength α_k: it can be held fixed and, usually, its value depends on an *a priori* estimate of the Lipschitz constant of Φ; otherwise it can be adaptively updated at each step, satisfying each time the sufficent condition for convergence of the method. The second choice is often the best, since an *a priori* estimate of Lipschitz constant could be difficult.

An adaptive rule to select α_k is to define the steplength so that the following condition is satisfied:

$$1 - \alpha_k^2 \frac{\left|\left|\Phi\left(v^{(k)}\right) - \Phi\left(\bar{v}^{(k)}\right)\right|\right|_2^2}{\left|\left|v^{(k)} - \bar{v}^{(k)}\right|\right|_2} > 0$$

In practice, such α_k is determined by a backtracking procedure which guaratnees the convergence of the method under suitable monotonicity and Lipschitz conditions on Φ [19].

3.3 Alternating extragradient method

Following the strategy in [19], we can state a method for saddle point problems. The iterate $v^{(k)}$ is split in two successive iterates $x^{(k)}$ and $y^{(k)}$.

The choice of the steplength is adaptive and, furthermore, there is not explicit use of the Lipschitz constant of Φ , but only a local approximation of it is used.

The method is the following:

$$\bar{y}^{(k)} = P_Y\left(y^{(k)} + \alpha_k \nabla_y F(x^{(k)}, y^{(k)})\right) \tag{3.10}$$

$$x^{(k+1)} = P_X\left(x^{(k)} + \alpha_k \nabla_x F\left(x^{(k)}, \bar{y}^{(k)}\right)\right) \tag{3.11}$$

$$y^{(k+1)} = P_Y\left(y^{(k)} + \alpha_k \nabla_y F\left(x^{(k+1)}, y^{(k)}\right)\right) \tag{3.12}$$

with the steplength α_k is chosen in the bounded interval $[\alpha_{min}, \alpha_{max}]$ with $0 < \alpha_{min} < \alpha_{max}$ and

$$\begin{cases} 1 - 2\alpha_k A_k - 2\alpha_k^2 B_k^2 & \geq \quad \epsilon \\ 1 - 2\alpha_k C_k & \geq \quad \epsilon \end{cases} \tag{3.13}$$

where $\epsilon \in (0, 1)$ is costant and

$$A_k = \frac{\left|\left|\nabla_x F\left(x^{(k+1)}, \bar{y}^{(k)}\right) - \nabla_x F\left(x^{(k)}, \bar{y}^{(k)}\right)\right|\right|_2}{\left|\left|x^{(k+1)} - x^{(k)}\right|\right|_2}$$

$$B_k = \frac{\left|\left|\nabla_y F\left(x^{(k+1)}, y^{(k)}\right) - \nabla_y F\left(x^{(k)}, y^{(k)}\right)\right|\right|_2}{\left|\left|x^{(k+1)} - x^{(k)}\right|\right|_2} \tag{3.14}$$

$$C_k = \frac{\left|\left|\nabla_y F\left(x^{(k+1)}, y^{(k)}\right) - \nabla_y F\left(x^{(k+1)}, \bar{y}^{(k)}\right)\right|\right|_2}{\left|\left|y^{(k)} - \bar{y}^{(k)}\right|\right|_2}$$

This scheme substantially consist in successive gradient ascent (3.10) and descent (3.11) steps, followed by an extragradient step (3.12). The method can also be written exchanging the roles of x and y.

In [18] it is outlined that such scheme has two main advantages to other methods:

- it is defined only by means of explicits steps;
- the convergence of the scheme is guaranteed without strict convexity assumptions, thanks to the extragradient step (3.12).

In particular, it is shown that the convergence is guaranteed by an appropriate selection of α_k, and it is shown a self-adjusting backtracking procedure to compute it.

3.4 Algorithm AEM

This is the scheme of the algoritm (3.10)-(3.12):

Algorithm 6 Alternating Extragradient Method (AEM)

Choose $\left(x^{(0)}, y^{(0)}\right) \in X \times Y$
Set θ
Set $\epsilon \in (0,1)$
Set $\alpha_{max} > 0$

Step 1:
Choose $\alpha \leq \alpha_{max}$

Step 2: Compute tentative points

$$\bar{y}^{+} \leftarrow P_Y\left(y^{(k)} + \alpha \nabla_y F\left(x^{(k)}, y^{(k)}\right)\right)$$

$$x^{+} \leftarrow P_X\left(x^{(k)} - \alpha \nabla_x F\left(x^{(k)}, \bar{y}^{+}\right)\right)$$

$$A \leftarrow \frac{||\nabla_x F\left(x^{+}, \bar{y}^{+}\right) - \nabla_x F\left(x^{(k)}, \bar{y}^{+}\right)||_2}{||x^{+} - x^{(k)}||_2}$$

$$B \leftarrow \frac{||\nabla_y F\left(x^{+}, y^{(k)}\right) - \nabla_y F\left(x^{(k)}, y^{(k)}\right)||_2}{||x^{+} - x^{(k)}||_2}$$

$$C \leftarrow \frac{||\nabla_y F\left(x^{+}, y^{(k)}\right) - \nabla_y F\left(x^{+}, \bar{y}^{+}\right)||_2}{||y^{(k)} - \bar{y}^{+}||_2}$$

$$\bar{\alpha} \leftarrow \begin{cases} \min\left\{\dfrac{\sqrt{A^2 + 2B^2(1-\epsilon)} - A}{2B^2}, \dfrac{1-\epsilon}{2C}\right\} & \text{if } B > 0, C > 0 \\ \min\left\{\dfrac{1-\epsilon}{2A}, \dfrac{1-\epsilon}{2C}\right\} & \text{if } A > 0, C > 0, B = 0 \\ \dfrac{\sqrt{A^2 + 2B^2(1-\epsilon)} - A}{2B^2} & \text{if } B > 0, C = 0 \\ \dfrac{1-\epsilon}{2C} & \text{if } A = 0, C > 0, B = 0 \\ \dfrac{1-\epsilon}{2A} & \text{if } A > 0, C = 0, B = 0 \\ \alpha & \text{otherwise} \end{cases}$$

Step 3: Check convergence condition
if $\alpha \leq \bar{\alpha}$ **then**
$\alpha_k = \alpha$
$\bar{y}^{(k)} = \bar{y}^{+}$
$x^{(k+1)} = x^{+}$
else
$\alpha \leftarrow \min\{\bar{\alpha}, \theta\alpha\}$
go to Step 2
end if

Step 4:

$$y^{(k+1)} = P_Y\left(y^{(k)} + \alpha_k \nabla_y F\left(x^{(k+1)}, y^{(k)}\right)\right)$$

In particular, for $F(x, y)$ in (3.4), we have that

$$A_k = \frac{\left|\left| \frac{f}{x^{(k)}} - \frac{f}{x^{(k+1)}} \right|\right|_2}{||x^{(k+1)} - x^{(k)}||_2}$$

$$B_k = \frac{\left|\left|\beta A x^{(k+1)} - \beta A x^{(k)}\right|\right|_2}{\left|\left|x^{(k+1)} - x^{(k)}\right|\right|_2}$$

$$C_k = 0$$

Chapter 4

Numerical simulations

This chapter will cover the numerical simulation used to compare and analyse the behavior of two image restoration methods previously described: PIDSplit+ and AEM.

4.1 Test-problems

In the experiments, we used a set of test-problems, mainly from two categories: artificial and real problems.

4.1.1 Simulated problems

This class of problems consists in images in which Poisson noise is simulated by the `imnoise` function, provided by the Matlab Image Processing Toolbox.

LCR phantom

The original image is an array 256×256 consisting in three concentric circles of intensities 70, 135 and 200 respectively, enclosed by a square frame of intensity 10, all on a background of intensity 5 (LCR-1).

Several noise levels can be simulated multiplying such LCR phantom by a factor 10 (LCR-10) and 0.2 (LCR-0.2) and subsequently generating the corresponding noisy images.

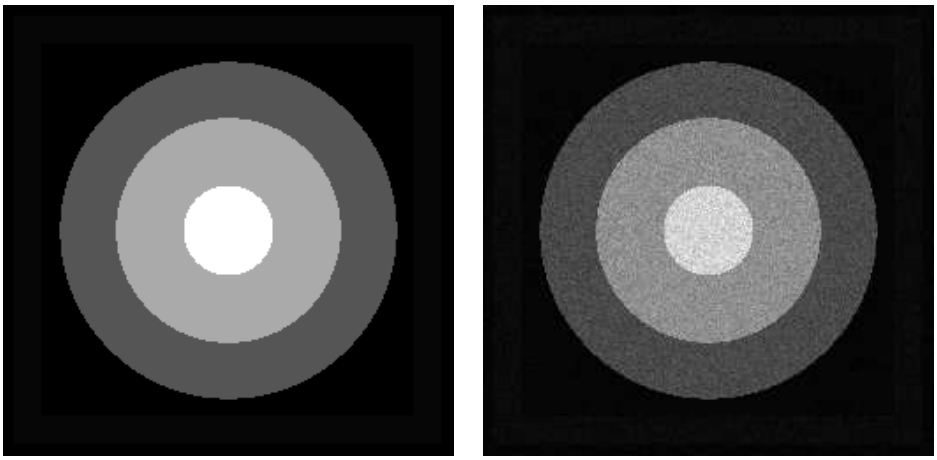


Figure 4.1: LCR-1: the original image and the noisy one

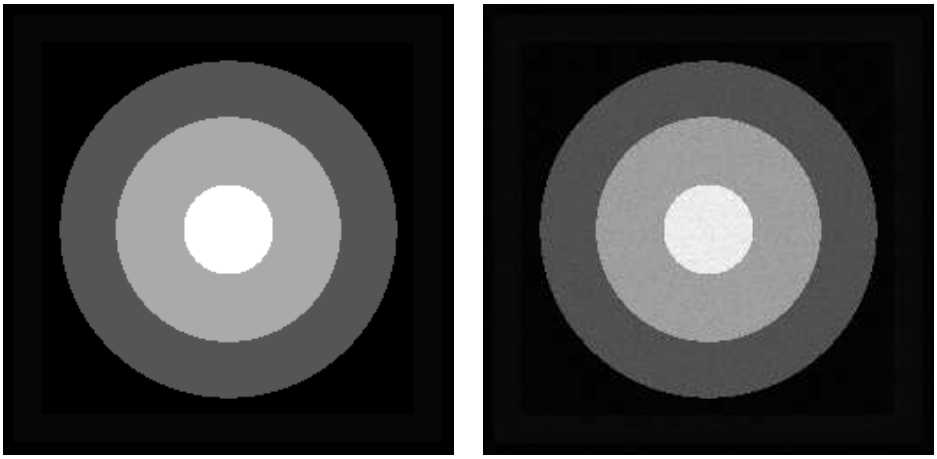


Figure 4.2: LCR-10: the original image and the noisy one

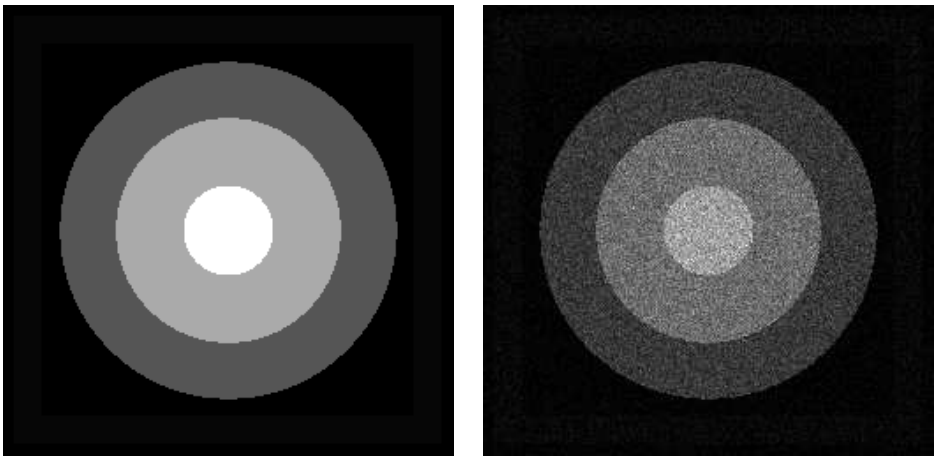


Figure 4.3: LCR-0.2: the original image and the noisy one

Airplane

The original image is an array 256×256 depicting an aerial image of an airplane on a runway, with values in the range $[0, 232]$.

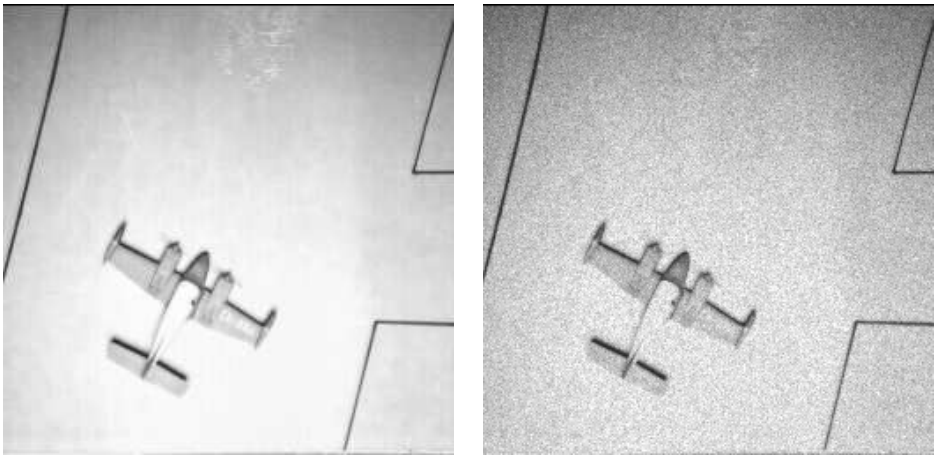


Figure 4.4: Airplane: the original image and the noisy one

Dental Radiography (DR)

The original image is an array 512×512 with values in the range $[0, 255]$.

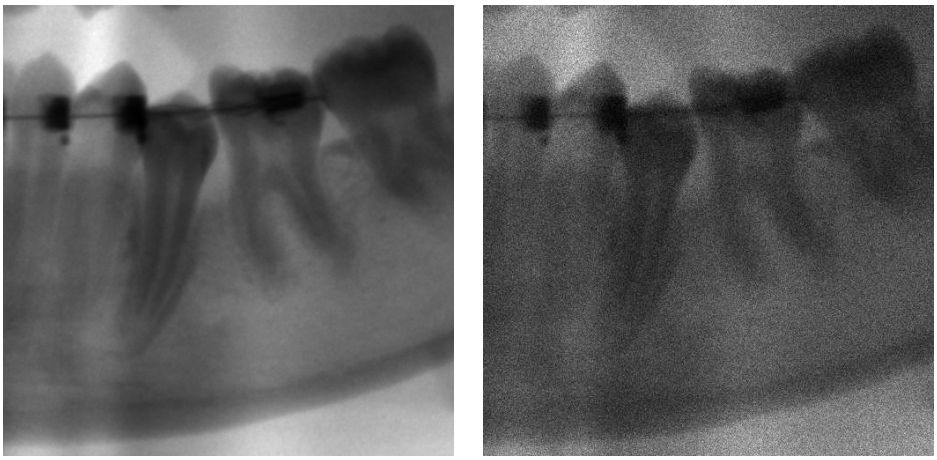


Figure 4.5: DR: the original image and the noisy one

4.1.2 Real-world problems

As an example of problems that occurs in real world, we studied a number of images (`Image11`, `Image13`, `Image22`, `Image48`, `Image49`, `Image50`) from the project "PRISMA"[1] (optimizAtion Methods and Software for Inverse PRoblems) which represent dental radiographies affected by Poisson noise.

Every image is a "negative" array of 266×266 meaning that the level 4095 stands for 0 photon collected, whereas 0 stands for sensor saturation.

To obtain an image in which the gray level is linear with respect to the number of photon collected, the following transformation (with X as the image array) has been used:

$$Z = 4095 - X$$

An important difference between the simulated problems and these real ones, is that for the formers we had a noise-free image, while for the latters we do not. For this reason, a provided "ground truth" image has been used.

[1] See `http://www.unife.it/prisma`.

Figure 4.6: Ground truth.

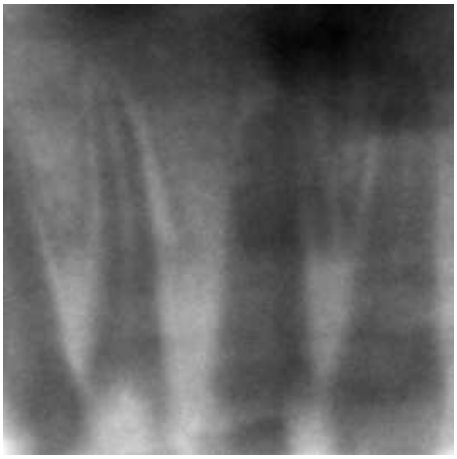


Image11

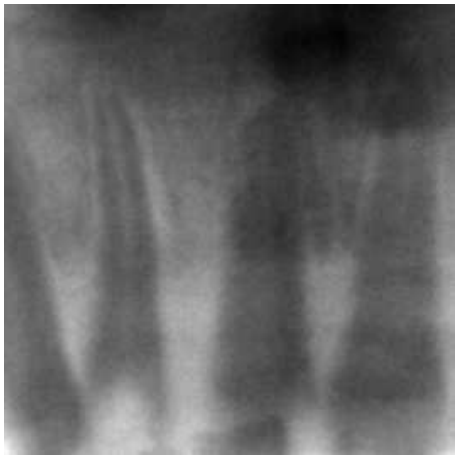


Image 13

Image22

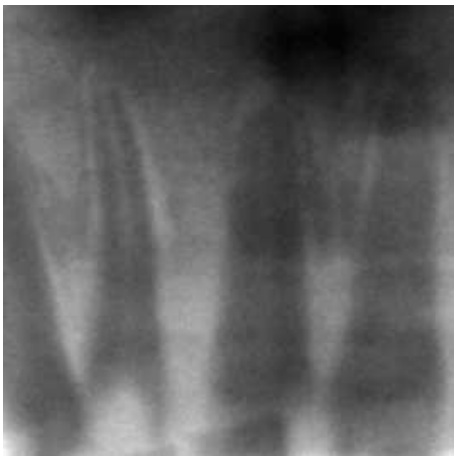


Image 48

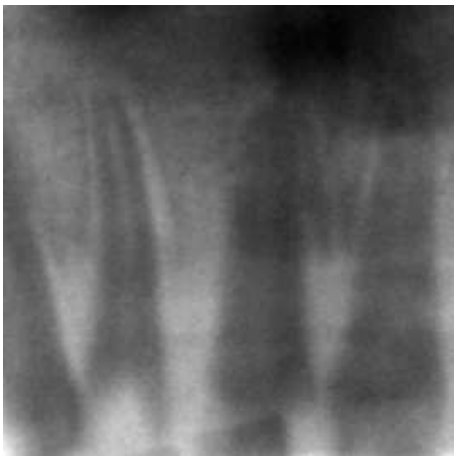


Image49

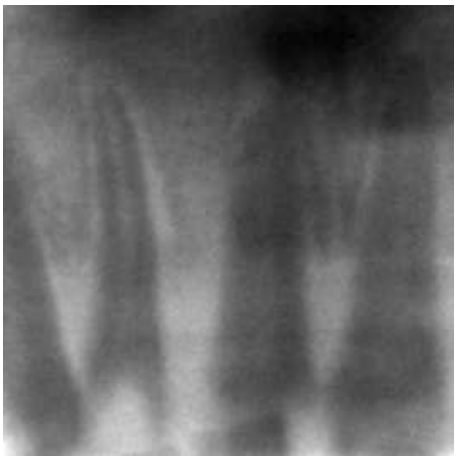


Image 50

4.2 Numerical stability of AEM and PIDSplit+

In the first set of experiments we compare the numerical behavior of the methods previously described, PIDSplit+ and AEM, on the five denoising problems LCR-1, LCR-10, LCR-0.2, Airplane, and DR.

