Image processing with variational approaches and Partial Differential Equations Practice 1: Denoising and restauration

Nicolas Papadakis and Jean-François Aujol

IMB, Université Bordeaux 1
351 Cours de la libération, 33405 Talence Cedex, FRANCE

Email : nicolas.papadakis@math.u-bordeaux.fr

1. Introduction

All along this practice, we will consider gray scale images and we will use MATLAB®.

1.1 General advices when using MATLAB

MATLAB has been first developed for solving matrix problems and its intern functions are optimized for such purpose. Hence, for optimizing computational runtime, it is recommended to write your algorithms in a vectorial/matrix form and to avoid the use of *for* loops when possible. Even if not necessary, it is recommended to initialize variables (with functions such as *zeros*, *ones*). To comment a line, use the character %. All created figures can be closed with the command *close all*. All created variables can be cleared with the command *clear all*. Do not hesitate to look at the documentation to correctly use the predefined functions (command: *help*).

1.2 MATLAB and images

In MATLAB, a gray scale image I of size (m, n) is a matrix I of size (m, n), and the value $I(i, j)$ corresponds to the gray value at pixel location (i, j), $i = 1 \cdots m$, $j = 1 \cdots n$.

MATLAB can read images written in any standard format with the command *imread*. In the same way, results can be written with the command *imwrite*. When reading an image with *imread*, the loaded values are integers and the gray values are in the range $[0; 255]$, from black to white.

Remark: You can copy/paste from this pdf the following scripts and commands.

1.3 Basic examples

Example 1: Reading and displaying an image

```
%load the image camerama.tif that is included in MATLAB's own dataset:

Im_data=imread('cameraman.tif');

%VERY IMPORTANT: do not forget to convert integer values
%in real ones for future manipulations:

Im_data=double(Im_data);

%Display:

imagesc(Im_data);
colormap gray;

%To open a second figure:

figure;

%Other command for image display:

imshow(Im_data/255.);

%If the image has real values, imshow require these values to be
%in the range [0;1] for a correct display. A normalization is thus
%required if working in the range [0;255]
```

The previous script should display:

Display of image Cameraman with *imagesc*

Display of image Cameraman with *imshow*

Example 2: Avoid “for loops”

```
N=2000;
A=ones(N,N);   %Initialize A with values 1
B=randn(N,N);  %Initialize B with random values in [0;1]

%Compute product between matrices elements.
tic
for i=1:N,
     for j=1:N,
          A(i,j)=A(i,j)*B(i,j);
     end
end
toc    %Display runtime since tic command

tic
A=A.*B;  % .* for an element wise operation between matrices
%(different from A*B that realizes a matrix multiplication)
toc
```

The previous script should give you as output something (depending on the computer) like:

```
Result after execution:
Elapsed time is 0.129287 seconds.
Elapsed time is 0.005119 seconds.
```

Exemple 3: Adding gaussian noise to an image with a function

Create the file add_gaussian_noise.m containing:

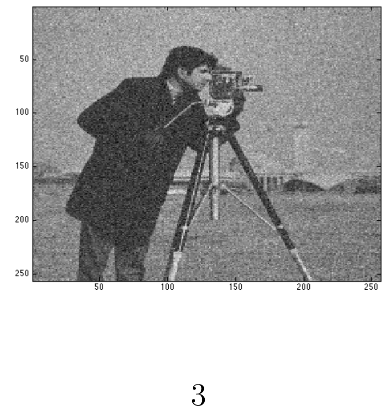
```
function out = add_gaussian_noise(I,s)
%Add Gaussian noise of standard deviation s to an image I

[m,n]=size(I);
%creation and addition of gaussian noise
out=I+s*randn(m,n);
```

The script:

```
Im_noised=add_gaussian_noise(Im_data,30);
imagesc(Im_noised)
colormap gray;
```

should give you:

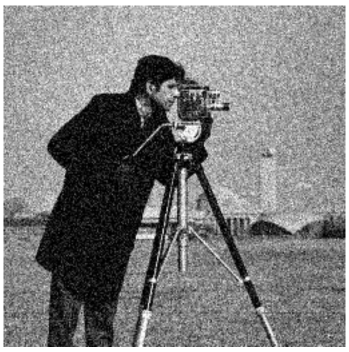


2 Denoising with PDEs

Let Ω be a bounded open set of $\mathbb{R}^2$. We will typically consider Ω as a rectangle representing the image domain. An image is a function defined on Ω such as $u : \Omega \to \mathbb{R}$ ($u : \Omega \to \mathbb{R}^3$ for color images), i.e. for pixels $x \in \Omega$.

We now consider that we have an image f that corresponds to an unknown ground truth image u^* perturbed with an additional Gaussian noise ω. From the previous examples we can take:

u^*


$f = u^* + \omega$

and the objective is to denoise the available image f and recover an image as close as possible to u^*.

2.1 Heat equation

The first PDE that have been used in image processing is the Heat equation that realizes a spatial diffusion of the gray values of a given image. It is a parabolic equation that reads:

$$\begin{cases} \frac{\partial u(t,x)}{\partial t} & = \Delta u(t,x) \quad \text{for } t \geq 0 \text{ and } x \in \Omega \\ u(0,x) & = f(x) \quad \text{for } x \in \Omega \\ \frac{\partial u(t,x)}{\partial N} & = 0 \quad \text{for } t > 0 \text{ and } x \in \partial\Omega, \end{cases} \tag{2.1}$$

where Δ is the Laplacian operator, f is the initial temporal condition and $\frac{\partial u}{\partial N} = 0$ are Neumann boundary conditions. This model diffuses the initial condition f (that can be seen as an initial temperature) along time.

The introduction of this equation comes from the following remark. If f, the initial condition, is smooth enough, then the explicit solution of (2.1) is given by:

$$u(t,x) = \int_{\mathbb{R}^2} G_{\sqrt{2t}}(x-y) f(y)\, dy = \left(G_{\sqrt{2t}} * u_0\right)(x) \tag{2.2}$$

where G_σ is a Gaussian kernel of dimension 2:

$$G_\sigma(x) = \frac{1}{2\pi\sigma^2} \exp\left(-\frac{|x|^2}{2\sigma^2}\right) \tag{2.3}$$

The convolution of a data f with a positive kernel is a basic operation in image processing that corresponds to a low-pass filter that will “kill” high frequencies of f and thus remove its noise.

2.2 From continuous to discrete

In image processing, we only have access to discrete values $u(i,j)$ that are located at pixels $x = (i,j)$, $i = 1 \cdots m$, $j = 1 \cdots n$, on the discrete grid describing the image domain Ω.

Discrete spatial operators A discrete image is thus a matrix of dimension $m \times n$. We denote as X the euclidean space $\mathbb{R}^{m\times n}$, and $Y = X \times X$. The space X is equipped with the scalar product:

$$\langle u, v\rangle_X = \sum_{1\le i,j\le N} u_{i,j}v_{i,j} \tag{2.4}$$

and the norm:

$$\|u\|_X = \sqrt{\langle u, u\rangle_X}. \tag{2.5}$$

If $u \in X$, the gradient $\nabla u = [\partial_x u; \partial_y u]^T$ is a vector of Y given by :

$$(\nabla u)_{i,j} = ((\nabla u)^1_{i,j}, (\nabla u)^2_{i,j}) \tag{2.6}$$

where the horizontal gradient can be discretized as

$$(\nabla u)^1_{i,j} = \begin{cases} u_{i+1,j} - u_{i,j} & \text{si } i < M \\ 0 & \text{si } i = M \end{cases} \tag{2.7}$$

and the vertical gradient as:

$$(\nabla u)^2_{i,j} = \begin{cases} u_{i,j+1} - u_{i,j} & \text{si } j < N \\ 0 & \text{si } j = N \end{cases} \tag{2.8}$$

These schemes correspond to the general forward discretization $\partial_x u(i,j) = \frac{u(i+h,j)-u(i,j)}{h}$ with a spatial step of $h = 1$.

