# Optimization-based Advanced Image Processing

Master VIBOT, Le Creusot, France

## 1 Sparse Regularization Algorithm

### 1.1 Sparse Regularization Principle

Let measurements be $y = \Phi f_0$ where $\Phi$ is a masking operator.

Image is recovered using sparsity from the measurements $y$. It considers a synthesis-based regularization, that compute a sparse set of coefficients $(a^\star_m)_m$ in a frame $\Psi = (\psi_m)_m$ that solves

$$a^\star \in \operatorname{argmin}_a \frac{1}{2}\|y - \Phi\Psi a\|^2 + \lambda J(a),$$

where $\lambda$ should be adapted to the noise level $\|w\|$.

$\Psi a$ indicates the synthesis operator or dictionary, and $J(a)$ is the $\ell_1$ sparsity prior:

$$J(a) = \sum_m \|a_m\|.$$

### 1.2 Damaged Image

Images can be loaded and rescaled as following:

```
n = 256;
f0 = double(imread('cameraman.png'));
f0 = f0(1:n,1:n);
clf;
imshow(f0, []);
```

In the framework of an inpainting problem, we consider a linear imaging operator $\Phi : f \mapsto \Phi(f)$ that maps high resolution images to low dimensional observations. A masking operator $\Phi$ has to be computed from a random mask $\Lambda$:

```
rho = .8;
Lambda = rand(n,n)>rho;
Phi = @(f) f.*Lambda;
```

Parameter $\rho$ ajusts the damage. Observations $y = \Phi f_0$ have computed as following:

```
y = Phi(f0);
imshow(y, []);
```

#### 1.2.1 Inpainting processing

The soft thresholding operator is the simplest $\ell_1$ minimization schemes. It can be applied to coefficients $a$, or to an image $f$ in an ortho-basis. The soft thresholding is a 1-D functional that shrinks the value of coefficients.

$$s_T(u) = \max\left(0, 1 - \frac{T}{|u|}\right) u,$$

with $T$ the threshold.

```
SoftThresh = @(x,T)x.*max(0, 1-T./max(abs(x),1e-10));
```

Note that the function SoftThresh can also be applied to vector which defines an operator on coefficients:

$$S_T(a) = (s_T(a_m))_m.$$

We use an orthogonal wavelet basis $\Psi$ and we set the following parameters:

```
Jmax = log2(n)-1;
Jmin = Jmax-3;
options.ti = 0; % use orthogonality.
Psi = @(a)perform_wavelet_transf(a, Jmin, -1,options);
PsiS = @(f)perform_wavelet_transf(f, Jmin, +1,options);
```

The soft thresholding operator in the basis $\Psi$ corresponds to applying the transform $\Psi^*$, thresholding the coefficients using $S_T$ and then undoing the transform using $\Psi$:

$$S_T^{\Psi}(f) = \Psi \circ S_T \circ \Psi^*$$

This process on $f_0$ can be computed and displayed with:

```
SoftThreshPsi = @(f,T)Psi(SoftThresh(PsiS(f),T));
imshow(SoftThreshPsi(y,.1), []);
```

Implement the algorithm with a static threshold and analyse the evolution of the restored images according to the iterations.

We can improve the reconstruction result by using a soft thresholding 1-D functional.

```
T = linspace(-1,1,1000);
plot(T, SoftThresh(T,.5));
```

Implement the algorithm and analyse the evolution of the restored images according to the iterations. Use different images and different values of the damage parameter $\rho$.

## 2 Primal-Dual Total Variation Algorithm

### 2.1 Douglas-Rachford Algorithm

The Douglas-Rachford algorithm is an iterative scheme to minimize functionals of the form

$$\min_x F(x) + G(x),$$

where $F$ and $G$ are convex functions, of which one is able to compute the proximity operators. This algorithm was is a generalization of an algorithm introduced by Douglas and Rachford in the case of quadratic minimization.

The Douglas-Rachford algorithm takes an arbitrary element $s^{(0)}$, a parameter $\gamma > 0$, a relaxation parameter $0 < \rho < 2$ and iterates, for $k = 1, 2, \dots$

$$\left| \begin{array}{l} x^{(k)} = \mathrm{prox}_{\gamma F}(s^{(k-1)}), \\ s^{(k)} = s^{(k-1)} + \rho\big(\mathrm{prox}_{\gamma g}(2x^{(k)} - s^{(k-1)}) - x^{(k)}\big). \end{array} \right.$$

It is of course possible to inter-change the roles of $F$ and $G$, which defines a different algorithm. The iterates $x^{(k)}$ converge to a solution $x^\star$ of the problem, i.e. a minimizer of $f = F + G$.