To get the ideal solution $\tilde{x}$ of the minimization problem, we run AEM until 10000 iterations are reached.

Both the algorithms have been initialized with $x^{(0)} = max\{\eta, f\}$, where the maximum is meant component-wise, η is the same as in (1.12) and f is the noisy image. The initial guess for the dual variable $y^{(0)}$ for AEM has been set to zero.

The values of β used for each problem in (1.9), as suggested in [20], are shown in the following table:

Problem	β
LCR-1	0.25
LCR-10	0.05
LCR-0.2	0.575
Airplane	0.05
DR	0.27

Table 4.1: Values of β

Since the value of γ required by the PIDSplit+ algorithm is a user-supplied parameter which depends on the problem, we choose the values shown in the table below, along with the name used to refer to the PIDSplit+ algorithm with a particular value of γ.

γ	Label
$50/\beta$	PID50
$5/\beta$	PID5
$1/\beta$	PID1
$0.5/\beta$	PID05

Table 4.2: Values of γ and the labels used.

In order to evaluate the accuracy of the results and the effectiveness of the methods, in the following tables we report the iteration ($iter$) and time ($t^{(iter)}$) in seconds required to reach a particular threshold of relative error, defined as

$$\epsilon^{(iter)} = \frac{\left|\left|x^{(iter)} - \tilde{x}\right|\right|_2}{||\tilde{x}||_2}$$

where $iter$ is the iteration number and $\tilde{x}$ is the approximate solution of the minimization problem, as described previously.

Furthermore, we define the relative reconstruction error:

$$e^{(iter)} = \frac{\left|\left|x^{(iter)} - x^*\right|\right|_2}{||x^*||_2}$$

where x^* is the original, noise-free, image.

An empty cell in the following tables indicates that the particular error threshold ϵ has not been reached in 3000 iterations.

Method	$\epsilon = 10^{-2}$			$\epsilon = 10^{-4}$		
	$iter$	$t^{(iter)}$	$e^{(iter)}$	$iter$	$t^{(iter)}$	$e^{(iter)}$
AEM	64	3.45	0.0230578	605	29.91	0.0249536
PID50	65	3.47	0.0277052			
PID5	27	1.42	0.0250464	1023	57.9	0.0249536
PID1	82	4.43	0.022816	594	33.43	0.0249649
PID05	158	9.17	0.0226965	1183	66.91	0.0249641
	$\epsilon = 10^{-5}$			$\epsilon = 5 \cdot 10^{-6}$		
	$iter$	$t^{(iter)}$	$e^{(iter)}$	$iter$	$t^{(iter)}$	$e^{(iter)}$
AEM	1943	96.67	0.0249837			
PID50						
PID5						
PID1	1029	58.08	0.0249835	1478	83.33	0.0249836
PID05	1932	109.24	0.0249832	2186	124.34	0.249834

Table 4.3: Results for LCR-1

Method	$t^{(3000)}$	$\epsilon^{(3000)}$	$e^{(3000)}$
AEM	149.91	$6.3934 \cdot 10^{-6}$	0.0249836
PID50	167.23	$3.30784 \cdot 10^{-4}$	0.0249921
PID5	170.81	$1.86697 \cdot 10^{-5}$	0.0249839
PID1	169.24	$2.40734 \cdot 10^{-6}$	0.0249835
PID05	169.9	$1.0246 \cdot 10^{-6}$	0.0249835

Table 4.4: Results for LCR-1 after 3000 iterations.

Method	$\epsilon = 10^{-2}$			$\epsilon = 10^{-4}$		
	$iter$	$t^{(iter)}$	$e^{(iter)}$	$iter$	$t^{(iter)}$	$e^{(iter)}$
AEM	92	4.65	0.0118767	816	40.9	0.00835666
PID50	5	0.22	0.0110561	540	30.13	0.00838595
PID5	24	1.24	0.00119889	320	17.94	0.00833781
PID1	116	6.39	0.0119236	1592	96.69	0.00833768
	$\epsilon = 10^{-5}$			$\epsilon = 5 \cdot 10^{-6}$		
	$iter$	$t^{(iter)}$	$e^{(iter)}$	$iter$	$t^{(iter)}$	$e^{(iter)}$
AEM	1360	66.87	0.00838057	1531	74.92	0.00838169
PID50	2622	146.63	0.00838286			
PID5	558	31.19	0.00837926	654	36.6	0.00838144
PID1	2731	153.58	0.00837883			

Table 4.5: Results for LCR-10

Method	$t^{(3000)}$	$\epsilon^{(3000)}$	$e^{(3000)}$
AEM	143.01	$2.23187 \cdot 10^{-7}$	0.00838268
PID50	167.66	$8.25419 \cdot 10^{-6}$	0.00838283
PID5	169.4	$7.34523 \cdot 10^{-7}$	0.00838268
PID1	169.15	$5.8626 \cdot 10^{-6}$	0.00832767

Table 4.6: Results for LCR-10 after 3000 iterations.

Method	$\epsilon = 10^{-2}$			$\epsilon = 10^{-4}$		
	$iter$	$t^{(iter)}$	$e^{(iter)}$	$iter$	$t^{(iter)}$	$e^{(iter)}$
AEM	102	5.97	0.0441267	2104	113.06	0.0447745
PID50	478	25.79	0.0465756			
PID5	60	3.04	0.0466034			
PID1	68	3.76	0.0438891	696	37.92	0.0447745
PID05	119	6.50	0.0432181	606	33.01	0.0447726
	$\epsilon = 10^{-5}$			$\epsilon = 5 \cdot 10^{-6}$		
	$iter$	$t^{(iter)}$	$e^{(iter)}$	$iter$	$t^{(iter)}$	$e^{(iter)}$
AEM						
PID50						
PID5						
PID1	1418	77.80	0.044773	1591	87.5	0.047729
PID05						

Table 4.7: Results for LCR-0.2

Method	$t^{(3000)}$	$\epsilon^{(3000)}$	$e^{(3000)}$
AEM	160.72	$4.42311 \cdot 10^{-5}$	0.0447735
PID50	164.51	$1.00901 \cdot 10^{-3}$	0.0448293
PID5	165.22	$1.34896 \cdot 10^{-4}$	0.0447752
PID1	164.91	$1.80577 \cdot 10^{-5}$	0.0447724
PID05	164.95	$2.79695 \cdot 10^{-5}$	0.0447722

Table 4.8: Results for LCR-0.2 after 3000 iterations.

Method	$\epsilon = 10^{-2}$			$\epsilon = 10^{-4}$		
	$iter$	$t^{(iter)}$	$e^{(iter)}$	$iter$	$t^{(iter)}$	$e^{(iter)}$
AEM	68	3.32	0.0258942	192	9.02	0.0214011
PID50	7	0.33	0.0227146	451	25.82	0.0213743
PID5	18	0.96	0.0252871	75	4.23	0.0213782
PID1	82	4.53	0.0258994	324	17.98	0.0213914
PID05	163	9.06	0.0258684	646	36.93	0.0213913
	$\epsilon = 10^{-5}$			$\epsilon = 5 \cdot 10^{-6}$		
	$iter$	$t^{(iter)}$	$e^{(iter)}$	$iter$	$t^{(iter)}$	$e^{(iter)}$
AEM	266	12.86	0.0213756	350	17.03	0.0213742
PID50						
PID5	362	20.63	0.0213741	679	38.41	0.0213742
PID1	481	26.78	0.0213754	534	30.02	0.0213748
PID05	956	54.98	0.0213754	1056	60.53	0.0213748

Table 4.9: Results for Airplane

Method	$t^{(3000)}$	$\epsilon^{(3000)}$	$e^{(3000)}$
AEM	148.21	$4.66582 \cdot 10^{-7}$	0.0213742
PID50	172.8	$1.07853 \cdot 10^{-5}$	0.0213741
PID5	169.26	$1.01562 \cdot 10^{-6}$	0.0213742
PID1	169.1	$1.97154 \cdot 10^{-7}$	0.0213742
PID05	171.54	$9.10447 \cdot 10^{-8}$	0.0213742

Table 4.10: Results for Airplane after 3000 iterations.

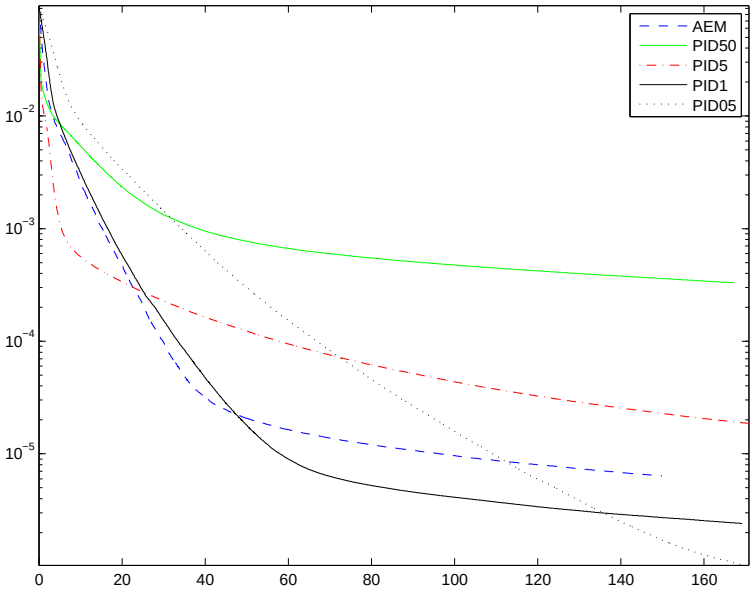
Method	$\epsilon = 10^{-2}$			$\epsilon = 10^{-4}$		
	$iter$	$t^{(iter)}$	$e^{(iter)}$	$iter$	$t^{(iter)}$	$e^{(iter)}$
AEM	43	8.24	0.0275251	481	93.32	0.0296553
PID50	91	17.86	0.0277751			
PID5	19	3.53	0.0281259	921	195.09	0.0296544
PID1	47	9.91	0.0288035	234	49.92	0.0296558
PID05	90	19.26	0.0291545	378	80.88	0.0296571
	$\epsilon = 10^{-5}$			$\epsilon = 5 \cdot 10^{-6}$		
	$iter$	$t^{(iter)}$	$e^{(iter)}$	$iter$	$t^{(iter)}$	$e^{(iter)}$
AEM						
PID50						
PID5						
PID1	1361	291.52	0.0296611	2567	549.45	0.0296614
PID05	726	154.72	0.0296613	1131	240.76	0.0296614

Table 4.11: Results for DR

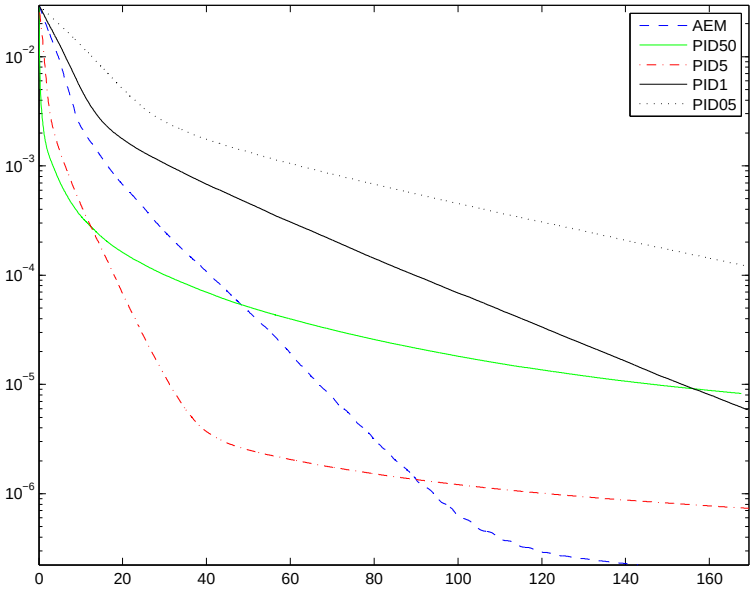
Method	$t^{(3000)}$	$\epsilon^{(3000)}$	$e^{(3000)}$
AEM	592.32	$1.11678 \cdot 10^{-5}$	0.029661
PID50	632.11	$3.33547 \cdot 10^{-4}$	0.0296372
PID5	637.26	$2.34327 \cdot 10^{-5}$	0.02966
PID1	642.70	$4.20732 \cdot 10^{-6}$	0.0296615
PID05	638.78	$1.58306 \cdot 10^{-6}$	0.0296616

Table 4.12: Results for DR after 3000 iterations.

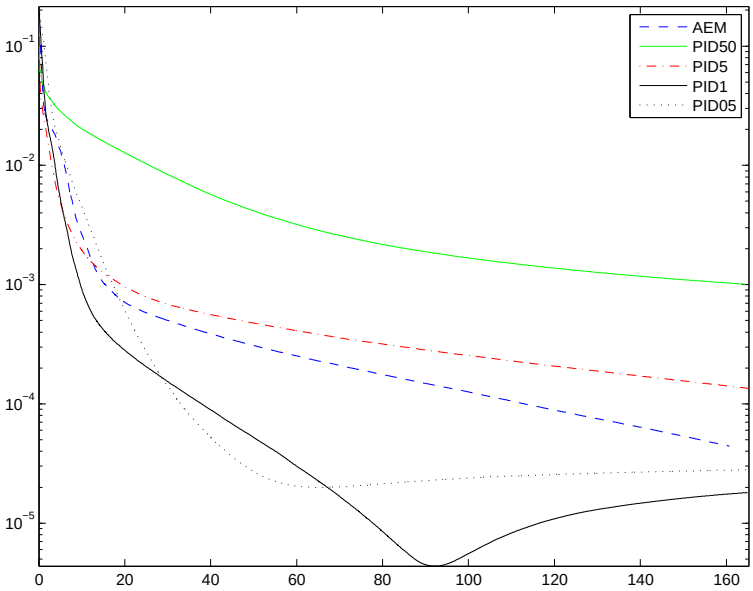
In the following figures, we show the speed of convergence of every method in 3000 iterations, showing the relation between ϵ and the time elapsed in seconds.