We also introduce a discrete version of the divergence operator $\operatorname{div}(p) = \partial_x p_1 + \partial_y p_2$ for a vector $p = [p^1, p^2] \in Y$. It is defined in analogy with the continuous case as:

$$\operatorname{div} = -\nabla^* \tag{2.9}$$

where ∇^* is the adjoint operator of ∇ : i.e., for all $p \in Y$ and $u \in X$, $\langle -\operatorname{div} p, u\rangle_X = \langle p, \nabla u\rangle_Y$. One can then show that:

$$(\operatorname{div}(p))_{i,j} = \begin{cases} p^1_{i,j} - p^1_{i-1,j} & \text{si } 1 < i < M \\ p^1_{i,j} & \text{si i=1} \\ -p^1_{i-1,j} & \text{si i=M} \end{cases} + \begin{cases} p^2_{i,j} - p^2_{i,j-1} & \text{si } 1 < j < N \\ p^2_{i,j} & \text{si j=1} \\ -p^2_{i,j-1} & \text{si j=N} \end{cases} \tag{2.10}$$

Hence, the discretization of the Laplacian operator $\Delta u = \partial_{xx}u + \partial_{yy}u$ for $u \in X$ will be given as:

$$\Delta u = \operatorname{div} \nabla u. \tag{2.11}$$

One can check that combining (2.7), (2.8) and (2.10), we recover the standard Laplacian discretization for $1 < i < m$ and $1 < j < n$:

$$\Delta u^n_{i,j} = u^n_{i+1,j} + u^n_{i-1,j} + u^n_{i,j+1} + u^n_{i,j-1} - 4u^n_{i,j} \tag{2.12}$$

The functions gradx.m , grady.m and div.m computing these operators are available here: gradx.m grady.m div.m.

Boundary conditions As we consider a bounded open set Ω, adequate boundary conditions are required. Neumann conditions are the natural conditions that arise in image processing.

Nevertheless from the above discretizations of the gradient and divergence operators that ensures that $\langle p, \nabla u\rangle_Y = \langle -\operatorname{div} p, u\rangle_X$ for all $p \in Y$ and $u \in X$, we have the Neumann conditions for free from the Theorem of Stokes. ***As a consequence, we won't have to deal with the conditions*** $\frac{\partial u(t,x)}{\partial N} = 0$ ***in the implementation.***

Temporal scheme Let us now detail how discretizating in time the PDE (2.1). We denote as δt the numerical time step and $u^k_{i,j}$ represents the value of the image at time $t = k\delta_t$. We consider an explicit Euler scheme of order 1 in time, which leads to the following approximation of the temporal partial derivative:

$$\frac{\partial u^{k+1}_{i,j}}{\partial t} = \frac{u^{k+1}_{i,j} - u^k_{i,j}}{\delta t}. \tag{2.13}$$

Gathering all previous information, the discretization of problem (2.1) finally reads:

$$\begin{cases} u^{k+1}_{i,j} &= u^k + \delta_t(\Delta u^k)_{ij} \\ u^0_{ij} &= f_{ij}, \end{cases} \tag{2.14}$$

where, from relation (2.11), the discretization of the Laplacian is obtained with schemes (2.7), (2.8) and (2.10).

2.3 Time to work:

Heat equation: The objective is now to solve numerically the problem (2.14).

1 Write a function *heat_ equation* that

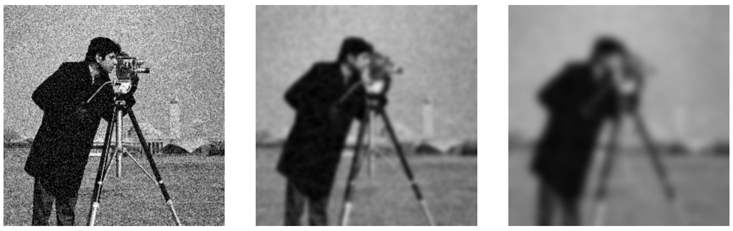
- takes as argument the noisy image f, a time step δ_t and a number of iterations K.
- solves problem (2.14) for iterations $k = 1 \cdots K$ (the values u^{k+1} should erase the temporary values u^k to save memory space).
- returns u^K.

2 Write a script that test this function on a noisy image f for different evolution times K. The time step can be set to $\delta_t = 1/8$ to ensure the stability of the scheme.

One can display u^k along iterations to visualize the diffusion process (i.e. the evolution of the solution):

```
imagesc(u);
colormap gray;
drawnow;  %to have a real time display
```

Depending on the evolution time K, we observe images u^K that are more or less smooth:



$K = 0$ ($u^0 = f$) $K = 20$ $K = 100$

Convolution with a Gaussan kernel: We now check that (2.2) is indeed a solution of (2.1).

3 Define a matrix G of size $(2P-1, 2P-1)$ representing a Gaussian kernel of standard deviation $\sigma^2 = 2K\delta_t$. One can take for instance $P = \lfloor K\delta_t + 1 \rfloor$. Two options are possible to define this kernel:

  - Hard one: using (2.3) (in which the index $x = (0,0)$ will correspond to the position $G(P,P)$).
  - Esay one: using *fspecial* with options *'gaussian'*, size $(2P-1, 2P-1)$ and standard deviation $\sqrt{2K\delta_t}$.

4 Realize a convolution of f with G with the function *imfilter* and the option *'replicate'* to mimic Neumann conditions.

5 Compare the result with the one obtained with the Heat equation.

Application to contour detection: PDEs can be used as pre-processing tasks for other applications, such as contour detection that consists in finding the significant contours present in an image. A main ingredient of contour detectors is to look for pixels (i,j) for which the gradient norm $||\nabla u(i,j)||$ is large. The detection of significant contours of an image it then more robust when the image is smooth enough.

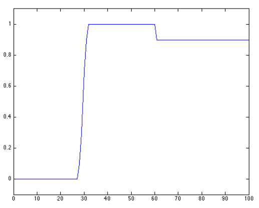
We illustrate this fact with the Marr-Hildreth contour detector that estimates that a pixel belongs to a significant contour if 2 conditions are met:

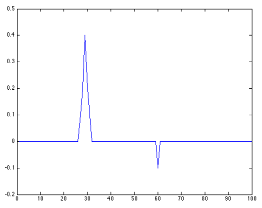
(1) $||\nabla u(i,j)|| > \eta > 0$, where η is a threshold that selects pixels of high gradient. A small parameters involves the selection of too many pixel (and bold contours), whereas a large one will only select a few one. Considering only this criteria does not give accurate detection as the threshold is hard to tune.

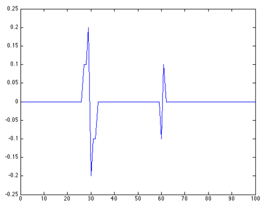
(2) $\Delta u(i,j)$ changes its sign at location (i,j). This means that the pixel (i,j) is a local extrema of $\nabla u(i,j)$. This criteria will detect thin contours but will be sensible to noise.

We now given a $1D$ illustration of this algorithm. In the next Figure, we consider a line of the image (a) denoted as u and shown in (b). The gradient ∇u and the Laplacian Δu are illustrated respectively in (c) and (d).

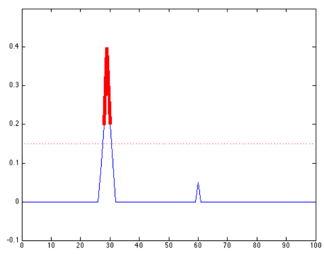
(a) *Image*

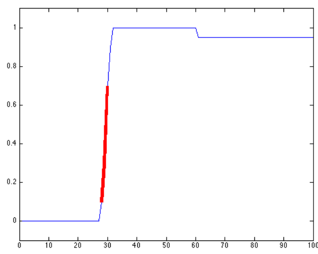

(b) u


(c) ∇u


(d) Δu

Let us now observe the effect of condition (1) on ∇u. When taking (a) a threshold of $\eta = 0.15$, we see that (b) the condition $||\nabla u(i,j)|| > \eta$ find the main contour of the image but it involves a bold contour in the final detection (c).


(a) $||\nabla u|| > \eta = 0.15$


(b) Contours in u

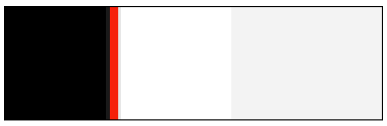
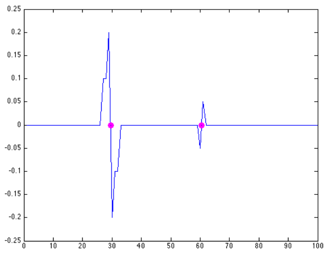
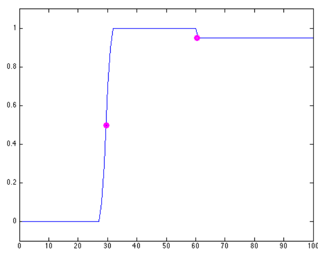

(b) Contour in the image

Illustration of condition (1)

We now detail the effect of condition (2) on Δu. When looking (a) at pixels (i,j) where Δu changes its sign, we see that (b) the local extrema of ∇u are found but it involves the detection of non significant contours in the final detection (c).


(a) Δu changes its sign


(b) Contours in u

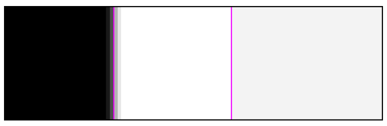
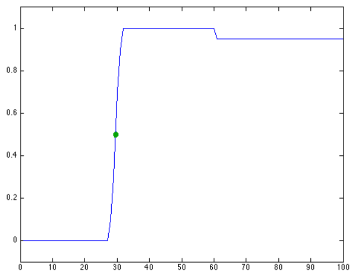

(b) Contour in the image

Illustration of condition (2)

Combining the two criteria, we see below that the algorithm only selects the pixel with a large enough gradient norm that is also a local extrema. This gives an accurate contour detection (b).


(a) Contours in u

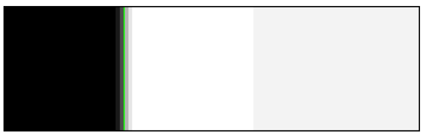

(b) Contour in the image

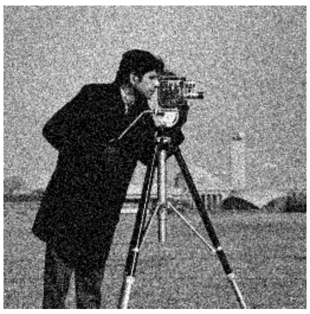
Illustration of conditions (1) + (2)

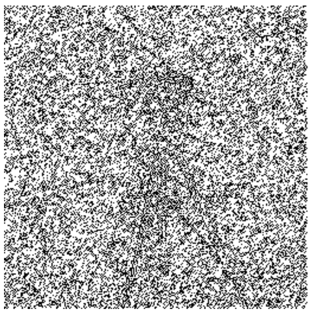
6 Write a function *Marr_Hildreth* that takes as input an image u and a threshold $\eta > 0$ and returns the contour image that contains the value 1 for pixels checking conditions (1) and (2) and 0 otherwise. The function detecting change of sign can be found here: change_sign.m.

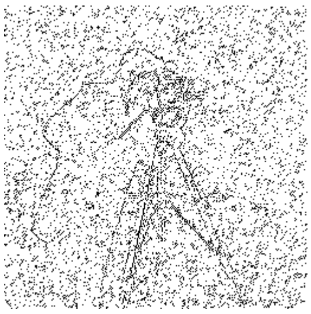
7 Apply the function to the noisy image f and to the denoised one u^K.

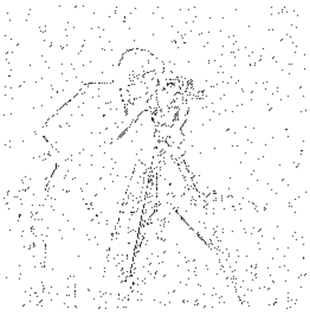
We illustrate the influence of the threshold η and the smoothness of the image on the contour detection in the following Figure. When the image is noisy (first line), the detector finds a lot of pixels with a large gradient that are local extrema. When smoothing the image (lines 2 and 3), the noise is removed. However when the smoothing of the image is too important (line 3), the contours are also smoothed in the detection.

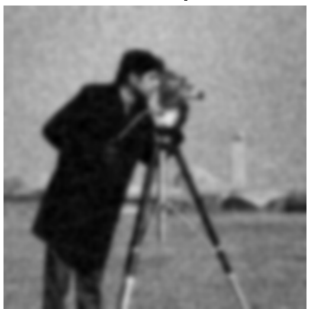
Notice that this algorithm is basic and more evolved detectors exist, see for instance the MATLAB's function *edge* and its options.

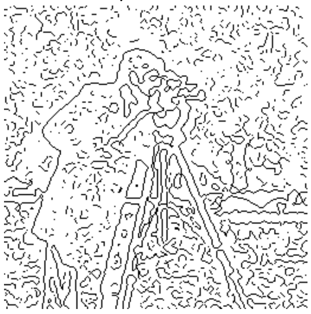

$u^0 = f$

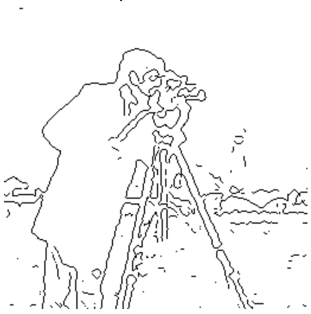

$\eta = 50$

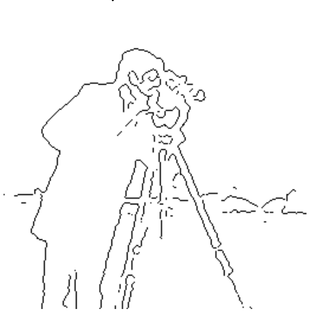

$\eta = 80$

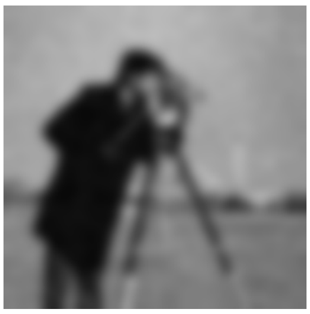

$\eta = 120$

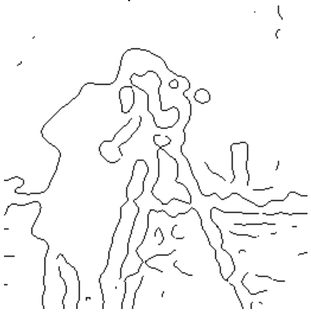

u^{20}

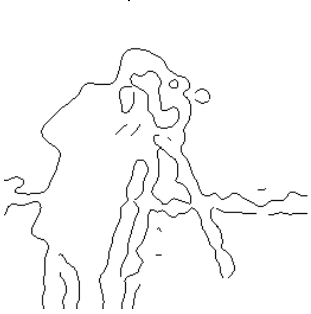

$\eta = 2$

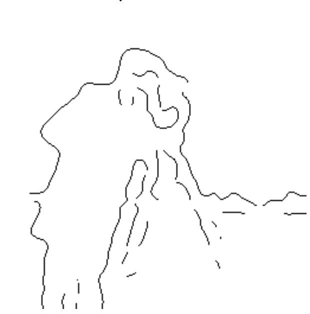

$\eta = 5$


$\eta = 10$


u^{100}


$\eta = 1$


$\eta = 2$


$\eta = 5$

The Heat equation which is equivalent to the convolution with a Gaussian Kernel, is isotropic, meaning that no particular direction are considered during the diffusion. This is an issue for denoising and contour detection, since one would like to preserve the significant image discontinuities present in the original image.