AEM
PID50
PID5
PID1
PID05
10^{-2}
10^{-3}
10^{-4}
10^{-5}
0
20
40
60
80
100
120
140
160

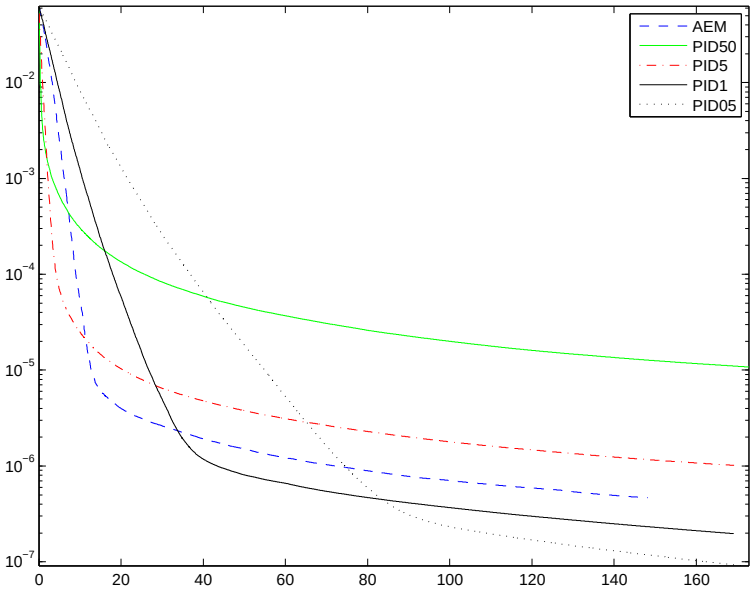
LCR-1


AEM
PID50
PID5
PID1
PID05
10^{-2}
10^{-3}
10^{-4}
10^{-5}
10^{-6}
0
20
40
60
80
100
120
140
160

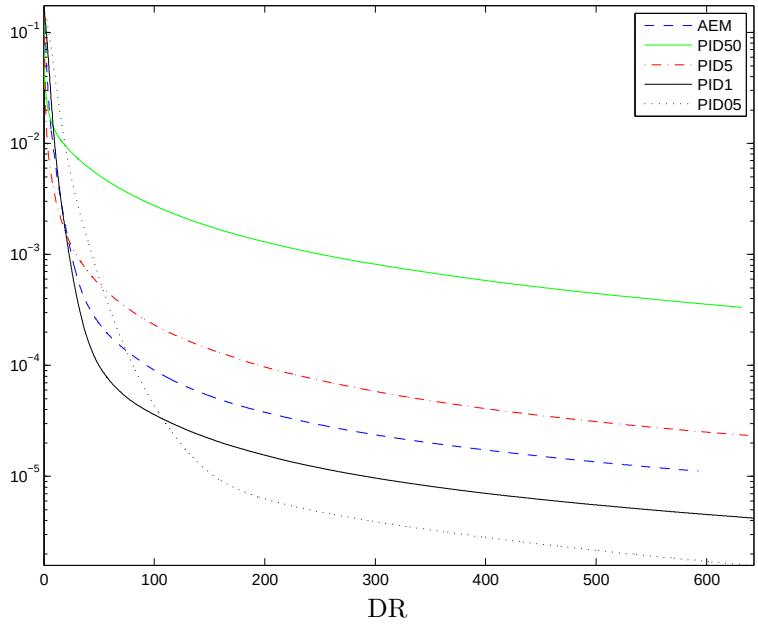
LCR-10



LCR-0.2



Airplane

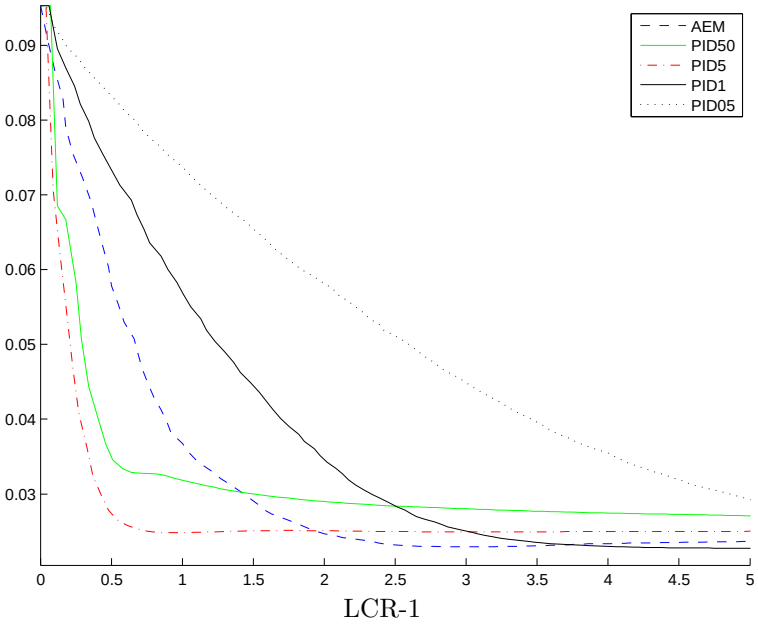


In order to evaluate the initial rate of convergence of the methods, we stopped both the algorithms after 5 seconds. In Table 4.13 we report the number of iterations performed and the corresponding reconstruction error e.

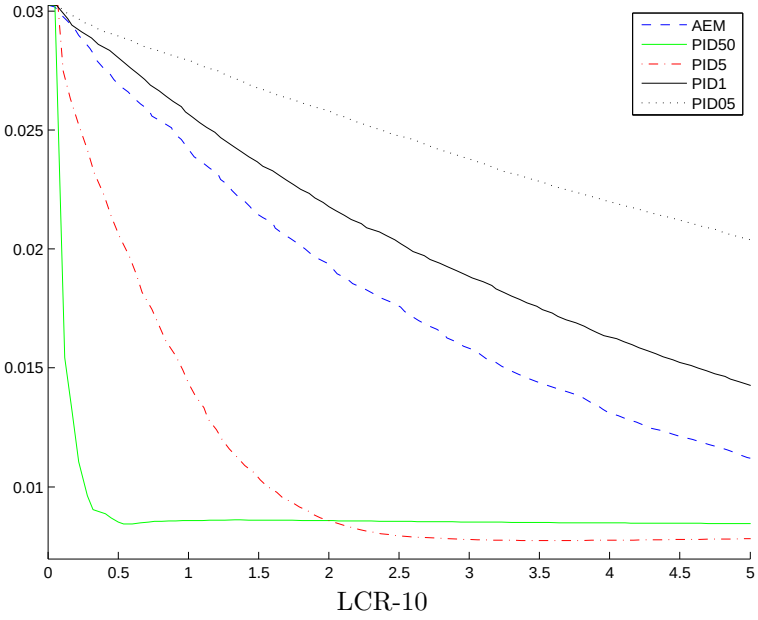
Method	LCR-1		LCR-10		LCR-0.2	
	iter	$e^{(iter)}$	*iter*	$e^{(iter)}$	*iter*	$e^{(iter)}$
AEM	95	0.0236706	99	0.01113338	86	0.0444788
PID50	90	0.0270578	91	0.00845165	98	0.0542035
PID5	93	0.0250121	92	0.0078265	97	0.0453821
PID1	92	0.0227458	92	0.0140898	91	0.0439414
PID05	88	0.028708	93	0.020299	93	0.043499
	Airplane		DR			
	iter	$e^{(iter)}$	*iter*	$e^{(iter)}$		
AEM	109	0.0221838	27	0.0293069		
PID50	91	0.021386	27	0.0284085		
PID5	88	0.0213743	27	0.0286992		
PID1	91	0.024726	24	0.0472261		
PID05	91	0.0347613	24	0.0932484		

Table 4.13: Number of iterations performed and reconstruction errors after 5 seconds.

In the following figures, we report the initial rate of convergence for both the algorithms, showing the relation between time and e in the first 5 seconds.

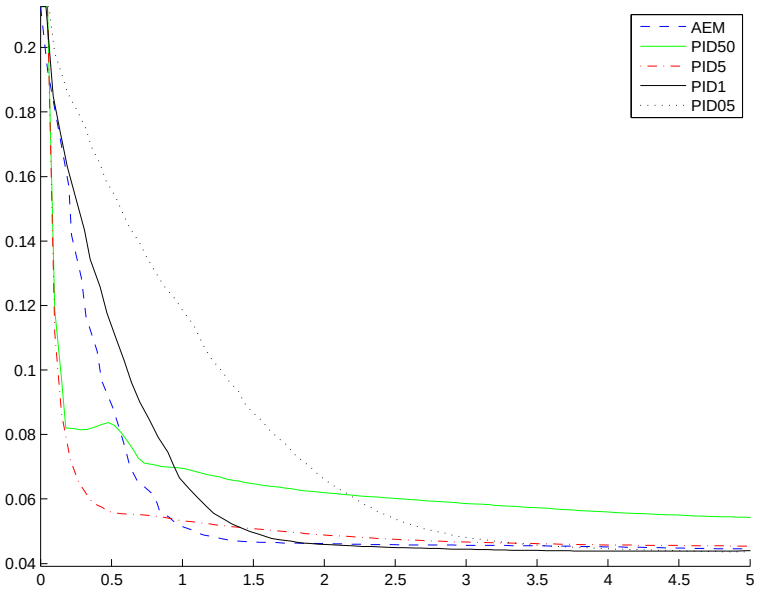


LCR-1

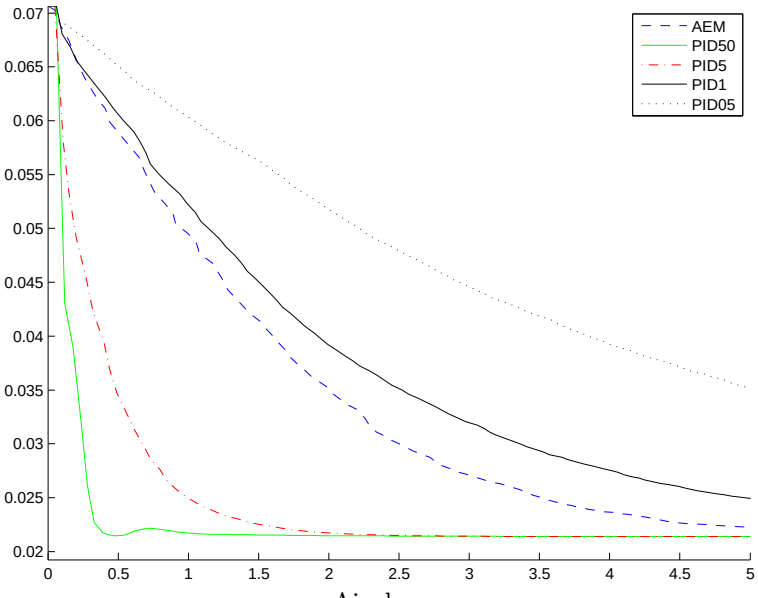


LCR-10

Figure 4.7: Behavior of relative reconstruction error in the first 5 seconds of the different methods for LCR-1 and LCR-10.



LCR-0.2



Airplane

Figure 4.8: Behavior of relative reconstruction error in the first 5 seconds of the different methods for LCR-0.2 and Airplane.

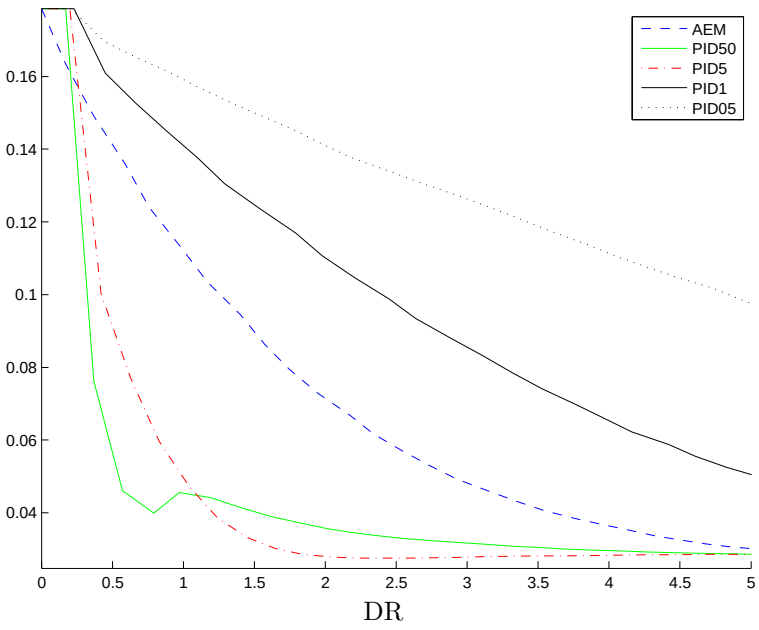
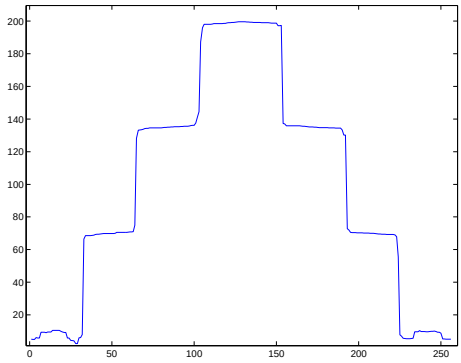
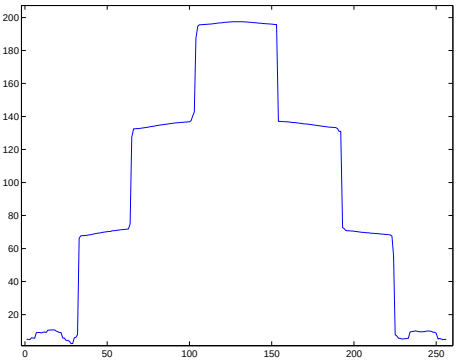


Figure 4.9: Behavior of relative reconstruction error in the first 5 seconds of the different methods for DR.

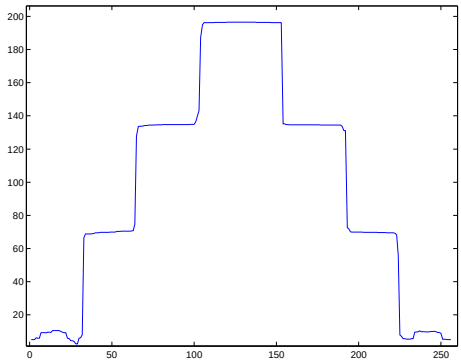
In order to evaluate if 5 seconds are enough to determine a good reconstruction of the image, we report the results after 5 seconds. For LCR-1, LCR-10, LCR-0.2, instead of the resulting image, we show the relation between the number of the column and the intensity, for the line 128.



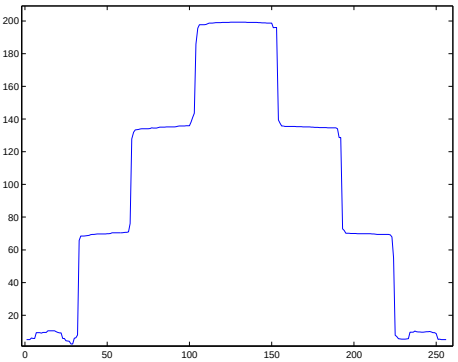
AEM: 95 iterations



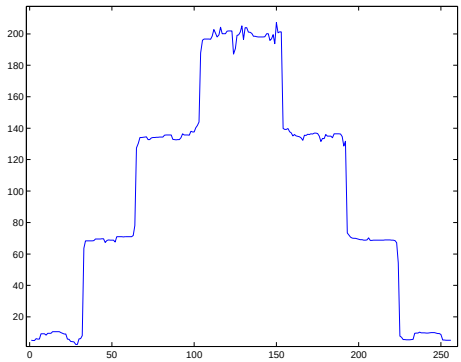
PID50: 90 iterations



PID5: 93 iterations

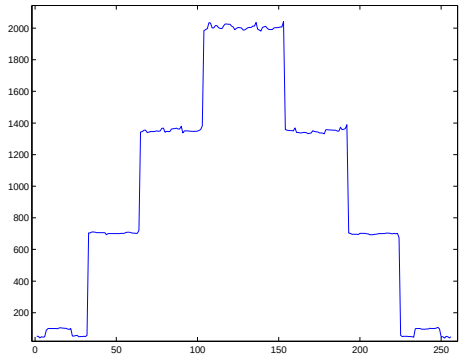


PID1: 92 iterations

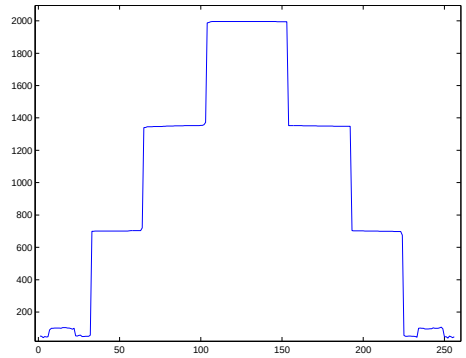


PID05: 88 iterations

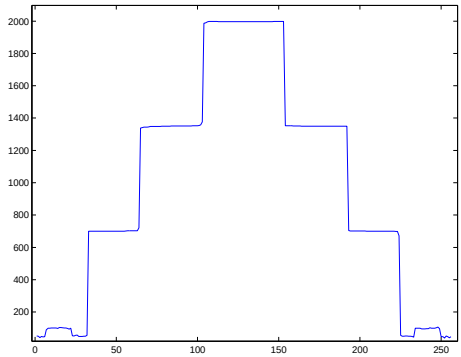
Figure 4.10: Line 128 of LCR-1 after 5 seconds



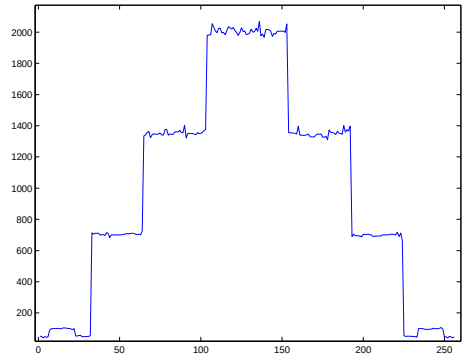
AEM: 99 iterations



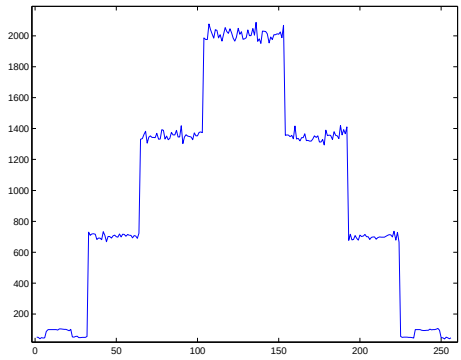
PID50: 91 iterations



PID5: 92 iterations

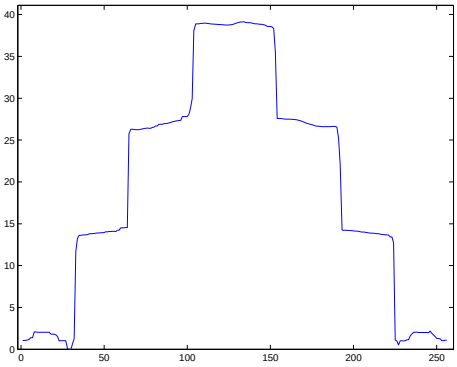


PID1: 92 iterations

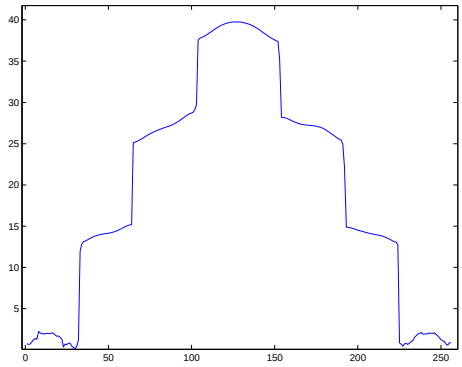


PID05: 93 iterations

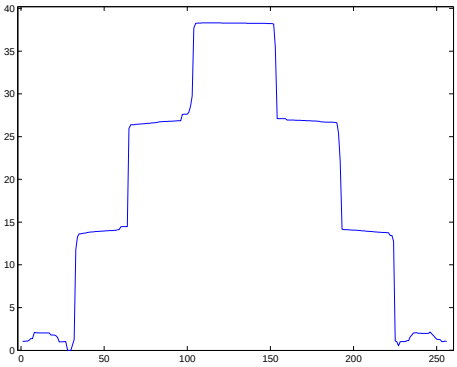
Figure 4.11: Line 128 of LCR-10 after 5 seconds