2.4 Perona-Malik

In order to enhance the results obtained with the Heat equation, Perona and Malik proposed to modify the equation by integrating information related to the presence of boundaries:

$$\begin{cases} \frac{\partial u(t,x)}{\partial t} &= \operatorname{div}\Big(g\big(||\nabla u(t,x)||\big)\nabla u(t,x)\Big) & \text{for } t \geq 0 \text{ and } x \in \Omega \\ u(0,x) &= f(x) & \text{for } x \in \Omega \\ \frac{\partial u(t,x)}{\partial N} &= 0 & \text{for } t > 0 \text{ and } x \in \partial\Omega, \end{cases} \tag{2.15}$$

where c is a decreasing function from $\mathbb{R}_+$ to $\mathbb{R}_+$.

Remark: If taking a constant function $g = 1$, we recover the ***isotropic*** Heat equation.

The function g is classically taken such that $g(0) = 1$ and $\lim_{\xi \to +\infty} g(\xi) = 0$, with for instance $g(\xi) = exp(-\xi^2/\alpha^2)$. Hence, in an uniform (i.e constant) area where the gradient of a pixel (i,j) is small ($||\nabla u(i,j)|| \approx 0$), the Perona-Malik model acts like an isotropic diffusion with $g = 1$ at this pixel. On the other hand, when the gradient is large ($||\nabla u(i,j)|| >> 0$), the diffusion is stopped with $g = 0$ at this location of high gradient, which allows a better preservation of contours. The Perona-Malik PDE is thus an ***anisotropic diffusion*** since specific directions are discouraged. In the following we will consider

$$g(\xi) = \frac{1}{\sqrt{(\xi/\alpha)^2 + 1}}. \tag{2.16}$$

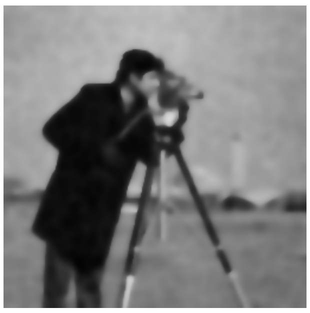
8 Write a function *Perona_Malik* that

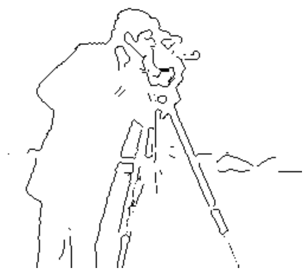
- takes as argument the noisy image f, a time step δ_t, a number of iterations K and a parameter α
- solves problem (2.15), with the temporal derivative (2.13) and the function g defined in (2.16), for iterations $k = 1 \cdots K$
- returns u^K.

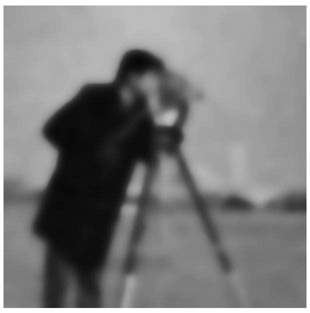
9 Write a script that test this function on a noisy image f for different evolution times K and parameters α.

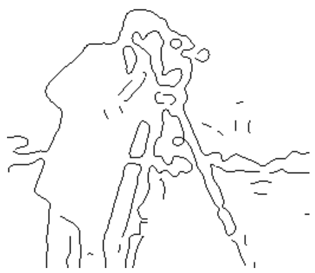
10 Apply the Marr Hildreth contour detector on the obtained results.

As shown in the next figure, the Perona-Malik PDE better preserves the discontinuities in the final images u^K, which involves more accurate contour detections.


u^{50}, $\alpha = 15$


$\eta = 10$


u^{100}, $\alpha = 15$


$\eta = 2$

Enhacement with a convolution of the gradient with a Gaussian As previously observed, whene the original image f is very noisy, its gradients present strong oscillations that perturb the diffusion. The noise may indeed be considered as a significant contour. E A simple approach to circumvent this issue is to smooth the gradient that is send to the function c. The improved model then reads:

$$\begin{cases} \frac{\partial u(t,x)}{\partial t} & = \operatorname{div}\Big(g(G_\sigma * ||\nabla u(t,x)||)\nabla u(t,x)\Big) & \text{for } t \geq 0 \text{ and } x \in \Omega \\ u(0,x) & = f(x) & \text{for } x \in \Omega \\ \frac{\partial u(t,x)}{\partial N} & = 0 & \text{for } t > 0 \text{ and } x \in \partial\Omega. \end{cases} \tag{2.17}$$

Optional: Implementation of this model and comparison with the previous one.

With this convolution, the main drawbacks of the previously studied PDEs are still present. Namely, tuning the number of iterations is a hard task. Doing a few iterations does not denoise the image, whereas iterating a lot over smooth the image. The main problem comes from the fact that the initial condition f is "lost" during the diffusion. We now see how re-injecting such information into the process.

3 Denoising with variational approaches

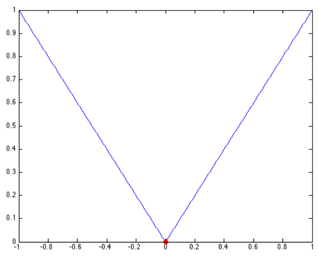
Before coming to variational models, we first recall some properties of convex functionals.

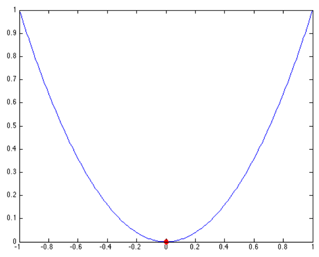
3.1 Minimization of convex and differentiable functionals

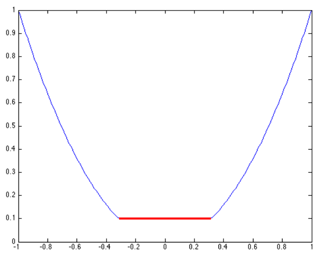
In this practice, we consider $u \in X$, where X is the euclidean space $\mathbb{R}^{m\times n}$. A function $J : X \to \mathbb{R}$ is said convex iff:

$$\forall (u,v) \in X \times X, \quad \textit{and} \quad \forall t \in [0;1], \quad J(tu + (1-t)v) \leq tJ(u) + (1-t)J(v).$$

The function is strictly convex if the above inequality is strict $\forall u \neq v$ and $t \in]0;1[$. Classical $1D$ examples (i.e. $m = n = 1$) of convex and strictly convex functions are given below.


$J(u) = |u|$ is convex


$J(u) = ||u||^2$ is strictly convex

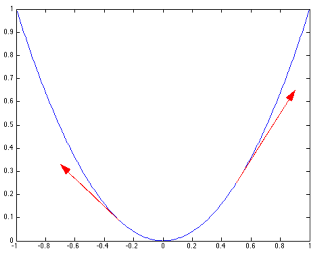

$J(u) = \max(||u||^2, 0.1)$ is convex

Assuming that J is proper, lower semi-continuous and coercive (see Lecture for definitions and details), then a convex function admits at least one global minimizer. If the function is not strictly convex, the minimizers are not necessarily unique (the values of the functions at the minimizers are in red in the previous Figures).

From now, we will only consider differentiable functions J so that $\nabla J(u) \in X$ exists for all $u \in X$ and it is the only vector of X that checks:

$$J(v) \geq J(u) + \langle \nabla J(u), v - u \rangle, \quad \forall v \in X.$$

Notice that ∇ here represents the derivative of the function J with respect to u: $\nabla J(u) = \partial_u J(u)$, and not the spatial gradient as before. The vector $\nabla J(u)$ then defines the tangent of J at point u, as illustrated below:



$J(u) = ||u||^2$, the red vectors represent (with a rescaling) $\nabla J(u) = 2u$ at points $u = 0.5$ and $u = -0.3$.

In this differentiable and convex context, we can observe that u is a minimizer of J iff $\nabla J(u) = 0$. The gradient ∇J thus allows to characterize minimizers of J, but it also permits to get closer to one minimizer from a current point u, by going in the opposite direction from $\nabla J(u)$ and thus decrease the function J. Hence, with an adequate time step $\tau > 0$ and with an additional condition on ∇J (discussed in the remark page 13), the well-known gradient descent algorithm:

$$u^{k+1} = u^k - \tau \nabla J(u^k),$$

converges to a global minimizer of J for any $u^0 \in X$.

3.2 Previous PDEs as gradient descent of convex functionals

From now on, we will consider the discrete framework so that the following sums on Ω corresponds to a discretization of the continuous integrals on Ω in the Lecture. Let us now consider the convex function $J_H(u) = \frac{1}{2}\sum_{x\in\Omega} ||\nabla u(x)||^2 = \frac{1}{2}||\nabla u||_Y^2$. From calculus of variations, one obtains $\nabla J_H(u) = -\text{div}\,(\nabla u) = -\Delta u \in X$.

We can then apply the gradient descent algorithm in order to compute a minimizer of this convex function. Initializing $u^0 = f$, it reads:

$$u^{k+1} = u^k + \tau \Delta u^k,$$

which exactly corresponds to the previous discretization of the Heat equation (2.14).

In the same vein, we can show that the Perona-Malik PDE[1] defined in (2.15) corresponds to a gradient descent algorithm applied to the convex functional $J_{PM}(u) = \sum_{x\in\Omega} \sqrt{||\nabla u(x)||^2 + 1}$, since:

$$\nabla J_{PM}(u) = -\text{div}\left(\frac{\nabla u}{\sqrt{||\nabla u||^2 + 1}}\right).$$

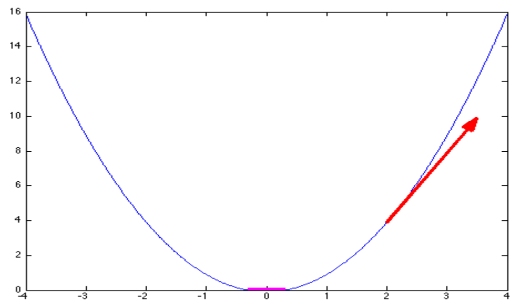
Homework: Check the convexity and the computation of the gradients of the above functions J_H and J_{PM}.

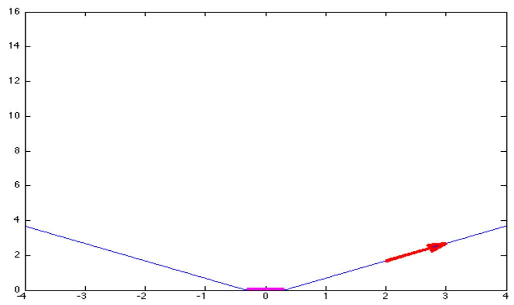
[1]where $c(t)$ of (2.16) is parameterized with $\alpha = 1$.

Interpretations From the gradient descent point of view, we see that with an infinite time t, the previous PDEs (2.1) and (2.15) will respectively converge to a global minimizer of the functions J_H and J_{PM}.

We denote as C the set of uniform images (i.e. constant) $u \in X$ (i.e. so that $u \in C$ iff $\nabla u(x) = 0$, $\forall x \in \Omega$). It is clear that $J_H(u) = 0$ if $u \in C$ and $J_H(u) > 0$ if $u \notin C$. The same observation can be made for J_{PM}. Hence C is the set of global minimizers of functions J_H and J_{PM}. This confirms that the PDEs will converge to constant images[2].

The difference between both PDEs concerns the paths u^k between $u^0 = f$ and $u^\infty = constant$. The isotropic Heat equation corresponds to a quadratic function J_H, whereas the anisotropic Perona-Malik model relies on a differentiable approximation of the piecewise-linear and non differentiable Total Variation regularization $\sum_{x\in\Omega} \sqrt{||\nabla u(x)||^2}$. These functions are displayed below:


$J_H(u)$


$J_{PM}(u)$

From now on, in the Figures, the x-axis is a $1D$ representation of $X = \mathbb{R}^{m\times n}$. The set C of global minimizers is in purple: $J_H(u) = J_{PM}(u)$ for all $u \in C$. The vectors $\nabla J_H(u)$ and $\nabla J_{PM}(u)$ are displayed in red for a same u.

From the shape of these functions, we understand that for an initialization f far from C and the same time steps, the gradient descent on the quadratic J_H will involve gradients $\nabla J_H(u)$ with higher norms than $\nabla J_{PM}(u)$ and it will reach faster the neighborhood of C than the gradient descent on J_{PM}. This explains why the Heat equation gives very smooth images (close to C) with few iterations.

Remark on the convergence of gradient descent If ∇J is Lipschitz continuous with constant L (i.e. $||\nabla J(u) - \nabla J(v)|| \leq L||u - v||$, for all $u, v \in X$), then the gradient descent converges for all $\tau < 2/L$. For the Heat equation, $\nabla J_H = -\Delta$ is Lispchitz continuous. With the discretization of the Laplacian considered page 5, the constant L of Δ is 8, so that we can take $\tau < 1/8$. On the other hand, it is worth noting that ∇J_{PM} is not Lispchitz continuous so that the Perona-Malik PDE may diverge if not optimizing the time step at each iteration with line search methods.

Enhancing PDEs methods A previously underlined drawback of the presented PDEs is that the original data f is only used as an initialization and forgotten along the process. We now see how defining a convex function that will take into account this information so that its minimizers will have a stronger link with f.

[2]With the Neumann condition, the PDEs will converge to the constant image where the constant is the mean value of f. It corresponds to a homogeneous diffusion on the whole domain Ω of the initial temperature f.

3.3 Variational model with data fidelity term

As previously noticed, the functions J_H and J_{PM} have a smoothing effect. A simple idea to counter balance this regularization behavior is to consider an additional data fidelity term:

$$J_D(u) = \frac{1}{2}\sum_{x\in\Omega} ||u(x) - f(x)||^2 = \frac{1}{2}||u - f||_X^2$$

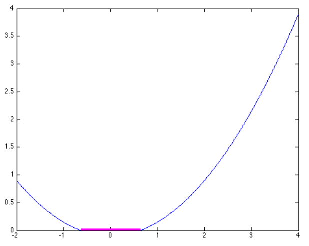
This function is strictly convex and its obvious minimizer is obtained for $u = f$. This model assume that the data is a degradation of the perfect unknown image u^* with a Gaussian noise ω, i.e. $f = u^* + \omega$. Notice that minimizing the $||.||_X^2$ norm corresponds, in the Bayesian framework, to the minimization of the likelihood of the image u with respect to the data noisy f.