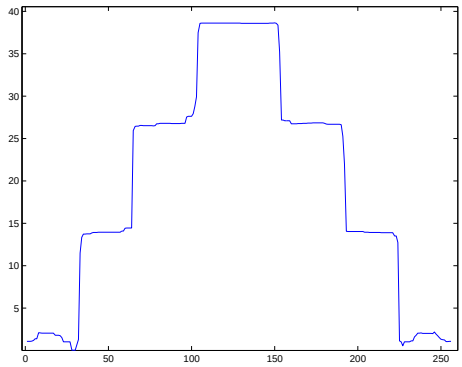
AEM: 86 iterations



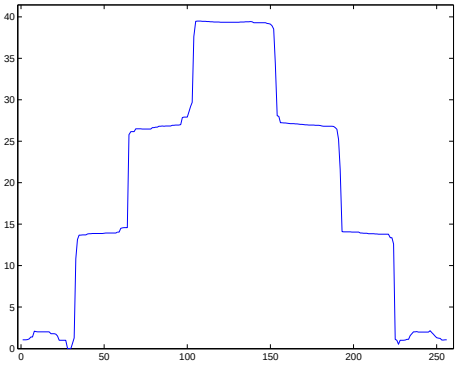
PID50: 98 iterations



PID5: 97 iterations

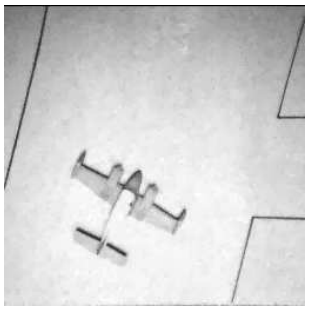


PID1: 91 iterations

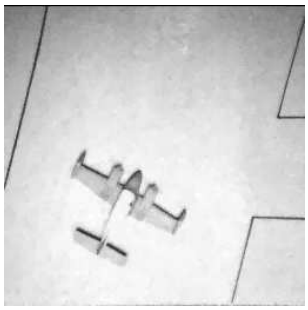


PID05: 93 iterations

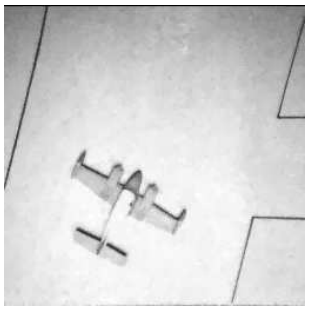
Figure 4.12: Line 128 of LCR-0.2 after 5 seconds



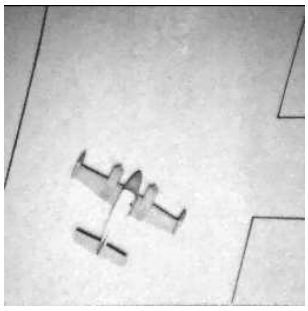
AEM: 109 iterations



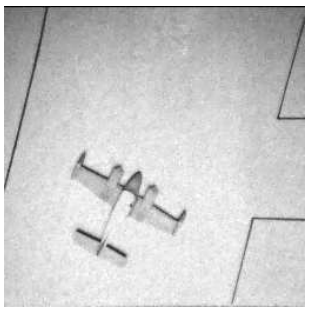
PID50: 91 iterations



PID5: 88 iterations



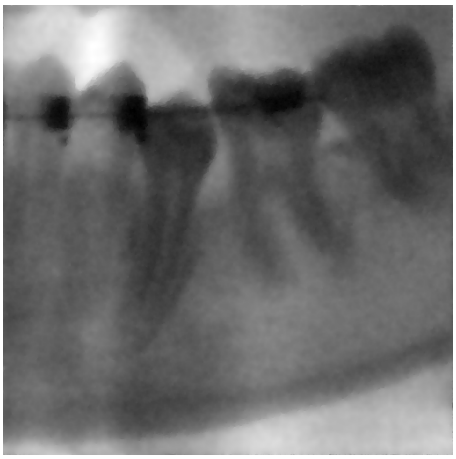
PID1: 91 iterations



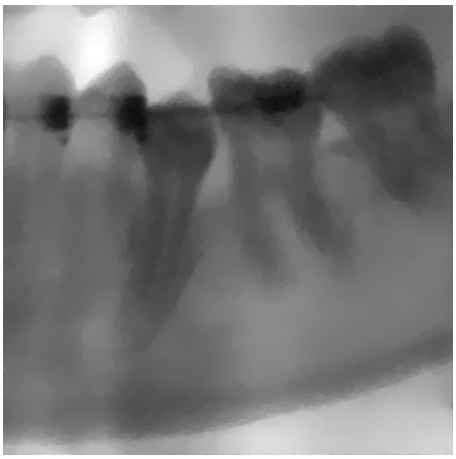
PID05: 91 iterations

Figure 4.13: Airplane after 5 seconds

AEM: 27 iterations



PID50: 27 iterations



PID5: 27 iterations

PID1: 24 iterations

PID05: 24 iterations

Figure 4.14: DR after 5 seconds

From these results we can draw the following remarks.

- The rate of convergence of PIDSplit+ as a minimization method is strongly conditioned by the choice of the parameter γ: values which have been observed to give a good initial rate of convergence $\left(\text{i.e. high values such as } \frac{50}{\beta} \text{ and } \frac{5}{\beta}\right)$ are not asymptotically satisfactory and *vice versa*.
- In terms of rates of convergence, AEM is instead located somewhat between the 4 different versions of PIDSplit+, regarding both the initial behavior and the asymptotic one.
- Any iteration of AEM is, in general, less demanding in terms of computational complexity than any of PIDSplit+, since in the latter the solution of a system is required at each step with the Fast Fourier Transform algorithm (see section A.1.3).
- AEM is self-consistent, since it does not depend from any parameter, because the choice of the steplength is adaptive; however, an optimal choice of the parameter γ for the PIDSplit+ may provide good results even faster than AEM.
- If the image considered has a high signal-to-noise ratio (as LCR-0.2) or is significantly bigger than the others (as DR), AEM is observed to have an overall good behavior with respect to the reconstruction error $e^{(rec)}$ also at the initial steps.

4.3 Behavior of AEM and PIDSplit+ with real-world problems

In the second set of experiments, considering the images from the project "PRISMA", we first compute the optimal value β_{disc} for β, using the discrepancy principle (1.11) proposed in [1].

In order to solve the nonlinear equation in β involved in the principle, we use a bisection method with the tolerance of 10^{-3} combined with AEM to evaluate x^*_β, with the maximum number of iterations set at 3000. The results are the following:

Problem	β_{disc}
Image11	0.3119
Image13	0.317748
Image22	0.31385
Image48	0.31385
Image49	0.309951
Image50	0.31385

Table 4.14: Values of β_{disc} for each image.

In order to understand the efficacy of such discrepancy principle, we examine some values in the left neighborhood of β_{disc}, defining as β_{opt} the one which provides, among them, the lowest reconstruction error with respect to the ground truth image. Such relative reconstruction error is defined, as previously, as:

$$e^{(iter)} = \frac{||x^{(iter)} - x^*||_2}{||x^*||_2}$$

where x^* is now the ground truth.

The results are shown in the following tables (the line in bold represents the value of β chosen as β_{opt}):

Problem	β	$e^{(3000)}$	KL	TV
Image11	0.01	0.0177422	2149.67	940135
	0.02	**0.016393**	**3915.19**	**814481**
	0.05	0.0164795	7272	709033
	0.07	0.0175498	9217.41	676261
	0.1	0.0196065	12111.6	641869
	0.2	0.0272481	22348.4	571412
	0.3	0.0348308	34156.3	523622
Problem	β	$e^{(3000)}$	KL	TV
Image13	0.01	0.016544	2160.85	928899
	0.02	**0.0152543**	**3911.13**	**804136**
	0.05	0.0157587	7220.98	700156
	0.07	0.0170977	9135.89	667931
	0.1	0.0194859	11991.5	633970
	0.2	0.027614	21957.3	565456
	0.3	0.0353886	33566.1	518485
Problem	β	$e^{(3000)}$	KL	TV
Image22	0.01	0.0162132	2198.92	941487
	0.02	**0.0148022**	**4025.25**	**811476**
	0.05	0.0152197	7450.87	703634
	0.07	0.0165708	9409.18	670663
	0.1	0.0190068	12318.1	636078
	0.2	0.0272591	22447.1	566297
	0.3	0.0350216	34060.9	519290

Problem	β	$e^{(3000)}$	KL	TV
Image48	0.01	0.0154639	2157.87	932984
	0.02	**0.0139999**	**3917.88**	**807767**
	0.05	0.0144383	7249.03	703296
	0.07	0.0157969	9165.57	671017
	0.1	0.0182356	12013.4	637173
	0.2	0.0267319	22221.7	566942
	0.3	0.0346593	33915.8	519601
Problem	β	$e^{(3000)}$	KL	TV
Image49	0.01	0.0160722	2159.3	940036
	0.02	**0.0145997**	**3949.21**	**812753**
	0.05	0.148771	7346.71	706008
	0.07	0.0161719	9307.99	672987
	0.1	0.0185724	12237.8	638159
	0.2	0.0268924	22499.1	567536
	0.3	0.0348179	34338.2	519626
Problem	β	$e^{(3000)}$	KL	TV
Image50	0.01	0.0158575	2175.75	932012
	0.02	**0.0144898**	**3963.06**	**804892**
	0.05	0.0150870	7921.30	699159
	0.07	0.0165020	9225.63	667082
	0.1	0.0902397	12087.30	633079
	0.2	0.0276045	22290.52	562893
	0.3	0.0355275	33892.61	515937

Using both β_{opt} and β_{disc} we run AEM and PIDSplit+ for all of the images considered, setting a maximum number of 3000 iterations and using the following stopping criterion:

$$\frac{||w^{(k+1)} - w^{(k)}||_2}{||w^{(k+1)}||_2} < 10^{-6} \tag{4.1}$$

where, for AEM:

$$w^{(k)} = \left(x^{(k)}, y^{(k)}\right)^T$$

while, for PIDSplit+:

$$w^{(k)} = \left(x^{(k)}, w_1^{(k)}, w_2^{(k)}, w_3^{(k)}, b_1^{(k)}, b_2^{(k)}, b_3^{(k)}\right)^T$$

In the following tables we show the time elapsed for each run of the algorithm considered, along with the number of iterations performed before satisfying the stopping criterion, the relative reconstruction error obtained and the value of the primal function.

The * mark indicates that the stopping criterion has not been fulfilled and the algorithm stopped due to the maximum number of iterations reached.

Method	$t^{(iter)}$	$iter$	$e^{(iter)}$	$\phi\left(x^{(iter)}\right)$
	β_{opt}			
AEM	11.93	242	0.0163191	19931.5
PID50	69.68	793	0.0163193	19931.6
PID5	8.77	101	0.0163191	19931.6
PID1	25.67	295	0.0163184	19931.6
PID05	42.75	486	0.0163183	19331.7
	β_{disc}			
AEM	140.05	2616	0.356273	194203
PID50	78.41	906	0.035957	194200
PID5	56.73	651	0.0358966	194198
PID1	191.92	2228	0.0355936	194204
PID05	258.6*	3000*	0.0345798*	194273*

Table 4.15: Results for Image11.

Method	$t^{(iter)}$	$iter$	$e^{(iter)}$	$\phi\left(x^{(iter)}\right)$
	β_{opt}			
AEM	12.74	241	0.0152054	19735.4
PID50	70.85	819	0.015209	19735.4
PID5	8.82	103	0.0152087	19735.4
PID1	25.9	294	0.0152068	19735.4
PID05	41.75	483	0.015205	19735.6
	β_{disc}			
AEM	133.62	2648	0.0365654	195065
PID50	79.05	910	0.0369184	195062
PID5	55.79	655	0.0368513	195060
PID1	195.75	22.48	0.0365267	195066
PID05	260.33*	3000*	0.354137*	195142*

Table 4.16: Results for Image13.

Method	$t^{(iter)}$	$iter$	$e^{(iter)}$	$\phi\left(x^{(iter)}\right)$
	β_{opt}			
AEM	12.45	241	0.0147852	19966
PID50	73.4	843	0.0147872	19966.1
PID5	9.15	105	0.0147871	19966.1
PID1	25.98	297	0.0147855	19966
PID05	43.06	488	0.0147841	19966.2
	β_{disc}			
AEM	134.51	2628	0.0359532	193683
PID50	79.08	909	0.0362905	193680
PID5	56.64	651	0.0362275	193678
PID1	190.93	2234	0.0359172	193684
PID05	258.15*	3000*	0.0348683*	193755*

Table 4.17: Results for Image22.

Method	$t^{(iter)}$	$iter$	$e^{(iter)}$	$\phi\left(x^{(iter)}\right)$
	β_{opt}			
AEM	13.71	242	0.0139344	19793.2
PID50	77.43	887	0.013936	19793.3
PID5	9.69	110	0.0139358	19793.3
PID1	26.21	302	0.0139345	19793.2
PID05	43.13	495	0.013934	19793.4
	β_{disc}			
AEM	140.12	2623	0.0356696	193509
PID50	79.23	928	0.0360122	193506
PID5	56.87	651	0.0359488	193504
PID1	195.64	2234	0.0356353	193510
PID05	259.99*	3000*	0.0345653*	193582*

Table 4.18: Results for Image48.

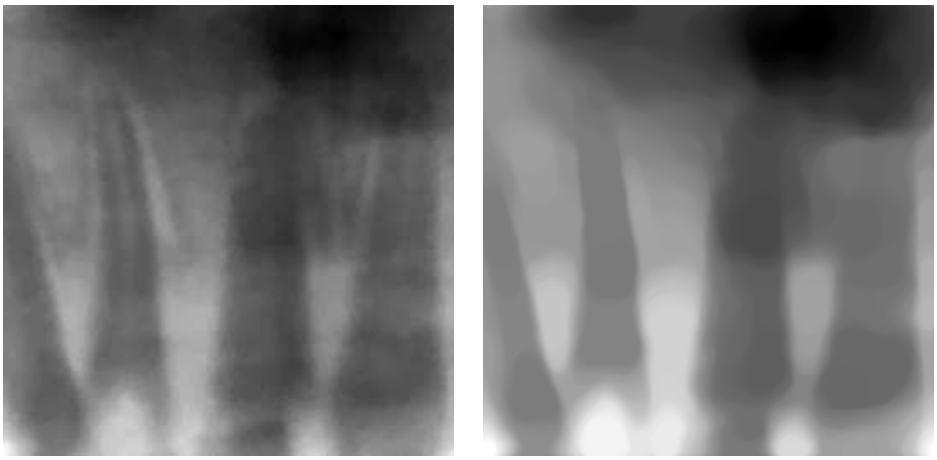
Method	$t^{(iter)}$	$iter$	$e^{(iter)}$	$\phi\left(x^{(iter)}\right)$
	β_{opt}			
AEM	12.77	243	0.014611	19908.6
PID50	73.32	841	0.0146128	19908.6
PID5	9.27	106	0.0146125	19908.6
PID1	26.21	302	0.0146112	19908.6
PID05	43.59	496	0.0146104	19908.8
	β_{disc}			
AEM	130.64	2594	0.0355119	191946
PID50	79.4	921	0.0358461	191943
PID5	55.04	645	0.035784	191941
PID1	192.5	2205	0.035474	191947
PID05	259.79*	3000*	0.0344835*	192011*

Table 4.19: Results for Image49.

Method	$t^{(iter)}$	$iter$	$e^{(iter)}$	$\phi\left(x^{(iter)}\right)$
	β_{opt}			
AEM	11.88	242	0.0145248	19762.3
PID50	73.02	837	0.014528	19762.4
PID5	9.5	105	0.0145277	19762.4
PID1	26.03	300	0.0145258	19762.3
PID05	43.01	494	0.0145241	19762.5
	β_{disc}			
AEM	132.21	2619	0.0365043	192324
PID50	81.43	951	0.036849	192321
PID5	55.45	647	0.0367837	192319
PID1	195.34	2223	0.0394668	192324
PID05	261.35*	3000*	0.0354126*	192394*

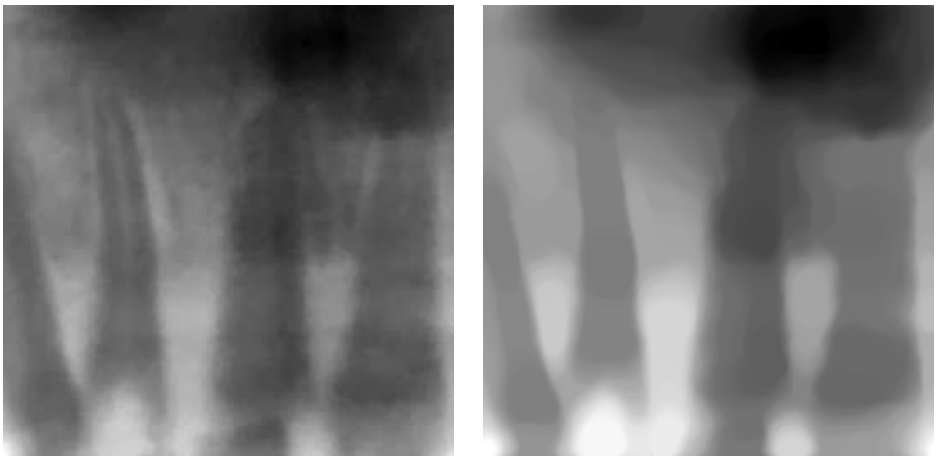
Table 4.20: Results for Image50.

Below are shown the images resulting from PIDSplit+ with $\gamma = \dfrac{5}{\beta}$ for both β_{opt} and β_{disc}.