We can now consider the following kind of models:

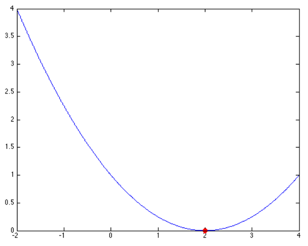
$$J_\lambda(u) = \lambda J_D(u) + J_R(u), \tag{3.18}$$

where $\lambda \geq 0$ is a parameter weighting the influence of the regularization with respect to the data and J_R is a regularization function that can be taken as $J_H(u) = \frac{1}{2}\sum_{x\in\Omega} ||\nabla u(x)||^2$ or $J_{PM}(u) = \sum_{x\in\Omega}\sqrt{||\nabla u(x)||^2 + 1}$.

It is worth noting that the function J_λ in (3.18) is strictly convex for $\lambda > 0$. We call u_λ a minimizer of (3.18) for a given λ. For $\lambda = 0$, we recover the previously studied models and u_λ is not unique (any $u \in C$ is a minimizer). For $\lambda \to 0^+$, the minimum is achieved for the constant image with the same mean as f. For $\lambda \to \infty$, the data term is prominent so we have $\lim_{\lambda\to\infty} u_\lambda = f$. The interesting values are thus in between. We next illustrate the influence of this parameter on the minimizer u_λ.

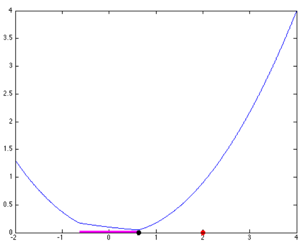


$J_R(u)$

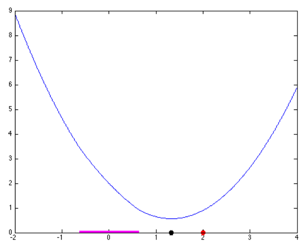


$J_D(u)$

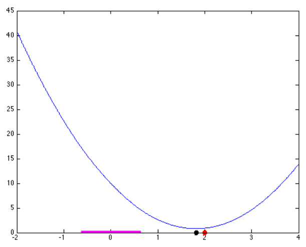
The set C of global minimizers of J_R is in purple. The minimizer of J_D $(= f)$ is in red.



$J_\lambda(u)$, $\lambda = 0.1$



$J_\lambda(u)$, $\lambda = 2$



$J_\lambda(u)$, $\lambda = 10$

Illustration of J_λ and its minimizer u_λ (black points) for different values of λ.

Hence, we clearly observe that the minimizer of J_λ will be a compromise between the data f and an uniform image in C. Tuning the parameter λ adequately is in practice more simple than setting a number of iteration in PDEs approaches. The parameter can for instance been set with respect to the expected noise level of the image.

Variational models in image processing In image processing applications (denoising, segmentation, optical flow estimation...), it is classical to design algorithms based on the minimization of a function. The important point is therefore to ensure that the minimizers of the proposed function have good properties with respect to the tackled application. Here these properties are the following: the denoised image u should be smooth and close to f.

3.3.1 Minimization of J_λ

We now consider the gradient descent algorithm to solve problem (3.18):

$$u^{k+1} = u^k - \tau(\lambda \nabla J_D(u^k) + \nabla J_R(u^k)).$$

We can first observe that $\nabla J_D(u) = (u - f) \in X$. Next, we detail the algorithm for different regularizers J_R.

Easy homework: Check the strict convexity and the computation of the gradient of the function J_D.

Tikhonov regularization The so-called Tikhonov regularization is $J_R(u) = J_H(u) = ||\nabla u||_X^2$. This function enforces the image to be ***smooth***. The corresponding the gradient descent algorithm reads:

$$u^{k+1} = u^k + \tau(\lambda(f - u^k) + \Delta u^k), \tag{3.19}$$

where the time step can be taken as $\tau = 1/(\lambda + 4)$.

Smoothed Total Variation regularization The so-called smoothed Total Variation regularization is $J_R(u) = J_{TV}^\epsilon(u) := \sum_{x\in\Omega} \sqrt{||\nabla u(x)||^2 + \epsilon}$. This function enforces the image to be ***piecewise constant***. The gradient descent algorithm is in this case:

$$u^{k+1} = u^k + \tau\left(\lambda(f - u^k) + \operatorname{div}\left(\frac{\nabla u^k}{\sqrt{||\nabla u^k||^2 + \epsilon}}\right)\right), \tag{3.20}$$

where the time step τ must be taken small enough to avoid numerical instabilities (see Remark page 13). With the previous notations, we have $J_{PM} = J_{TV}^1$. Taking $\epsilon = 1$ gives a good approximation of the Total Variation regularization if the gray values of f are within the range $[0; 255]$.

PDE point of view It is interesting to interpret the algorithms (3.19) and (3.20) as PDEs. For the Tikhonov regularization, one recovers:

$$\begin{cases} \frac{\partial u(t,x)}{\partial t} &= \lambda(f(x) - u(t,x)) + \Delta u(t,x) & \text{for } t \geq 0 \text{ and } x \in \Omega \\ u(0,x) &= f(x) & \text{for } x \in \Omega \\ \frac{\partial u(t,x)}{\partial N} &= 0 & \text{for } t > 0 \text{ and } x \in \partial\Omega. \end{cases} \tag{3.21}$$

With respect to the Heat equation and the Perona-Malink one, the additional term $\lambda(f(x) - u(t,x))$ now enforces, the solution $u(t,x)$ to go into the direction of $f(x)$, with an influence given by λ.

3.4 Back to work

We can now solve the problem (3.18) for the different regularizers.

11 Write a function *Denoise_ Tikhonov* that

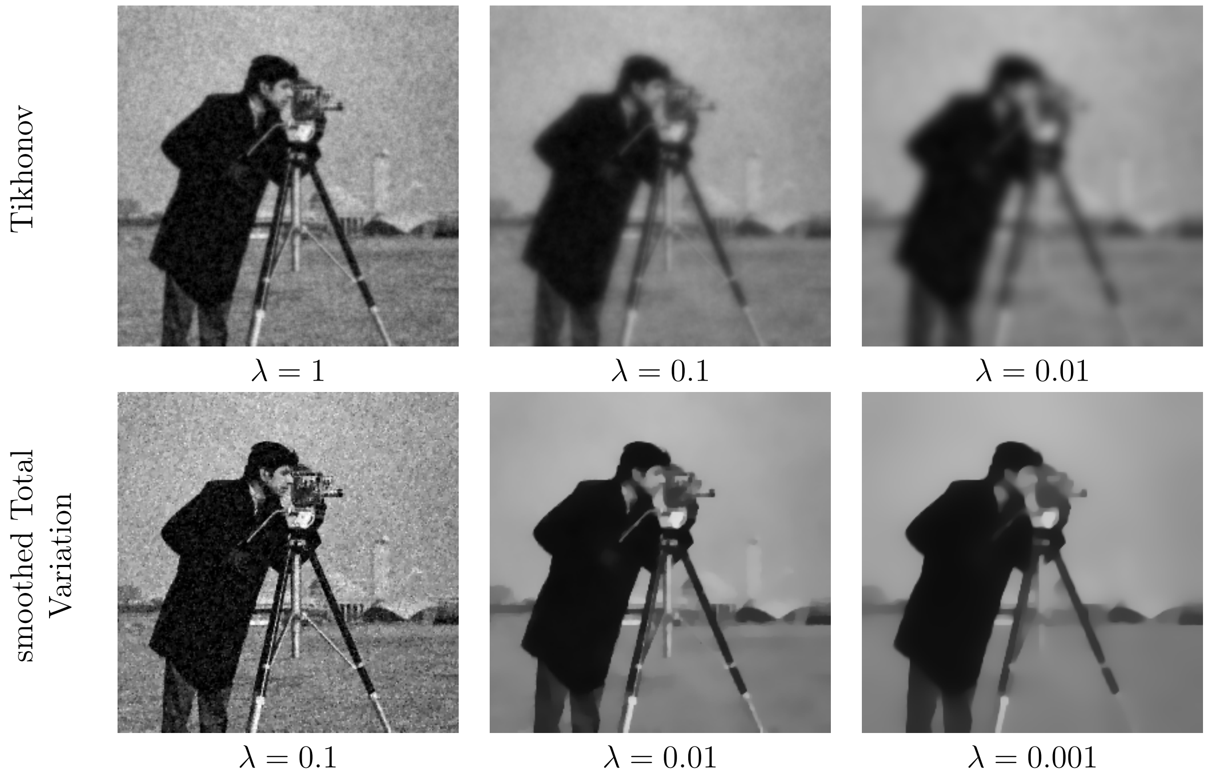
- takes as argument the noisy image f, a time step τ (that can be automatically set to $1/(\lambda+4)$), a number of iterations K and a parameter λ
- realizes the gradient descent algorithm (3.19) for iterations $k = 1 \cdots K$
- returns u^K.