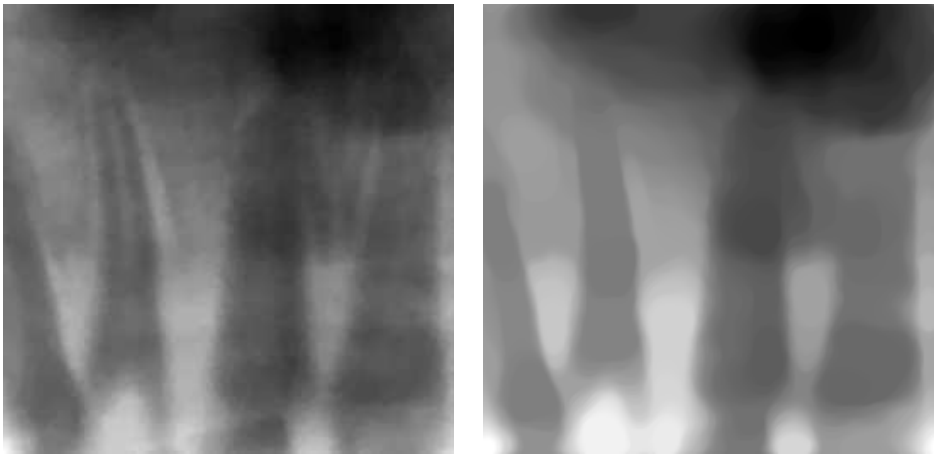
β_{opt} β_{disc}

Figure 4.15: Image11 after the stopping criterion has been reached.



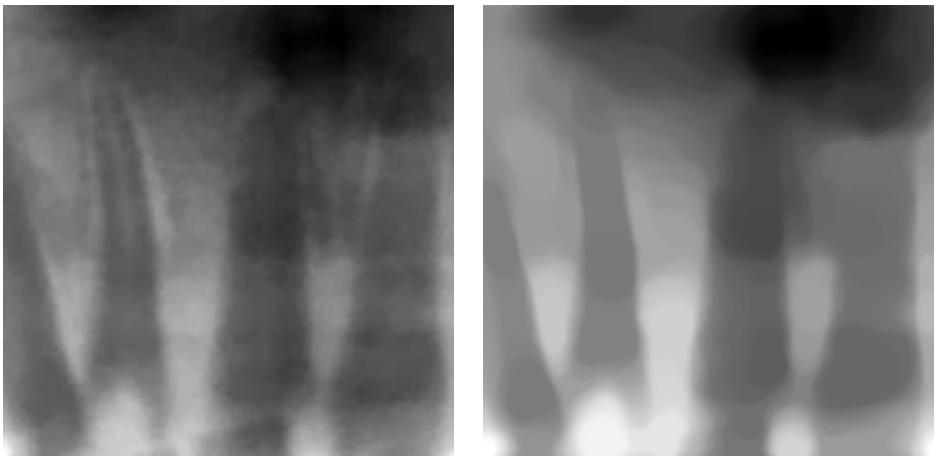
β_{opt} β_{disc}

Figure 4.16: Image13 after the stopping criterion has been reached.



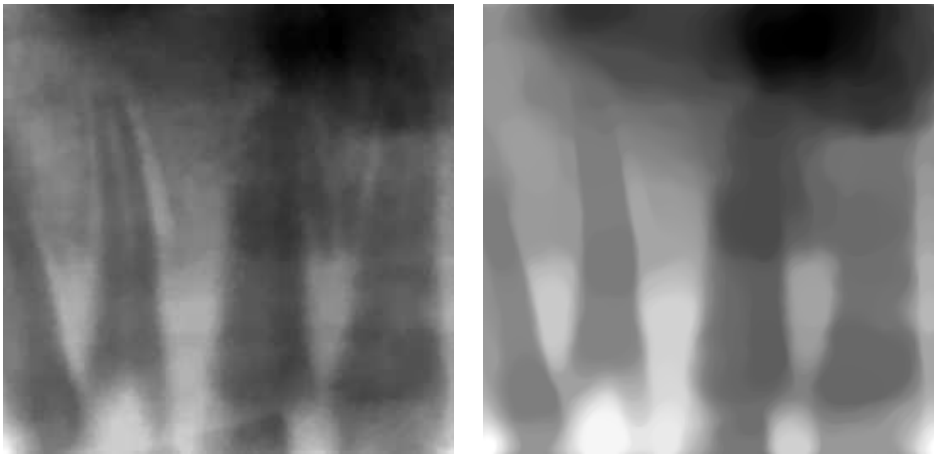
β_{opt} β_{disc}

Figure 4.17: Image22 after the stopping criterion has been reached.



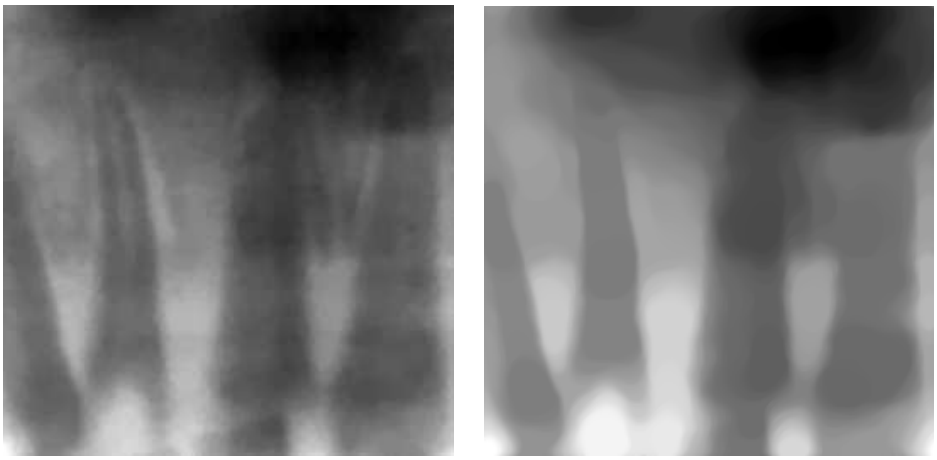
β_{opt} β_{disc}

Figure 4.18: Image48 after the stopping criterion has been reached.



β_{opt} β_{disc}

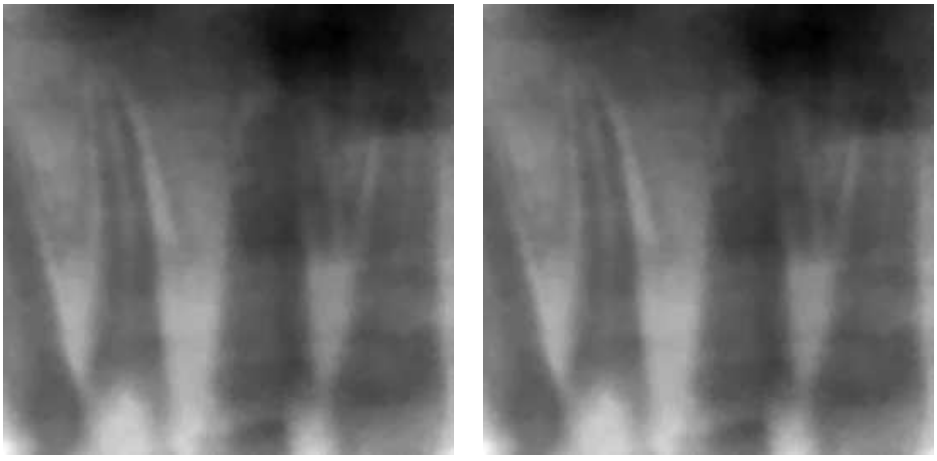
Figure 4.19: Image49 after the stopping criterion has been reached.



β_{opt} β_{disc}

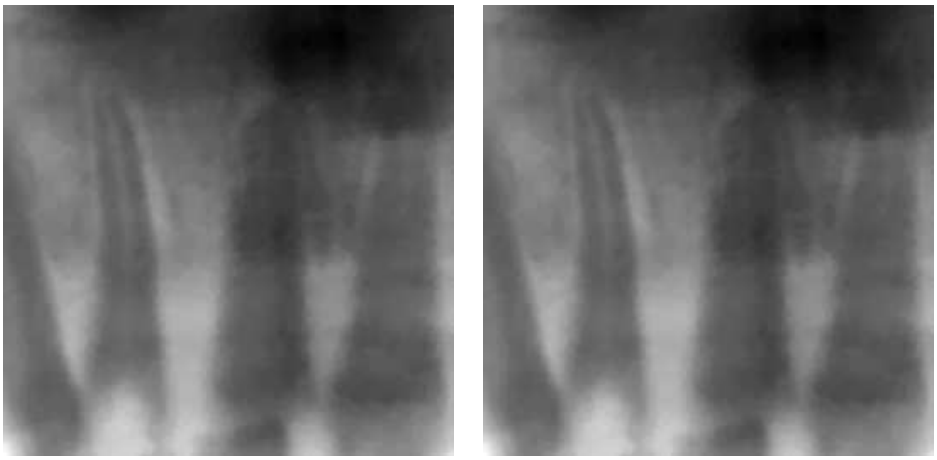
Figure 4.20: Image50 after the stopping criterion has been reached.

Below, instead, are shown the images resulting from the PIDSplit+ with the optimal value of γ (always $\frac{5}{\beta}$, as can be seen from the previous tables) and AEM, for $\beta = \beta_{opt}$:



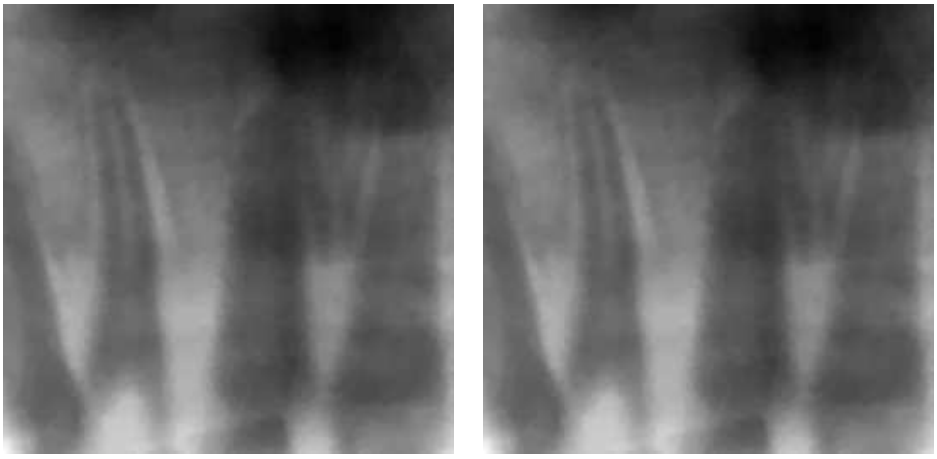
PID5 AEM

Figure 4.21: Image11 after the stopping criterion has been reached.



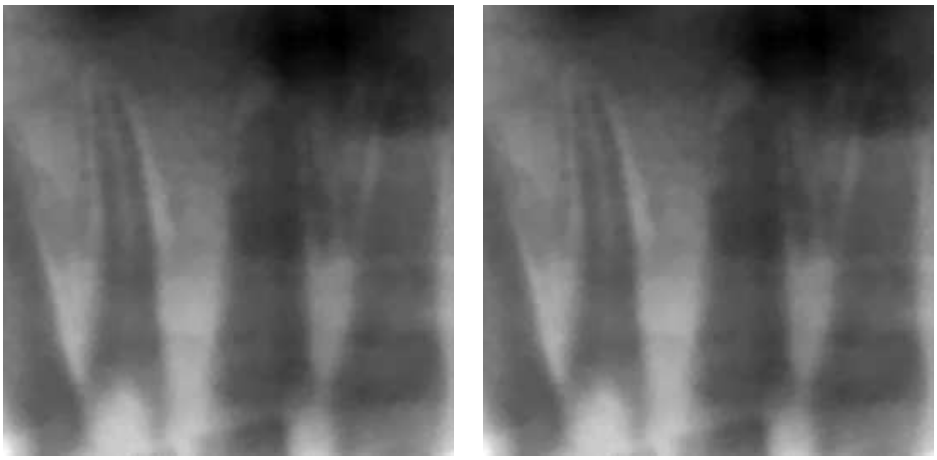
PID5 AEM

Figure 4.22: Image13 after the stopping criterion has been reached.



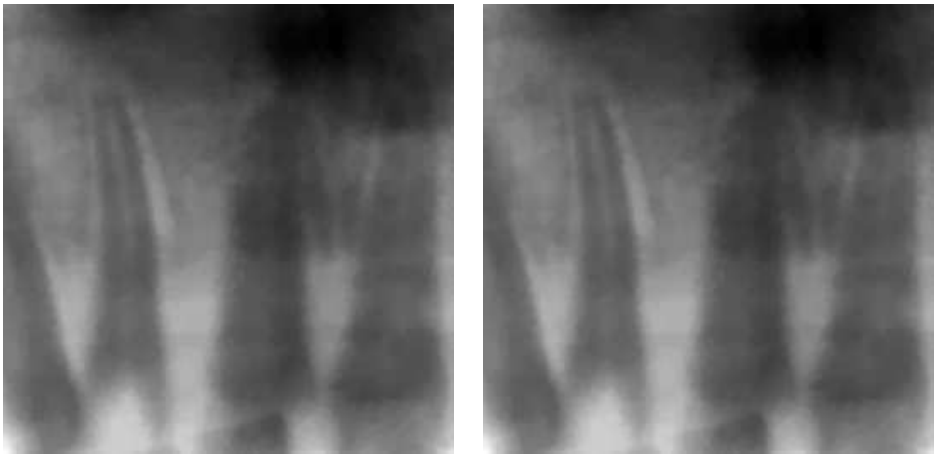
PID5 AEM

Figure 4.23: Image22 after the stopping criterion has been reached.



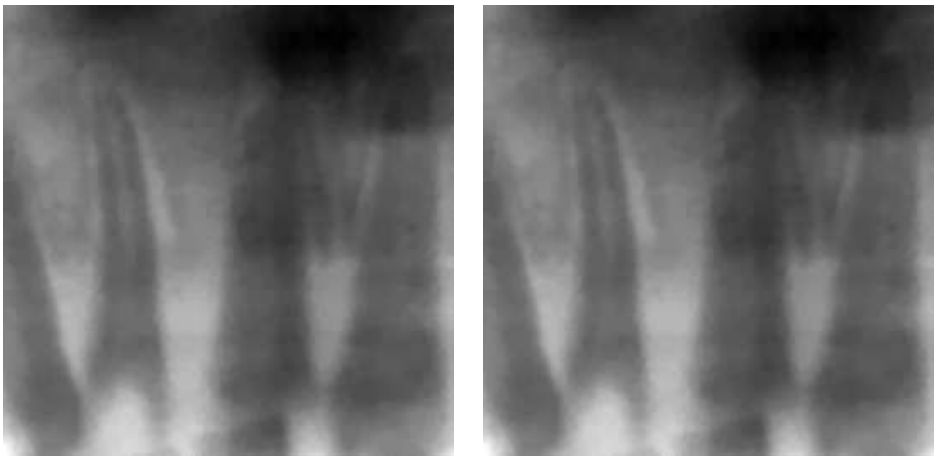
PID5 AEM

Figure 4.24: Image48 after the stopping criterion has been reached.



PID5 AEM

Figure 4.25: Image49 after the stopping criterion has been reached.



PID5 AEM

Figure 4.26: Image50 after the stopping criterion has been reached.

From the results of these experiments we can draw the following remarks.

- The stopping criterion (4.1) used for both the algorithms provides good results, considering also that the values of the primal function $f\left(x^{(k)}\right)$ using both AEM and PIDSplit+ (with the optimal value for γ), are comparable.
- From the results obtained with β_{opt} and β_{disc}, indipendently from the image considered, it can bee seen that the reconstruction obtained using β_{disc} seems to be excessively smoothed, with the loss of many details of the original image. For this reason, the adoption of such value for β seems advisable if the result required is to distinguish between different areas of the image. It appears that β_{disc} provide an overestimation for the order of magnitude of β.
- As seen before, the values of γ affects heavily the behavior of the PIDSplit+ algorithm, to the point that the stopping criterion (4.1) is sometimes not satisfied within the maximum number of iterations.
- As previously, an optimal choice for the user-supplied parameter γ provides good results even with the considered stopping criterion; in general AEM has a less efficient behavior than the optimal PIDSplit+, although with fairly similar values.

Appendix A

A.1 Notable matrix structures

A.1.1 Toepliz matrices

A *Toepliz matrix*, named after the mathematician Otto Toepliz, is a matrix in which each descending diagonal from left to right is costant, as in the following example:

$$\begin{pmatrix} a & b & c & d & e \\ f & a & b & c & d \\ g & f & a & b & c \\ h & g & f & a & b \\ i & h & g & f & a \end{pmatrix}$$

More generally, as a $n \times n$ matrix, a Toepliz matrix has the form:

$$T = \begin{pmatrix} a_0 & a_{-1} & a_{-2} & \cdots & \cdots & a_{-n+1} \\ a_1 & a_0 & a_{-1} & \ddots & \ddots & a_{-n+2} \\ a_2 & a_1 & a_0 & \ddots & \ddots & \vdots \\ \vdots & \ddots & \ddots & \ddots & a_{-1} & a_{-2} \\ \vdots & \ddots & \ddots & a_1 & a_0 & a_{-1} \\ a_{n-1} & \cdots & \cdots & a_2 & a_1 & a_0 \end{pmatrix}$$

T is uniquely determined by their diagonals, thus one could consider their position with respect to the main diagonal and name the elements

$$a_i \begin{cases} i = 0 & \text{the main diagonal} \\ i = 1 & \text{the first diagonal below} \\ i = -1 & \text{the first diagonal above} \\ \vdots & \vdots \end{cases}$$

In a block-Toepliz matrix each element is a matrix as well; if such element is a

Toepliz matrix, we have a block-Toepliz with Toepliz blocks matrix (BTTB).

An interesting properties of Toepliz matrices is that the convolution operation can be constructed as a matrix multiplication, where one of the inputs is converted into a Toeplitz matrix.

A.1.2 Fourier matrices

A Fourier matrix, which represent the Fourier transform, is a matrix of order n

$$F_n = \frac{1}{\sqrt{n}} \begin{pmatrix} 1 & 1 & 1 & \dots & 1 \\ 1 & w & w^2 & \dots & w^{n-1} \\ 1 & w^2 & w^4 & \dots & w^{2(n-1)} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & w^{n-1} & w^{2(n-1)} & \dots & w^{(n-1)(n-1)} \end{pmatrix}$$

where w is a n-th root of 1:

$$w = e^{-i\frac{2\pi}{n}} = \cos\left(\frac{2\pi}{n}\right) - i\sin\left(\frac{2\pi}{n}\right)$$

and i is the imaginary unit.

Since w^k, $k = 0, 1, ...$ is periodic with period n, F_n is reduced to:

$$F_n = \frac{1}{\sqrt{n}} \begin{pmatrix} 1 & 1 & 1 & \dots & 1 \\ 1 & w & w^2 & \dots & w^{n-1} \\ 1 & w^2 & w^4 & \dots & w^{n-2} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & w^{n-1} & w^{n-2} & \dots & w \end{pmatrix}$$

A *discrete Fourier transform* (DFT), denoted by $\mathcal{F}$, of a vector $f \in \mathbb{R}^n$ is the matrix-vector product

$$\mathcal{F}(f) = F_n f$$

Fourier matrices has interesting properties: if we denote the matrix conjugate traspose of F_n with F_n^*, we have $F_n^* = \overline{F_n}^T$,

F_n and F_n^* are symmetric and F_n is unitary, i.e.

$$F_n^* F_n = F_n F_n^* = I \qquad \text{or} \qquad F_n^{-1} = F_n^*$$

The DFT of a vector of n entries can be computed using a *fast Fourier transform* (FFT) algorithm with complexity $O(n \log_2 n)$ and similarly for the inverse FFT.

The Fourier transform of a 2-D $n \times n$ image $f(x, y)$ can then be expressed using the Kroenecker product of F_n:

$$(F_n \otimes F_n) vec(f)$$

where $vec(f)$ is an array of n^2 entries column-wise ordered.

If we consider the n matrix form of the image, we can show that

$$mat\Big((F_n \otimes F_n)\, vec(f)\Big) = F_n f F_n^T = F_n f F_n$$

A.1.3 Circulant matrices

A circulant matrix $n \times n$ is a particular Toepliz matrix of the form

$$C = \begin{pmatrix} c_0 & c_{n-1} & \dots & c_2 & c_1 \\ c_1 & c_0 & c_{n-1} & \dots & c_2 \\ \vdots & \ddots & \ddots & \ddots & \vdots \\ c_{n-2} & \ddots & c_1 & c_0 & c_{n-1} \\ c_{n-1} & c_{n-2} & \dots & c_1 & c_0 \end{pmatrix}$$

C is then completely determined by its first column

$$c = (c_0, c_1, \dots, c_{n-1})^T$$

If the elements of this matrix are matrices as well, we have a block-circulant matrix, if such matrices are circulant, we have a block-circulant with circulant blocks matrix (BCCB).

Any Circulant matrix C can be diagonalized by the Fourier matrix [21]:

$$C = F_n^* \Lambda F_n$$

where F_n is the Fourier matrix of order n, Λ is the diagonal matrix of the eigenvalues of C and F_n^* are the corresponding eigenvectors.

We observe that

$$F_n C = \Lambda F_n$$

and if we consider only the first column of the matrix equality, we have

$$F_n c = \frac{1}{\sqrt{n}} (\lambda_1, ..., \lambda_n)^T$$

since the first column of F_n has all entries equal to $\frac{1}{\sqrt{n}}$.

Then $diag\Lambda = \sqrt{n}\mathcal{F}(c)$

Similarly, in a Block Circulant matrix, each block is decomposed using the Fourier matrix F_n: $C_k = F_n^* \Lambda_k F_n$, where Λ_k is the diagonal matrix of the eigenvalues of each block C_k.

If we have a BCCB matrix $\mathbf{C}$, it is diagonalized[21]:

$$\mathbf{C} = (F_n \otimes F_n)^* \Lambda (F_n \otimes F_n)$$

where

$$\Lambda = \sum_{k=0}^{n-1} (\Omega_n^k \otimes \Lambda_k) \qquad \text{and} \qquad \Omega_n = diag(1, w, w^2, ..., w^{n-1})$$

This relationship between circulant matrices and Fourier matrices is very useful: if we consider $Cx = b$, where C is circulant, we can write such equation as $c * x = b$, where c is, as previously, the first column of C, $*$ denotes the convolution operator, and both x and b are cyclically extended in each direction.

Using the result of the circular convolution theorem, we can trasform the cyclic convolution into component-wise multiplication using the DFT:

$$\mathcal{F}(c * x) = \mathcal{F}(c)\mathcal{F}(x) = \mathcal{F}(b)$$

Then we have

$$x = \mathcal{F}^{-1}\left(\frac{\mathcal{F}(b)}{\mathcal{F}(c)}\right)$$

If we use the FFT algorithm, such equation is a very efficient way to compute x. If we have to compute the matrix product vector Cx we have

$$\mathcal{F}^{-1}(\mathcal{F}(c) \cdot \mathcal{F}(x)) = b$$

A.2 Constrained optimization

A general constrained optimization problem can be expressed as:

$$\underset{x \in \mathbb{R}^n}{\text{minimize}}\, f(x) \qquad \text{subject to} \begin{cases} c_i(x) = 0 \\ c_j(x) \geq 0 \end{cases} \tag{A.1}$$

where $c_i(x)$ and $c_j(x)$ (with $i \in \mathcal{E}, j \in \mathcal{I}$) are vectorial functions at least C^2, called, respectively, the *equality constraints* and *inequality constraints*. f is called the *objective function*.

Furthermore, we define the *feasible set* Ω to be the set of points x that satisfy the constraints, that is

$$\Omega = \{x | h_i(x) = 0 \ , \ g_i(x) \geq 0\}$$

With this notation (A.1) can be written more compactly as

$$\underset{x \in \Omega}{\text{minimize}} \, f(x)$$

The solution of (A.1) is a point of *local minimum*, defined more rigorously as

Definition A.1. *A point $x^* \in \mathbb{R}^n$ is a point of local minimum if $x^* \in \Omega$ and exists a neighborhood U of x^* such that $f(x) \geq f(x^*) \ \forall x \in U \cap \Omega$.*

A.2.1 First order necessary conditions

In order to give the first order necessary conditions, a couple of definitions and an assumption, the *constraint qualification*, is required, to foresee any degenerate behaviour of the function.

Definition A.2. *The* Lagrangian *function for the constrained optimization problem* (A.1) *is:*

$$\mathcal{L}(x, \lambda) = f(x) - \sum_{i \in \mathcal{E} \cup I} \lambda_i c_i(x)$$

Definition A.3. *The* active set $A(x)$ *at any feasible point x is the set*

$$\mathcal{A}(x^*) = \mathcal{E} \cup \{i \in \mathcal{I} | c_i(x^*) = 0\}$$

Definition A.4. *Given the point x^*, we say that the* linear indipendence constraint qualifications *(LICQ) holds if the set of active constraint gradient $\{\nabla c_i(x^*) | i \in \mathcal{A}(x^*)\}$ is linearly independent.*

This condition allows the statement of the following optimality conditions for a general nonlinear programming problem as in (A.1). They are called the first-order conditions because they concern the properties of the gradients of the objective and constraint functions.

Theorem A.1 (First-order necessary condition)**.** *Suppose that x^* is a local solution of* (A.1) *and that LICQ holds at x^*. Then there is a Lagrange multiplier vector λ^* with components λ_i^* $(i \in \mathcal{E} \cup \mathcal{I})$ such that the following conditions are satisfied at (x^*, λ^*):*

$$
\begin{aligned}
\nabla_x \mathcal{L}(x^*, \lambda^*) &= 0 \\
c_i(x^*) &= 0 \quad \forall i \in \mathcal{E} \\
c_i(x^*) &\geq 0 \quad \forall i \in \mathcal{I} \\
\lambda_i^* &\geq 0 \quad \forall i \in \mathcal{I} \\
\lambda_i^* c_i(x^*) &= 0 \quad \forall i \in \mathcal{E} \cup \mathcal{I}
\end{aligned}
\tag{A.2}
$$

Proof. See [22] p. 331. ◊

The conditions expressed in this theorem are often known as the *Karush-Kuhn-Tucker conditions* (KKT).

A.2.2 Second order necessary and sufficient conditions

Definition A.5. *Given a point x^* and the active constraint set $\mathcal{A}(x^*)$, the set F_1 is defined by:*

$$
F_1 = \left\{ \alpha d \;\middle|\; \begin{array}{ll} d^T \nabla c_i(x^*) = 0 & \forall i \in \mathcal{E} \\ \\ d^T \nabla c_i(x^*) \geq 0 & \forall i \in \mathcal{I} \end{array} \right\}
$$

Such set F_1 is a cone, and when a constraint qualification as LICQ is satisfied, it represent the *tangent cone* to the feasible set at x^*.

Definition A.6. *Given F_1 and the Lagrange multiplier vector λ^* satisfying the KKT conditions* (A.2)*, we define the subset $F_2(\lambda^*)$ of F_1 by:*

$$
F_2(\lambda^*) = \left\{ w \in F_1 \,\middle|\, w^T \nabla c_i(x^*) = 0 \qquad \forall i \in \mathcal{A}(x^*) \cap \mathcal{I} \;\; \textit{with} \;\; \lambda_i^* > 0 \right\}
$$

Such subset $F_2(\lambda^*)$ contains the directions w that tend to "adhere" to the active inequality constraints for which the Lagrange multiplier component $\lambda_i^* > 0$, as well as to the equality constraints. This set is important because contains directions from F_1 for which it is not clear from first derivative information alone whether f will increase or decrease.

Then, the necessary condition involving the second derivatives is described by the following theorem:

Theorem A.2 (Second-order necessary condition)**.** *Suppose that x^* is a local solution of* (A.1) *and that LICQ condition is satisfied. Let λ^* be a Lagrange multiplier vector such that the KKT conditions* (A.2) *are satisfied and let $F_2(\lambda^*)$ be defined as above.*

Then

$$w^T \nabla_{xx} \mathcal{L}(x^*, \lambda^*) w \geq 0 \qquad \forall w \in F_2(\lambda^*)$$

Proof. See [22], p. 343. ◊

The second-order sufficient condition looks very much like the necessary conditions just stated, but it differs in that the constraint qualification is not required and the inequality is replaced by a strict inequality.

Theorem A.3 (Second-order sufficient condition)**.** *Suppose that for some feasible point $x^* \in \mathbb{R}^n$ there is a Lagrange multiplier vector λ^* such that KKT conditions* (A.2) *are satisfied.*

Suppose also that

$$w^T \nabla_{xx} \mathcal{L}(x^*, \lambda^*) w > 0 \qquad \forall w \in F_2(\lambda^*), w \neq 0$$

Then x^* is a strict local solution for (A.1)*.*

Proof. See [22], p. 345. ◊

A.2.3 Convex optimization

Definition A.7. *A function $f : S \to \mathbb{R}$ is convex if $\forall x_1, x_2 \in S$ we have that*

$$f\Big((1-\rho)x_1 + \rho x_2\Big) \leq (1-\rho) f(x_1) + \rho f(x_2)$$

with $\rho \in (0, 1)$.

Convex functions are very important in numerical optimization, as shows the following theorem:

Theorem A.4. *A convex problem satisfies one of the following:*

- *there is not a solution to the minimum problem*
- *every local solution is a global solution*

Proof. By *reductio ad absurdum* suppose that exists a local solution x^* which is not a global solution, that is

$$\exists z \text{ such that } f(z) \leq f(x^*)$$

f is convex:

$$f\Big((1-\rho)x^* + \rho z\Big) \leq (1-\rho) f(x^*) + \rho f(z) = f(x^*) + \rho(f(z) - f(x^*))$$

Note that this last term $\rho\Big(f(z) - f(x^*)\Big) \leq 0$ since $\rho \neq 0$.

If $x = x^* + \rho(z - x^*) = (1 - \rho)x^* + \rho z$, we have that

$$f(x) \leq f(x^*) \Rightarrow \nexists \delta > 0 \text{ such that } f(x) \geq f(x^*)$$

that means x^* is not a local solution, which contradicts our supposition. $\Diamond$

In order to ensure the existence of a global minimum, we require the function to be coercive:

Definition A.8. *A function $f : \mathbb{R}^n \to \mathbb{R}^n$ is* coercive *if*

$$\lim_{||x|| \to \infty} f(x) = +\infty$$

Convex functions have other interesting properties:

Theorem A.5. *Let $f : R^n \to R^n$ be a convex function.*

Then $\nabla f(x)$ is a monotone function.

Proof. Since $f(x)$ is convex, we have that

$$f(x) \geq f(y) + \nabla f(y)^T (x - y)$$
$$f(y) \geq f(x) + \nabla f(x)^T (y - x)$$

Adding the two equations, we have:

$$(\nabla f(y) - \nabla f(x))^T (y - x) \geq 0$$

$\Diamond$

Theorem A.6. *Let $f(x, y)$ be convex with respect to x and concave with respect to y.*

Then

$$\begin{pmatrix} \nabla_x f(x, y) \\ -\nabla_y f(x, y) \end{pmatrix}$$

is monotone.

Proof. Since

$$f(x_i, y_i) \geq f(x'_i, y_i) + \nabla_x f(x'_i, y_i)(x_i - x'_i)$$
$$f(x_i, y_i) \geq f(x_i, y'_i) + \nabla_y f(x_i, y'_i)(y_i - y'_i)$$

we have that

$$(\nabla_x f(x_1, y_1) - \nabla_x f(x_2, y_2))^T (x_1 - x_2) + (\nabla_y f(x_1, y_2) +$$
$$- \nabla_y f(x_2, y_1))^T (y_1 - y_2) \geq 0$$

$\Diamond$

Appendix B

Matlab scripts

In this chapter we will provide the MATLAB scripts used throughout this thesis, in alphabetical order:

B.1 AEM.m

```
function [x,y1,y2,y3,TimeCost,InnVec,Primal,phi_xy,alpha_vec,err,...
    varargout] = ...
    AEM(z, H, HT,bg, beta, delta, eta, grad1, grad2, div, NIT, ...
    tol, verbose, Nalpha, epsilon, obj)

%Nalpha = -1 per non tenere memoria%
%epsilon = 1e-4;%5e-1; %provare
% obj is a cell array of images to compute the relative difference

nobj = length(obj);
for i = 1:nobj
    normobj(i) = norm(obj{i}(:));
    err{i}     = zeros(NIT+1,1); %arrays to store errors per iteration
end

Primal = zeros(NIT+1,1);
phi_xy = zeros(NIT+1,1);
alpha_vec = zeros(NIT+1,1);
InnVec = zeros(NIT+1,1);
TimeCost = zeros(NIT+1,1);
kkt_vec  = zeros(NIT+1,1);
KL_vec   = zeros(NIT+1,1);

gamma   = 0.99;
theta   = 0.99;
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%


totred = 0;
n=length(z);                %Assume a square image
zeroindex = z <= 0;
nonzeroindex = ~zeroindex;
```

```
rapp = zeros(size(z));

%%% Initial point %%
x  = max(eta,z);
y1 = zeros(size(z));
y2 = y1;
y3 = zeros(size(z));

%%% Projection of the initial x
ynorm= max(1, sqrt(y1.^2+y2.^2+y3.^2));
y1 = y1./ynorm;
y2 = y2./ynorm;
y3 = y3./ynorm;
% periodic boundary conditions
dx1 = grad1(x);
dx2 = grad2(x);

gu_norm = sqrt(dx1.^2+dx2.^2+delta^2); %total variarion
den = H(x) + bg;%ifft2(fft2(y).*TF)+bg;
rapp(nonzeroindex) = z(nonzeroindex)./ den(nonzeroindex);
KL = sum( z(nonzeroindex).* log(rapp(nonzeroindex)) + ...
    den(nonzeroindex) - z(nonzeroindex) );
KL = KL + sum(den(zeroindex));
g_KL = 1 - HT(rapp);%1 - ifft2(conj(TF).*fft2(rapp));

Ay = div(y1,y2);
g = g_KL + beta*Ay;

Primal(1) = beta*sum(sum(gu_norm)) + KL;
phi_xy(1) = beta*sum(sum(Ay.*x))+ beta*delta* sum(sum(y3)) +KL;
KL_vec(1) = KL;
for i=1:nobj
    err{i}(1) = norm(obj{i}(:)-x(:))/normobj(i);
end

if verbose
    fprintf('\nInitial: Primal=%8.3e', Primal(1));
    for i=1:nobj
        fprintf(' err{%g} %g', i, err{i}(1));
    end

end
alpha = 1;
alphaOK(1) = alpha;
alpha_vec(1) = alpha;
TimeCost(1)=0;
t0 = cputime;                %Start CPU clock

for itr=1:NIT

    y1old = y1; y2old = y2; y3old = y3;

    y1 = y1 + alpha*beta*dx1;
    y2 = y2 + alpha*beta*dx2;
    y3 = y3 + alpha*beta*delta;
    % Projection on the set X={||x||<=1}
    ynorm= max(1, sqrt(y1.^2+y2.^2+y3.^2));
    y1 = y1./ynorm;
    y2 = y2./ynorm;
    y3 = y3./ynorm;
    Ay = div(y1,y2);
```

```
xold = x;
g_KLold = g_KL;
dx1old = dx1; dx2old = dx2;

x = x - alpha*( g_KL + beta*Ay );
x = max(x,eta);

dx1 = grad1(x);
dx2 = grad2(x);

den  = H(x) + bg;
rapp(nonzeroindex) = z(nonzeroindex)./ den(nonzeroindex);
g_KL = 1-HT(rapp);

Deltax = x - xold; normDeltax = norm(Deltax(:));
Ak = norm(g_KLold(:) - g_KL(:)) / normDeltax;
Bk = beta*sqrt( norm(dx1(:)-dx1old(:))^2 + norm(dx2(:)-...
    dx2old(:))^2 )/normDeltax;
cond = 1-2*alpha*Ak-2*alpha^2*Bk^2;
%fprintf('\n L1=%g L3=%g cond %g\n',L1,L3,cond);
alphabar = gamma*(sqrt(Ak^2+2*Bk^2*(1-epsilon))-Ak)/(2*Bk^2);
ired = 0;
while cond < epsilon
    alpha = min(alphabar,theta*alpha);
    y1 = y1old + alpha*beta*dx1old;
    y2 = y2old + alpha*beta*dx2old;
    y3 = y3old + alpha*beta*delta;
    % Projection on the set X={||x||<=1}
    ynorm= max(1, sqrt(y1.^2+y2.^2+y3.^2));
    y1 = y1./ynorm;
    y2 = y2./ynorm;
    y3 = y3./ynorm;
    Ay = div(y1,y2);

    x = xold - alpha*( g_KLold + beta*Ay );
    x = max(x,eta);

    dx1 = grad1(x);
    dx2 = grad2(x);

    den  = H(x)+bg;
    rapp(nonzeroindex) = z(nonzeroindex)./ den(nonzeroindex);
    g_KL = 1-HT(rapp);

    Deltax = x - xold; normDeltax = norm(Deltax(:));
    Ak = norm(g_KLold(:) - g_KL(:)) / normDeltax;
    Bk = beta*sqrt( norm(dx1(:)-dx1old(:))^2 + norm(dx2(:)-...
        dx2old(:))^2 )/normDeltax;
    cond = 1-2*alpha*Ak-2*alpha^2*Bk^2;
    alphabar = gamma*(sqrt(Ak^2+2*Bk^2*(1-epsilon))-...
        Ak)/(2*Bk^2);
    ired = ired + 1;
    %            return
end
alpha_vec(itr + 1) = alpha;
totred = totred + ired;
InnVec(itr + 1) = ired;
y1 = y1old + alpha*beta*dx1;
y2 = y2old + alpha*beta*dx2;
y3 = y3old + alpha*beta*delta;
% Projection on the set X={||x||<=1}
ynorm = max(1, sqrt(y1.^2+y2.^2+y3.^2));
```

```
    y1 = y1./ynorm;
    y2 = y2./ynorm;
    y3 = y3./ynorm;
    Ay = div(y1,y2);
    KL = sum( z(nonzeroindex).* log(rapp(nonzeroindex)) + ...
        den(nonzeroindex) - z(nonzeroindex) );
    KL = KL + sum(den(zeroindex));

    gu_norm = sqrt(dx1.^2+dx2.^2+delta^2);
    alphaOK(itr) = alpha;
    alpha = mean([alphaOK(max(1,itr-Nalpha):itr) alphabar]);


    Primal(itr+1) = beta*sum(sum(gu_norm)) + KL;
    phi_xy(itr+1) = beta*sum(sum(Ay.*x))+ beta*delta* ...
        sum(sum(y3)) + KL ;
    KL_vec(itr+1) = KL;
    TimeCost(itr+1)=cputime-t0;

    for i=1:nobj
        err{i}(itr + 1) = norm(obj{i}(:)-x(:))/normobj(i);
    end

    if verbose
        fprintf('\n%4d): f(x)=%g Phi(x,y)=%g alpha %g', itr, ...
            Primal(itr+1), phi_xy(itr+1), alpha );
        for i=1:nobj
            fprintf(' err{%g} %g', i, err{i}(itr + 1));
        end
    end
    normDeltay =sqrt(sum((y1(:)-y1old(:)).^2) + sum((y2(:)-...
        y2old(:)).^2) + sum((y3(:)-y3old(:)).^2));
    normy = sqrt(sum(y1(:).^2) + sum(y2(:).^2) + sum(y3(:).^2));
    kkt_vec(itr+1) = sqrt(normDeltax^2+normDeltay^2)/...
        sqrt(norm(x(:))^2+normy^2);
    if  kkt_vec(itr+1) < tol
        break;
    end
end
%end of the main loop
Primal(itr+2:end) = [];
for i = 1:nobj
    err{i}(itr + 2:end) = [];
end
phi_xy(itr+2:end) = [];
alpha_vec(itr+2:end) = [];
InnVec(itr+1:end) = [];
TimeCost(itr+2:end) = [];
if nargout >= 11
    varargout{1} = kkt_vec(1:itr+1);
    if nargout == 12
        varargout{2} = KL_vec(1:itr +1);
    end
end
if verbose
    fprintf('\n');
end
```