12 Write a function *Denoise_ TV* that

- takes as argument the noisy image f, a time step τ, a number of iterations K and parameters λ and ϵ
- realizes the gradient descent algorithm (3.20) for iterations $k = 1 \cdots K$
- returns u^K.

13 Write a script that test these functions on a noisy image f for different parameters λ.

Remark: The parameter K is a maximum number of iteration. The algorithm can be stopped if a convergence criteria is met. A standard criteria is to measure the normalized root-mean-square error (RMSE) between successive iterations: $||u^{k+1} - u^k||/||u^k||$ and stop the algorithm when it is small enough (for instance $< 10^{-5}$ for Tikhonov and $< 10^{-4}$ for smoothed Total Variation). Also notice that since the Tikhonov regularization is quadratic and the Total Variation one in piecewise linear, good values of λ are not in the same range for the 2 regularizations. An example of the obtained results is given below.



Denoising results for different values of λ. First line: Minimizer u_λ of J_λ with Tikhonov regularization. Second line: Minimizer u_λ with smoothed Total Variation regularization.

3.5 Solving Tikhonov regularization with Fourrier Transform

The Tikhonov regularization corresponds to solve the problem:

$$\min_u J_\lambda(u) = \lambda J_D(u) + J_H(u) = \frac{\lambda}{2}||u-f||^2_X + \frac{1}{2}||\nabla u||^2_Y. \tag{3.22}$$

The minimizer u of this convex function is then characterized by $\nabla J_\lambda(u) = 0$. Computing the Euler-Lagrange equation of (3.22), it gives as optimality condition:

$$\lambda(u-f) - \Delta u = 0. \tag{3.23}$$

As for the Heat equation, whose solution $u(t,x)$ can be explicitly obtained through a convolution with an adequate kernel depending on t (see relation (2.2)), the solution of the Tikhonov regularization problem can be explicitly computed (i.e. without minimizing J_λ iteratively).

The solution can indeed be exhibited by considering Discrete Fourier Transform (DFT) of the optimality condition (3.23).

We recall that the DFT of a $m \times n$ discrete image $f(k,l)$ ($0 \leq k \leq m-1$ et $0 \leq l \leq n-1$) is given, for $0 \leq p \leq m-1$ and $0 \leq q \leq n-1$, by:

$$\mathcal{F}(f)(p,q) = F(p,q) = \sum_{i=0}^{m-1}\sum_{j=0}^{n-1} f(k,l)e^{-j(2\pi/m)pk}e^{-j(2\pi/n)ql} \tag{3.24}$$

and the inverse transform is:

$$f(k,l) = \frac{1}{mn}\sum_{p=0}^{m-1}\sum_{q=0}^{n-1} F(p,q)e^{j(2\pi/m)pk}e^{j(2\pi/n)ql} \tag{3.25}$$

One can show that, for the centered discretization of the Laplacian operator (2.12) and periodic conditions, we have

$$\mathcal{F}(\Delta f)(p,q) = -4\mathcal{F}(f)(p,q)\left(\sin^2\left(\frac{\pi p}{m}\right) + \sin^2\left(\frac{\pi q}{n}\right)\right) \tag{3.26}$$

from which we can deduce that the minimizer u of (3.22) checks:

$$\mathcal{F}(u)(p,q) = \frac{\lambda\mathcal{F}(f)(p,q)}{\lambda + 4\left(\sin^2\left(\frac{\pi p}{m}\right) + \sin^2\left(\frac{\pi q}{n}\right)\right)} \tag{3.27}$$

Homework:

- Show relations (3.26) and (3.27). Recall: $2\sin^2(a) = 1 - \cos(2a)$.
- Implement relation (3.27) and find the minimizer using functions *fft2* and *ifft2*. Compare the solution with the one obtained with the *Denoise_ Tikhonov* function for the same λ.

Remark: The above DFT method assumes periodic conditions, while Neumann conditions were previously considered. As a consequence, differences between both approaches should be mainly visible on the image boundaries.

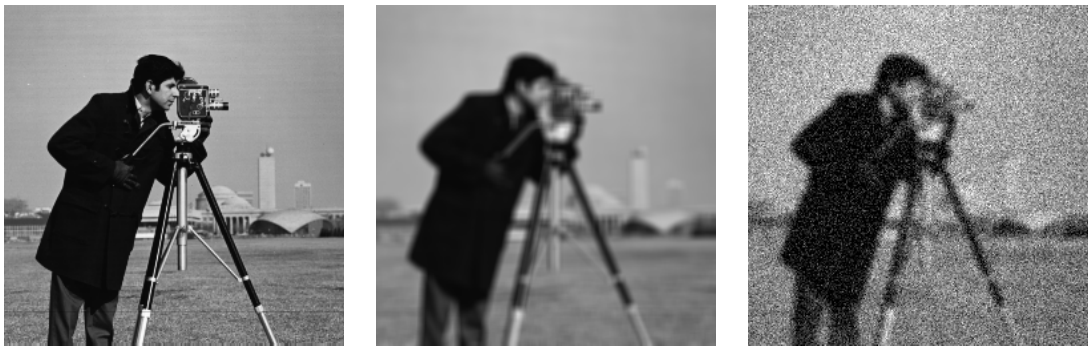
4 Extensions to deconvolution and inpainting

We now extend the previous variational model to a more general one dealing with other applications than denoising. To that end, instead of the data function J_D, we define a new data function J_A that will be minimized jointly with the two studied regularization functions. This data function reads:

$$J_A(u) = \frac{1}{2}\sum_{x\in\Omega} ||(Au)(x) - f(x)||^2 = \frac{1}{2}||Au - f||_X^2, \tag{4.28}$$

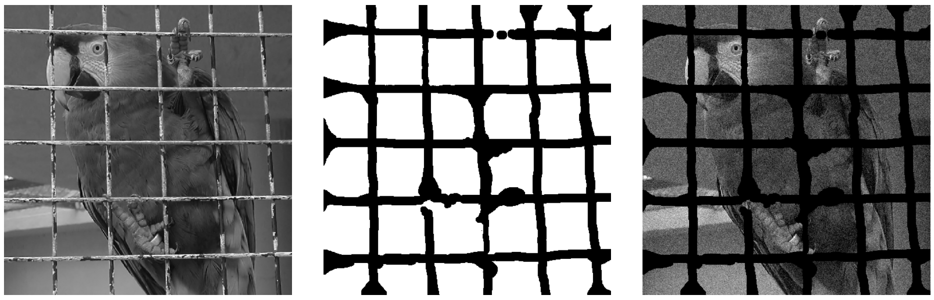
where A is a linear operator from X to X that can be represented as $mn \times mn$ matrix in the discrete setting. This data function can model different interesting applications:

- **Deconvolution**: We assume that the available image f is obtained as $G * u^* + \omega$, where u^* is the unknown ground truth image to recover, $G(x)$ is a known $2D$ convolution filter and ω is an additional Gaussian noise. In this case, the data function $||G*u-f||_X^2$ tries to estimate an image u which convolution with the kernel G is close to the one of u^*. With adequate change of indexes, the discrete convolution of u with a kernel K can be seen as a matrix vector multiplication, by considering the $mn \times mn$ matrix $G_{kl} = G(x_l - x_k)$.