B.2 app.m

```
function app(dir,im,beta)
% app(dir,im,beta)
% computes the AEM and the PID50,PID5,PID1,PID05
% on the given image in the given directory with the given beta
% and saves the results in an appropriate .mat file

load([dir '/' im]);
load([dir '/' im '_noisy']);
load([dir '/' 'sol' im]);

[n,n] = size(gn);

grad1  = @(y)[y(:,2:n)-y(:,1:n-1), y(:,1)-y(:,n)];
grad2  = @(y)[y(2:n,:)-y(1:n-1,:); y(1,:)-y(n,:)];
div    = @(x1,x2)([-x1(:,1)+x1(:,n),-x1(:,2:n)+x1(:,1:n-1)] + ...
    [-x2(1,:)+x2(n,:);-x2(2:n,:)+x2(1:n-1,:)]);
H = @(x)x;
HT = @(x)x;

%scale = 2/(r^2);

bg = 0;
delta = 0;
NIT = 3000;
tol = 1e-6;

zeroindex =gn <= 0;
nonzeroindex = ~zeroindex;
eta = min(gn(nonzeroindex))*ones(size(gn));
eta(zeroindex) = 0;

verbose=true;
Nalpha=10;
epsilon = 1e-4;

benchsol={obj,x};

TF = ones(size(obj));

tic;

[x,y1,y2,y3,TimeCost,InnVec,Primal,phi_xy,alpha_vec,err,KKT,...
    KL] = AEM(gn, H, HT,bg, beta, delta, eta, grad1, grad2, ...
    div, NIT, tol,verbose, Nalpha, epsilon, benchsol);
save([dir '/' 'data_' im]);
clear x y1 y2 y3 TimeCost InnVec Primal phi_xy alpha vec err KKT KL;

gamma = 50/beta;
[u_50,w3_50,TimeCost_50,fobj_50,err_50,w2_y_50,b2_x_50,...
    b2_y_50,kkterr_50]  = PIDSplit_plus(gn, TF, H, HT, bg,...
    beta, gamma, ...
    eta, grad1, grad2, div, NIT, tol, verbose, benchsol);
save([dir '/' 'data_' im '_PID_50']);
clear u_50 w3_50 TimeCost_50 fobj_50 err_50 w2_y_50 b2_x_50...
    b2_y_50 kkterr_50;

gamma = 5/beta;
[u_5,w3_5,TimeCost_5,fobj_5,err_5,w2_y_5,b2_x_5,b2_y_5, ...
    kkterr_5]=PIDSplit_plus(gn, TF, H, HT, bg, beta, gamma, ...
    eta, grad1,grad2, div, NIT, tol, verbose, benchsol);
save([dir '/' 'data_' im '_PID_5']);
clear u_5 w3_5 TimeCost_5 fobj_5 err_5 w2_y_5 b2_x_5 ...
```

```
    b2_y_5 kkterr_5;

gamma = 1/beta;
[u_1,w3_1,TimeCost_1,fobj_1,err_1,w2_y_1,b2_x_1,b2_y_1,...
    kkterr_1]=PIDSplit_plus(gn, TF, H, HT, bg, beta, gamma, ...
    eta, grad1,grad2, div, NIT, tol, verbose, benchsol);
save([dir '/' 'data_' im '_PID_1']);
clear u_1 w3_1 TimeCost_1 fobj_1 err_1 w2_y_1 b2_x_1 b2_y_1 ...
    kkterr_1;

gamma = 0.5/beta;
[u_05,w3_05,TimeCost_05,fobj_05,err_05,w2_y_05,b2_x_05, ...
    b2_y_05,kkterr_05]  = PIDSplit_plus(gn, TF, H, HT, bg, ...
    beta, gamma,eta, grad1, grad2, div, NIT, tol,  ...
    verbose, benchsol);
save([dir '/' 'data_' im '_PID_05']);

fprintf('Time elapsed: %g seconds',toc);
%exit;
```

B.3 beta_array.m

```
% trial of given values for beta for regularization
% using AEM
clear all;

for k=[11,13,22,48,49,50]

    image = ['PRISMA/Image' int2str(k) '.tif'];
    gn = 4095-double(imread(image));
    gn = gn(:,138:404);

    obj = 4095-double(imread('PRISMA/Avg.tif'));
    obj = obj(:,138:404);

    [r,r] = size(gn);
    grad1  = @(y)[y(:,2:r)-y(:,1:r-1), y(:,1)-y(:,r)];
    grad2  = @(y)[y(2:r,:)-y(1:r-1,:); y(1,:)-y(r,:)];
    div    = @(x1,x2)([-x1(:,1)+x1(:,r),-x1(:,2:r)+x1(:,1:r-1)] + ...
        [-x2(1,:)+x2(r,:);-x2(2:r,:)+x2(1:r-1,:)]);
    H = @(x)x;
    HT = @(x)x;

    scale = 2/(r^2);

    bg = 0;
    delta = 0;
    NIT = 3000;
    tol = 1e-10; %% very small so NIT is reached

    zeroindex = gn <= 0;
    nonzeroindex = ~zeroindex;
    eta = min(gn(nonzeroindex))*ones(size(gn));
    eta(zeroindex) = 0;

    verbose =false;
    Nalpha=10;
    epsilon = 1e-4;

    benchsol={obj};
```

```
    out = [];

    tolbeta = 1e-3;
    zeroindex = gn <= 0;
    nonzeroindex = ~zeroindex;
    rapp = zeros(size(gn));
    den = H(obj)+bg;
    rapp(nonzeroindex) = gn(nonzeroindex)./ den(nonzeroindex);

    KLstima = sum( gn(nonzeroindex).* log(rapp(nonzeroindex)) ...
        + den(nonzeroindex) - gn(nonzeroindex) );
    KLstima = KLstima + sum(den(zeroindex));
    KLstima=KLstima*scale;
    fprintf('KLSTIMA=%g\n',KLstima);

    %%% choice of beta
    beta_start = 0.01;
    beta_end = 0.02;
    beta=beta_start:0.01:beta_end;

    %sol=zeros(r,r,length(beta));

    for i=1:length(beta)
        [x,y1,y2,y3,TimeCost,InnVec,Primal,phi_xy,alpha_vec,...
            err,KKT,KL] = AEM(gn, H, HT,bg, beta(i), delta,...
            eta, grad1, grad2,div, NIT, tol, verbose, Nalpha,...
            epsilon, benchsol);
        %ratio = (KL(end)*scale)-1;
        %sol(:,:,i)=x;
        TV=(phi_xy(end)-KL(end))/beta(i);
        out = [out; beta(i), err{1}(end), KL(end), ...
            phi_xy(end),TV];
        fprintf('beta: %g, err:%g, KL: %g  TV=%g\n',beta(i),...
            err{1}(end),KL(end),TV);
    end

    %format long;
    %disp(out);
    save([ 'PRISMA/array_beta_im' int2str(k)]);

end
exit;
```

B.4 beta_bisez.m

```
% bisection method to solve the nonlinear
% equation in beta for the regularization
clear all;

for i=[11,13,22,48,49,50]

    image = ['PRISMA/Image' int2str(i) '.tif'];
    gn = 4095-double(imread(image));
    gn = gn(:,138:404);

    obj = 4095-double(imread('PRISMA/Avg.tif'));
    obj = obj(:,138:404);

    [n,n] = size(gn);
```

```
grad1  = @(y)[y(:,2:n)-y(:,1:n-1), y(:,1)-y(:,n)];
grad2  = @(y)[y(2:n,:)-y(1:n-1,:); y(1,:)-y(n,:)];
div    = @(x1,x2)([-x1(:,1)+x1(:,n),-x1(:,2:n)+...
    x1(:,1:n-1)] +[-x2(1,:)+x2(n,:);-x2(2:n,:)+...
    x2(1:n-1,:)]);
H = @(x)x;
HT = @(x)x;

scale = 2/(n^2);

bg = 0;
delta = 0;
NIT = 3000;
tol = 1e-10; %% very small so NIT is reached

zeroindex = gn <= 0;
nonzeroindex = ~zeroindex;
eta = min(gn(nonzeroindex))*ones(size(gn));
eta(zeroindex) = 0;

verbose =false;
Nalpha=10;
epsilon = 1e-4;

benchsol={obj};

out = [];

tolbeta = 1e-3;

beta_start = 0.001;
beta_end = 0.5;

zeroindex = gn <= 0;
nonzeroindex = ~zeroindex;
rapp = zeros(size(gn));
den = H(obj)+bg;
rapp(nonzeroindex) = gn(nonzeroindex)./ den(nonzeroindex);

KLstima = sum( gn(nonzeroindex).* log(rapp(nonzeroindex))...
    + den(nonzeroindex) - gn(nonzeroindex) );
KLstima = KLstima + sum(den(zeroindex));
KLstima=KLstima*scale;

fprintf('KLstima=%g\n',KLstima);

tic;

% first and last values are computed to initialize the
% bisection loop
[x,y1,y2,y3,TimeCost,InnVec,Primal,phi_xy,alpha_vec,err,...
    KKT,KLs]=AEM(gn, H, HT,bg, beta_start, delta, eta, ...
    grad1, grad2, div,...
    NIT, tol, verbose, Nalpha, epsilon, benchsol);
ratio_start = (KLs(end)*scale)-1;
out = [out; beta_start, err{1}(end), KLs(end), ...
    phi_xy(end), KLs(end)*scale];
[x,y1,y2,y3,TimeCost,InnVec,Primal,phi_xy,alpha_vec,err,...
    KKT,KLe]=AEM(gn, H, HT,bg, beta_end, delta, eta,...
    grad1, grad2, div, ...
    NIT, tol, verbose, Nalpha, epsilon, benchsol);
```

```
    ratio_end = (KLe(end)*scale)-1;
    out = [out; beta_end, err{1}(end), KLe(end), ...
        phi_xy(end),KLe(end)*scale];

    ratio = ratio_start;

    while(abs(beta_start-beta_end)>=tolbeta)
        beta_half = (beta_start+beta_end)/2;
        [x,y1,y2,y3,TimeCost,InnVec,Primal,phi_xy,alpha_vec,...
            err,KKT,KL]=AEM(gn, H, HT,bg, beta_half, delta,...
            eta, grad1, grad2, div,...
            NIT, tol, verbose, Nalpha, epsilon, benchsol);
        ratio_half = (KL(end)*scale)-1;
        out = [out; beta_half, err{1}(end), KL(end),...
            phi_xy(end),ratio_half];
        fprintf('beta_start:%g, beta_half:%g, beta_end:%g\n',...
            beta_start,beta_half,beta_end);
        fprintf('ratio_start:%g,ratio_half:%g,ratio_end:%g\n\n',...
            ratio_start,ratio_half,ratio_end);
        if (ratio_half * ratio_end < 0)
            beta_start = beta_half;
            ratio_start = ratio_half;
        elseif(ratio_start * ratio_half < 0)
            beta_end = beta_half;
            ratio_end = ratio_half;
        else
            break;
        end
        ratio = ratio_half;

    end

    fprintf('For Image %g \n',i);
    fprintf('Optimal beta value:%g,with a tolerance of:%g\n',...
        beta_half,tolbeta);
    fprintf('time elapsed: %g seconds \n', toc);

    save([ 'PRISMA/beta_im' int2str(i)]);

end
exit;
```

B.5 iter_AEM.m

```
function iter_AEM(dir,im,beta,NIT)
% iter_AEM(dir,im,beta,NIT)
%
% AEM for the given image im in the given directory dir
% with the given beta, with NIT iterations

load([dir '/' im]);
load([dir '/' im '_noisy']);
load([dir '/' 'sol' im]);

[n,n] = size(gn);

grad1  = @(y)[y(:,2:n)-y(:,1:n-1), y(:,1)-y(:,n)];
grad2  = @(y)[y(2:n,:)-y(1:n-1,:); y(1,:)-y(n,:)];
div    = @(x1,x2)([-x1(:,1)+x1(:,n),-x1(:,2:n)+x1(:,1:n-1)] + ...
    [-x2(1,:)+x2(n,:);-x2(2:n,:)+x2(1:n-1,:)]);
```

```
H = @(x)x;
HT = @(x)x;

bg = 0;
delta = 0;
tol = 0;

zeroindex =gn <= 0;
nonzeroindex = ~zeroindex;
eta = min(gn(nonzeroindex))*ones(size(gn));
eta(zeroindex) = 0;

verbose=false;
Nalpha=10;
epsilon =1e-4;

benchsol={obj,x};

[u,y1,y2,y3,TimeCost,InnVec,Primal,phi_xy,alpha_vec,err,KKT,KL]=...
    AEM(gn, H, HT,bg, beta, delta, eta, grad1, grad2, div, NIT,...
    tol, verbose, Nalpha, epsilon, benchsol);

save([dir '/' 'data_'  im '_it_' int2str(NIT)]);
```