Example of data f in case of deconvolution.

- **Inpainting**: We assume that the available image f is a partial observation of a perturbation of the ground truth image u^* to recover. More precisely, we only have access to observations of some pixels x that belong to a known region $\mathcal{M} \subset \Omega$. In this case, the data term can be taken as $\sum_{x\in\mathcal{M}} ||u(x) - f(x)||^2$. It corresponds to the framework of (4.28) by introducing the $mn \times mn$ matrix $M(k,l) = 1$ if $k = l$ and pixel $x_k \in \mathcal{M}$ and 0 otherwise.



Example of data f in case of inpainting.

Remark: Notice that J_A is strictly convex only if A is a positive definite matrix and thus an invertible matrix. In this case, one can equivalently consider as data term $||u - A^{-1}f||_X^2$ which enters in the framework of previous section. Hence, the interesting case to look at is when A has zero eigenvalues, for instance when A is positive semi-definite.

4.1 Gradient of J_A

The gradient of function (4.28) reads $\nabla J_A(u) = A^T(A(u-f)$. In the aforementioned applications one has:

- **Deconvolution**: With an isotropic kernel G, the gradient of $J_G(u) = \frac{1}{2}||G * u - f||_X^2$ is

$$\nabla J_G(u) = G * (G * u - f). \tag{4.29}$$

- **Inpainting**: The gradient of $J_{\mathcal{M}}(u) = \frac{1}{2}\sum_{x\in\mathcal{M}} ||u(x) - f(x)||^2$ is

$$\nabla J_{\mathcal{M}}(u)(x) = \begin{cases} u(x) - f(x) & if\, x \in \mathcal{M} \\ 0 & otherwise. \end{cases} \tag{4.30}$$

  Considering the mask function $M(x) = 1$ if $x \in \mathcal{M}$ and 0 otherwise, we also have: $\nabla J_{\mathcal{M}}(u) = (u - f)M$.

4.2 Variational models and minimization

We can now use the new data functions together with the regularization ones, in order to formalize and solve the deconvolution and inpainting problems.

- **Deconvolution**: The function to minimize reads:

$$\frac{\lambda}{2}||G * u - f||_X^2 + \sum_{x\in\Omega} \sqrt{||\nabla u(x)||^2 + 1}.$$

  Minimizing this function corresponds to find a piecewise constant image u whose convolution with G is close to f. As the data f is assumed to be blurred, considering a Tikhonov regularization is in this case not very smart, since it would not produce a sharp enough image. The gradient descent algorithm applied to this problem gives:

$$u^{k+1} = u^k + \tau \left(\lambda G * (f - G * u^k) + \text{div} \left(\frac{\nabla u^k}{\sqrt{||\nabla u^k||^2 + 1}} \right) \right), \tag{4.31}$$

- **Inpainting**: The function to minimize reads:

$$\frac{\lambda}{2} \sum_{x\in\mathcal{M}} ||u(x) - f(x)||^2 + \sum_{x\in\Omega} \sqrt{||\nabla u(x)||^2 + 1}.$$

  The idea is to diffuse the known information into the region $\mathcal{M}$ to the masked regions $\Omega\backslash\mathcal{M}$ with the regularization function. We thus obtain:

$$u^{k+1} = u^k + \tau \left(\lambda(f - u^k)M + \text{div} \left(\frac{\nabla u^k}{\sqrt{||\nabla u^k||^2 + 1}} \right) \right), \tag{4.32}$$

The Tikhonov regularization is of interest in this application, namely in almost uniform image regions (sky, water...), as it will realize an isotropic diffusion of the known information. The corresponding gradient descent algorithm is:

$$u^{k+1} = u^k + \tau \left(\lambda(f - u^k)M + \Delta u^k\right). \tag{4.33}$$

14 Write a function *Deconvolution_ TV* that

- takes as argument an image f, a kernel G, a time step τ, a number of iterations K and a parameter λ and returns u^K (see Page 7 for image convolution in Matlab)
- realizes the gradient descent algorithm (4.31) for iterations $k = 1 \cdots K$

15 Write a function *Inpainting_ TV* that

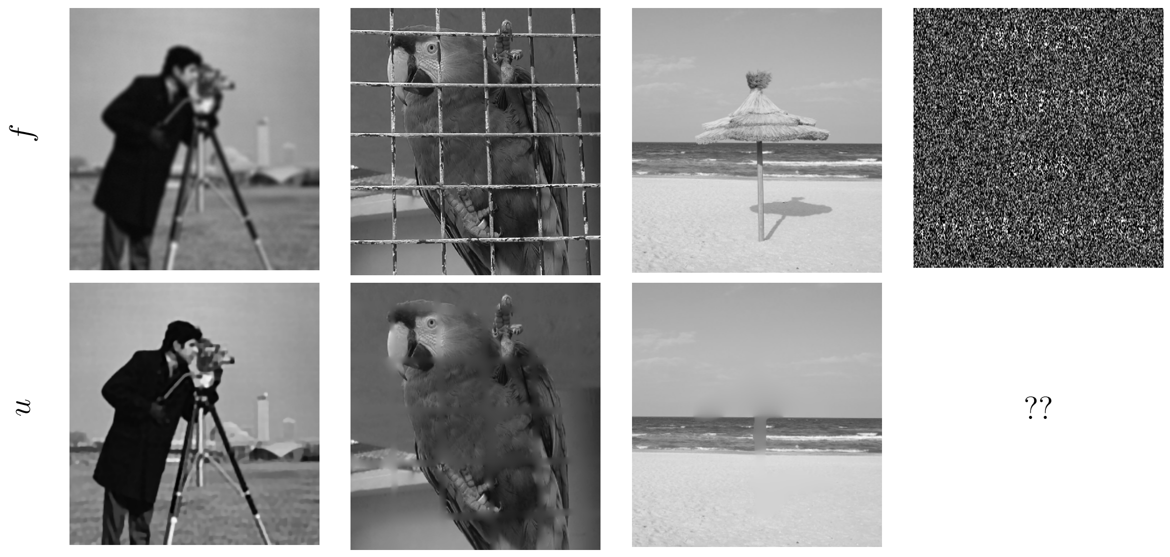
- takes as argument an image f, a mask image M, a time step τ, a number of iterations K and parameter λ and returns u^K
- realizes the gradient descent algorithm (4.32) f for iterations $k = 1 \cdots K$

16 Implement the function *Inpainting_ Tichonov* that

- takes as argument an image f, a mask image M, a time step τ, a number of iterations K and parameter λ and returns u^K
- realizes the gradient descent algorithm (4.33) f for iterations $k = 1 \cdots K$

17 Write a script that test, for different parameters λ, these functions on images f obtained as follows

- Deconvolution: convolve the image of your choice with a Kernel G (for instance *G= fspecial('gaussian',[7 7],5);*) and add noise. Give the same G to your function *Deconvolution_ TV*
- Inpainting: Take a large value for K and use the images and masks available here: Image 1, Mask 1, Image 2, Mask 2, Image 3, Mask 3.



First column: Deconvolution with smoothed Total Variation. Second column: Inpainting with smoothed total variation. Third column: Inpaitning with Tikhonov. Last column: Inpainting