B.6 iter_PID.m

```
function iter_PID(dir,im,beta,id,NIT)
% iter_PID(dir,im,beta,id,NIT)
%
% PIDSplit+ for the given image im in the given directory dir
% with the given beta, with NIT iterations
% accepts '50' '5' '1' '05' as id.

load([dir '/' im]);
load([dir '/' im '_noisy']);
load([dir '/' 'sol' im]);
fprintf('Working on %s with PID %s\n',im,id);

[n,n] = size(gn);

grad1  = @(y)[y(:,2:n)-y(:,1:n-1), y(:,1)-y(:,n)];
grad2  = @(y)[y(2:n,:)-y(1:n-1,:); y(1,:)-y(n,:)];
div    = @(x1,x2)([-x1(:,1)+x1(:,n),-x1(:,2:n)+x1(:,1:n-1)] + ...
    [-x2(1,:)+x2(n,:);-x2(2:n,:)+x2(1:n-1,:)]);
H = @(x)x;
HT = @(x)x;

bg = 0;
delta = 0;
tol = 0;

zeroindex =gn <= 0;
nonzeroindex = ~zeroindex;
eta = min(gn(nonzeroindex))*ones(size(gn));
eta(zeroindex) = 0;

verbose=false;

benchsol={obj,x};
TF = ones(size(obj));
```

```
if strcmp(id,'50')
    gamma = 50/beta;
    [u_50,w3_50,TimeCost_50,fobj_50,err_50,w2_y_50,b2_x_50,...
        b2_y_50,kkterr_50]=PIDSplit_plus(gn,TF,H,HT,bg,beta,...
        gamma,eta,grad1,grad2,div,NIT,tol,verbose,benchsol);
    save([dir '/' 'data_'  im '_PID_' id '_it_' int2str(NIT)]);
    clear u_50 w3_50 TimeCost_50 fobj_50 err_50 w2_y_50 b2_x_50 ...
        b2_y_50 kkterr_50;

elseif strcmp(id,'5')
    gamma = 5/beta;
    [u_5,w3_5,TimeCost_5,fobj_5,err_5,w2_y_5,b2_x_5,b2_y_5,...
        kkterr_5]  = PIDSplit_plus(gn, TF, H, HT, bg, beta, ...
        gamma, eta, grad1, grad2, div, NIT, tol, verbose, benchsol);
    save([dir '/' 'data_'  im '_PID_' id '_it_' int2str(NIT)]);
    clear u_5 w3_5 TimeCost_5 fobj_5 err_5 w2_y_5 b2_x_5 ...
        b2_y_5 kkterr_5;

elseif strcmp(id,'1')
    gamma = 1/beta;
    [u_1,w3_1,TimeCost_1,fobj_1,err_1,w2_y_1,b2_x_1,b2_y_1,...
        kkterr_1]  = PIDSplit_plus(gn, TF, H, HT, bg, beta, ...
        gamma, eta, grad1, grad2, div, NIT, tol, verbose, benchsol);
    save([dir '/' 'data_'  im '_PID_' id '_it_' int2str(NIT)]);
    clear u_1 w3_1 TimeCost_1 fobj_1 err_1 w2_y_1 b2_x_1 ...
        b2_y_1 kkterr_1;
elseif strcmp(id,'05')
    gamma = 0.5/beta;
    [u_05,w3_05,TimeCost_05,fobj_05,err_05,w2_y_05,b2_x_05,...
        b2_y_05,kkterr_05]=PIDSplit_plus(gn, TF, H, HT, bg, beta,...
        gamma,eta,grad1,grad2,div,NIT,tol,verbose, benchsol);
    save([dir '/' 'data_'  im '_PID_' id '_it_' int2str(NIT)]);
    clear u_05 w3_05 TimeCost_05 fobj_05 err_05 w2_y_05 b2_x_05 ...
        b2_y_05 kkterr_05;
end
```

B.7 PIDSplit_plus.m

```
function [u,w3,TimeCost,fobj,err,varargout] = ...
    PIDSplit_plus(f, TF, K, KT, bg, beta, gamma, eta, grad1,...
    grad2, div, NIT, tol, verbose, obj)

%Input parameters
%
%f              data
%TF             Fourier transform of the PSF
%K, KT          function handles to the matrix–vector products
%               by K and K^T
%bg             background constant term
%beta           regularization parameter
%eta            lower bound
%grad1, grad2   function handles to the gradient with respect to
%               the x and y directions
%div            function handle to the negative divergence operator
%NIT            maximum number of iterations
%tol            tolerance on the distance between two
%               successive iterates
%verbose        flag (=1 print information at each iteration;
%                    =0 no printing)
```

```
%obj            cell array containing reference images

%Output parameters
%
%u,w3           computed reconstructions (u may have negative
%               components)
%TimeCost       time per iteration (seconds)
%fobj           objective function value per iteration
%err            cell array containing relative errors per
%               iteration with respect
%               to each of the reference images in obj

% Code by Silvia Bonettini
% Edited by Davide Taviani


nobj = length(obj);
for i = 1:nobj
    normobj(i) = norm(obj{i}(:));
    err{i} = zeros(NIT+1,1); %arrays to store errors per iteration
end

fobj = zeros(NIT+1,1);
TimeCost = zeros(NIT+1,1);
kkterr=zeros(NIT+1,1);
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%


n=length(f);                %Assume a square image
zeroindex = f <= 0;
nonzeroindex = ~zeroindex;

rapp = zeros(size(f));

%%%Eigenvalues of the Laplacian operator
P = zeros(n);
P(n/2,n/2+1) = 1; P(n/2+1,n/2+1) = -1;
PTF = fft2(fftshift(P));
PTF2 = abs(PTF).^2;
Q   = P';
QTF = fft2(fftshift(Q));
QTF2 = abs(QTF).^2;
TF2  = abs(TF).^2;


%%% Initial points %%
b1 = zeros(n);
b2_x = zeros(n);
b2_y = zeros(n);
b3 = zeros(n);
w1 = K(f);
w2_x = grad1(f);%[f(:,2:n)-f(:,1:n-1), f(:,1)-f(:,n)];
w2_y = grad2(f);%[f(2:n,:)-f(1:n-1,:); f(1,:)-f(n,:)];
w3  = f;
u = f;

Ku  = K(w3);
den = Ku + bg;
rapp = zeros(n);
rapp(nonzeroindex) = f(nonzeroindex)./den(nonzeroindex);
KL = sum( f(nonzeroindex).* log(rapp(nonzeroindex)) + ...
```

```
    den(nonzeroindex) - f(nonzeroindex) );
KL = KL + sum(den(zeroindex));
gu_norm = sqrt(w2_x.^2+w2_y.^2);
fobj(1)   = KL + beta*sum(sum(gu_norm));

for i=1:nobj
    err{i}(1) = norm(obj{i}(:)-w3(:))/normobj(i);
end
if verbose
    fprintf('\nInitial: fobj=%8.3e', fobj(1));
    for i=1:nobj
        fprintf(' err{%g} %g', i, err{i}(1));
    end
end

TimeCost(1)=0;
t0 = cputime;                %Start CPU clock

for itr=1:NIT
    % Save old values
    uold = u;
    w1old = w1;
    w2_xold = w2_x;
    w2_yold = w2_y;
    w3old = w3;

    b1old = b1;

    b2_xold = b2_x;
    b2_yold = b2_y;

    b3old = b3;

    %Compute u^{(k+1)}
    t1 = KT( w1 - b1 );
    x1 = w2_x - b2_x; x2 = w2_y - b2_y;
    t2 = div(x1,x2);%([-x1(:,1)+x1(:,n),-x1(:,2:n)+ ...
                    %x1(:,1:n-1)]+[-x2(1,:)+x2(n,:);...
                    %-x2(2:n,:)+x2(1:n-1,:)]);
    t3 = w3 - b3;
    t = t1 + t2 + t3;

    u = ifft2(fft2(t1+t2+t3)./(1+TF2+PTF2+QTF2));


    %Update w1
    Ku = K(u) + bg;
    w1 = 0.5*( b1 + Ku - gamma + sqrt( ( b1 + Ku - ...
        gamma ).^2 + 4*gamma*f ) );

    %Update w2
    Du_x  = grad1(u);%[u(:,2:n)-u(:,1:n-1), u(:,1)-u(:,n)];
    Du_y  = grad2(u);%[u(2:n,:)-u(1:n-1,:); u(1,:)-u(n,:)];
    w2_x = b2_x + Du_x;
    w2_y = b2_y + Du_y;
    % shrink
    norm_w2 = sqrt(w2_x.^2 + w2_y.^2);
    ij  = norm_w2 >= gamma*beta;
    nij = ~ij;
    w2_x(ij) = w2_x(ij) - gamma*beta*w2_x(ij)./norm_w2(ij);
    w2_x(nij)  = 0;
    w2_y(ij) = w2_y(ij) - gamma*beta*w2_y(ij)./norm_w2(ij);
```

```
    w2_y(nij)  = 0;

    %update w3
    w3 = max(b3 + u, eta);

    %update b
    b1   = b1 + Ku - w1;
    b2_x = b2_x + Du_x - w2_x;
    b2_y = b2_y + Du_y - w2_y;
    b3   = b3 + u - w3;

    Kw3 = K(w3);
    den = Kw3 + bg;
    rapp(nonzeroindex) = f(nonzeroindex)./den(nonzeroindex);
    KL = sum( f(nonzeroindex).* log(rapp(nonzeroindex)) + ...
        den(nonzeroindex) - f(nonzeroindex) );
    KL = KL + sum(den(zeroindex));
    Dw3_x  = grad1(w3);%[w3(:,2:n)-w3(:,1:n-1), w3(:,1)-w3(:,n)];
    Dw3_y  = grad2(w3);%[w3(2:n,:)-w3(1:n-1,:); w3(1,:)-w3(n,:)];

    gu_norm = sqrt(Dw3_x.^2+Dw3_y.^2);
    fobj(itr+1)   = KL + beta*sum(sum(gu_norm));
    TimeCost(itr+1) = cputime - t0;

    for i=1:nobj
        err{i}(itr + 1) = norm(obj{i}(:)-w3(:))/normobj(i);
    end

    if verbose
        fprintf('\n%4d): f(x)=%g ', itr, ...
            fobj(itr+1) );
        for i=1:nobj
            fprintf(' err{%g} %g', i, err{i}(itr + 1));
        end
    end
    w = [u(:);w3(:);w2_x(:);w2_y(:);w1(:);b3(:);b2_x(:);...
        b2_y(:);b1(:)];
    wold = [uold(:);w3old(:);w2_xold(:);w2_yold(:);w1old(:);...
        b3old(:);b2_xold(:);b2_yold(:);b1old(:)];
    kkterr(itr+1) = norm(w(:)-wold(:))/norm(w(:));
    %kkterr(itr+1)=norm(w3(:)-w3old(:))/norm(w3(:));
    if kkterr(itr+1)< tol
        break
    end
end
% end of the main loop
fobj(itr +2:end) = [];
for i = 1:nobj
    err{i}(itr + 2:end) = [];
end
TimeCost(itr+2:end) = [];
kkterr(itr+2:end)=[];
if nargout > 0
    varargout{1} = w2_x;
    varargout{2} = w2_y;
    varargout{3} = b2_x;
    varargout{4} = b2_y;
    varargout{5} =kkterr;
end
```

B.8 prisma.m

```
function prisma(id,i,arrbeta)
%prisma(id,i,arrbeta)
% restoration of the image i from project PRISMA
% with the given array of beta arrbeta
% using both AEM and PIDSplit+
%
% id accepts 'aem','50','5','1','05'.

i = int2str(i);

image = ['PRISMA/Image' i '.tif'];
gn = 4095-double(imread(image));
gn = gn(1:266,138:403);

obj = 4095-double(imread('PRISMA/Avg.tif'));
obj = obj(1:266,138:403);

n = size(gn,1);

grad1  = @(y)[y(:,2:n)-y(:,1:n-1), y(:,1)-y(:,n)];
grad2  = @(y)[y(2:n,:)-y(1:n-1,:); y(1,:)-y(n,:)];
div    = @(x1,x2)([-x1(:,1)+x1(:,n),-x1(:,2:n)+x1(:,1:n-1)] +...
    [-x2(1,:)+x2(n,:);-x2(2:n,:)+x2(1:n-1,:)]);
H = @(x)x;
HT = @(x)x;

bg = 0;
delta = 0;
NIT = 3000;
tol = 1e-6;

zeroindex = gn <= 0;
nonzeroindex = ~zeroindex;
eta = min(gn(nonzeroindex))*ones(size(gn));
eta(zeroindex) = 0;

verbose =false;
Nalpha=10;
epsilon = 1e-4;

benchsol={obj};
TF = ones(size(obj));

for beta = arrbeta;

    if strcmp(id,'aem')

        fprintf('AEM: Working on Image%s with beta %g\n', i,beta);
        [u,y1,y2,y3,TimeCost,InnVec,Primal,phi_xy,alpha_vec,err,...
            KKT,KL]= AEM(gn, H, HT,bg, beta, delta, eta, grad1,...
            grad2, div, NIT, tol, verbose, Nalpha, epsilon,...
            benchsol);
        fprintf('Number of iteration performed:%g\n',...
            length(err{1})-1);
        save(['PRISMA/' 'data_Image' i '_'...
            num2str(beta) '.mat' ]);

    elseif strcmp(id,'50')
```

```
        gamma = 50/beta;
        fprintf('PID%s: Working on Image%s with beta %g\n',...
            id,i,beta);
        [u_50,w3_50,TimeCost_50,fobj_50,err_50,w2_y_50,...
            b2_x_50,b2_y_50,kkterr_50]  = PIDSplit_plus(gn, TF,...
            H,HT,bg, beta, gamma, eta, grad1, grad2, div, NIT,...
            tol,verbose, benchsol);
        fprintf('Number of iteration performed:%g\n',...
            length(err_50{1})-1);
        save(['PRISMA/' 'data_Image' i '_PID_' id '_' ...
            num2str(beta) '.mat']);

    elseif strcmp(id,'5')
        gamma = 5/beta;
        fprintf('PID%s: Working on Image%s with beta %g\n',...
            id,i,beta);
        [u_5,w3_5,TimeCost_5,fobj_5,err_5,w2_y_5,b2_x_5,...
            b2_y_5,kkterr_5]  = PIDSplit_plus(gn, TF, H, HT,...
            bg, beta, gamma, eta, grad1, grad2, div, NIT, ...
            tol, verbose, benchsol);
        fprintf('Number of iteration performed:%g\n',...
            length(err_5{1})-1);
        save(['PRISMA/' 'data_Image' i '_PID_' id '_' ...
            num2str(beta) '.mat']);

    elseif strcmp(id,'1')
        gamma = 1/beta;
        fprintf('PID%s: Working on Image%s with beta %g\n',...
            id,i,beta);
        [u_1,w3_1,TimeCost_1,fobj_1,err_1,w2_y_1,b2_x_1,...
            b2_y_1,kkterr_1]  = PIDSplit_plus(gn, TF, H, HT,...
            bg, beta, gamma, eta, grad1, grad2, div, NIT, ...
            tol, verbose, benchsol);
        fprintf('Number of iteration performed:%g\n',...
            length(err_1{1})-1);
        save(['PRISMA/' 'data_Image' i '_PID_' id '_' ...
            num2str(beta) '.mat']);

    elseif strcmp(id,'05')
        gamma = 0.5/beta;
        fprintf('PID%s: Working on Image%s with beta %g\n',...
            id,i,beta);
        [u_05,w3_05,TimeCost_05,fobj_05,err_05,w2_y_05,...
            b2_x_05,b2_y_05,kkterr_05]  = PIDSplit_plus(gn,...
            TF, H, HT, bg, beta, gamma, eta, grad1, grad2, ...
            div, NIT, tol, verbose, benchsol);
        fprintf('Number of iteration performed:%g\n',...
            length(err_05{1})-1);
        save(['PRISMA/' 'data_Image' i '_PID_' id '_' ...
            num2str(beta) '.mat']);

    end
end
%exit;
```

Bibliography

[1] M. Bertero, P. Boccacci, G. Talenti, R. Zanella, and L. Zanni. A discrepancy principle for Poisson data. *Inverse Problems*, 26, 2010.

[2] H.C. Andrews and B. R. Hunt. *Digital image restoration*. Prentice Hall, 1977.

[3] S. Geman and D. Geman. Stochastic relaxation, Gibbs distribution and the Bayesian restoration of images. *IEEE Trans. Pattern Analysis and Machine Intelligence*, 6:721–741, 1984.

[4] V.A. Mozorov. Choice of a parameter for the solution of functional equations by a regularization method. *Sov. Math. Doklady*, 8:1000–1003, 1967.

[5] L. Rudin, S. Osher, and E. Fatemi. Nonlinear total variation based noise removal algorithms. *Physica D*, 60:259–268, 1992.

[6] R. T. Rockafellar. *Convex Analysis*. Princeton University Press, 1970.

[7] S. Osher, M. Burger, D. Goldfarb, J. Xu, and W. Yin. An iterative regularization method for total variation-based image restoration. *Multiscale Model. Simul.*, 2005.

[8] T. Goldstein and S. Osher. The Split Bregman Method for L1 regularized problems. *SIAM J. of Imaging Science*, 2009.

[9] S. Setzer, G. Steidl, and T. Teuber. Deblurring Poissonian images by split Bregman techniques. *Journal of V. Commun. and Image R.*, 21:193–199, 2010.

[10] S. Setzer. Operator Splittings, Bregman Methods and Frame Shrinkage in Image Processing. *Int. J. Comput. Vis.*, 2010.

[11] M.R. Hestenes. Multiplier and gradient methods. *Journal of Optimization Theory and Applications*, 4:303–320, 1969.

[12] M.J.D. Powell. *Optimization*, chapter A method for nonlinear constraints in minimization problems, pages 283–298. Academic Press, New York, NY, 1969.

[13] D. Bertsekas and J.N. Tsitsiklis. *Parallel and Distributed Computation: Numerical Methods*. Prentice-Hall, 1989.

[14] D. Gabay and B. Mercier. A dual algorithm for the solution of nonlinear variational problems via finite element approximation. *Computers & Mathematics with Applications*, 2:17–40, 1976.

[15] R. Glowinski and A. Marocco. Sur l'approximation par éléments finis d'ordre un, et la résolution par pénalisation-dualité, d'une classe de problèmes de Dirichlet nonlinéaires. *RAIRO Anal. Num.*, 2:41–76, 1975.

[16] D. Gabay. *Augmented Lagrangian methods: applications for the numerical solutions of boundary-value problems*, volume 15 of *Studies in mathematics and its applications*, chapter Applications of the method of multipliers to variational inequalities, pages 299–331. North-Holland, 1983.

[17] J. Eckstein. Splitting methods for monotone operators with applications to parallel optimization, 1989.

[18] S. Bonettini and V. Ruggiero. An alternating extragradient method for total variation based image restoration from Poisson data. To appear in Inverse Problems, 2011.

[19] P. Marcotte. Application of Khobotov's algorithm to variational inequalities and network equilibrium problems. *INFOR.*, 29:258–270, 1991.

[20] R. Zanella, P. Boccacci, L. Zanni, and M. Bertero. Efficient gradient projection methods for edge-preserving removal of Poisson noise. *Inverse Problems*, 25, 2009.

[21] P.J. Davis. Circulant Matrices. *John Wiley & Sons, Inc.*, 1979.

[22] J. Nocedal and S. J. Wright. *Numerical Optimization*. Springer, first edition, 1999.