### 2.2 Convex Optimization with a Primal-Dual Proximal Splitting

We consider general optimization problems of the form

$$\min_f F(K(f)) + G(f),$$

where $F$ and $G$ are convex functions and $K : f \mapsto K(f)$ is a linear operator.

To apply the primal-dual algorithm, the proximal mapping of $F$ and $G$ can be computed from:

$$\text{Prox}_{\gamma F}(x) = \underset{y}{\text{argmin}}\ \frac{1}{2}\|x-y\|^2 + \gamma F(y),$$

The numerical processes are:

$$g_{k+1} = \text{Prox}_{\gamma F^*}(g_k + \gamma K(\tilde{f}_k)),$$
$$f_{k+1} = \text{Prox}_{\tau G}(f_k - \tau K^*(g_k)),$$
$$\tilde{f}_{k+1} = f_{k+1} + \theta(f_{k+1} - f_k).$$

And the dual functional is defined as following:

$$F^*(y) = \max_x \langle x, y\rangle - F(x).$$

To compute the proximal mapping of $F$ is equivalent to compute the proximal mapping of $F^*$ thanks to Moreau's identity:

$$x = \text{Prox}_{\tau F^*}(x) + \tau \text{Prox}_{F/\tau}(x/\tau).$$

### 2.3 Work: Inpainting using Primal-dual Total Variation Regularization scheme

The aim is to implement the primal-dual total variation algorithm and inpaint an image from an inverse problem using a total variation regularization:

$$\min_{y=\Phi f} \|\nabla f\|_1$$

For the application, the minimization of $F(K(f)) + G(f)$ can be rewritten as:

$$G(f) = i_H(f), \quad F(u) = \|u\|_1 \quad \text{and} \quad K = \nabla,$$

with $i_H$ the indicator function

These operators can be computed as following:

```
K  = @(f)grad(f);
KS = @(u)-div(u);
Amplitude = @(u)sqrt(sum(u.^2,3));
F = @(u)sum(sum(Amplitude(u)));
```

The proximal operator of the vectorial $\ell_1$ norm reads

$$\text{Prox}_{\lambda F}(u) = \max\left(0, 1 - \frac{\lambda}{\|u_k\|}\right) u_k$$

```
ProxF = @(u,lambda)max(0,1-lambda./repmat(Amplitude(u), [1 1 2])).*u;
```

A simple way to compute the proximal operator of the dual function $F^*$, we make use of Moreau's identity:

$$x = \text{Prox}_{\tau F^*}(x) + \tau \text{Prox}_{F/\tau}(x/\tau)$$

```
ProxFS = @(y,sigma)y-sigma*ProxF(y/sigma,1/sigma);
```

The proximal operator of $G = i_H$ is the projector on $H$. In our case, since $\Phi$ is a diagonal so that the projection is simple to compute

$$\mathrm{Prox}_{\tau G}(f) = \mathrm{Proj}_H(f) = f + \Phi(y - \Phi(f))$$

```
ProxG = @(f,tau)f + Phi(y - Phi(f));
```

We set parameters for the algorithm. In our case, $L = \|K\|^2 = 8$ and we need to have $L\sigma\tau < 1$.

```
L = 8;
sigma = 10;
tau = .9/(L*sigma);
theta = 1;
f = y;
g = K(y)*0;
f1 = f;
```

Example of one iteration.

```
fold = f;
g = ProxFS( g+sigma*K(f1), sigma);
f = ProxG(f-tau*KS(g), tau);
f1 = f + theta * (f-fold);
```

Implement the algorithm for several iterations (the number can be important) and analyse the evolution of the restored images according to the iterations. Plot the evolution of the Total Variation (TV) energy $F(K(fk))$ ($\ell_1$-norm of derivatives). Use different images and different values of the damage parameter $\rho$.

Display result $f$.

## 3 Image Denoising via Sparse Representation

### 3.1 Dictionary Learning Principle

#### 3.1.1 Dictionary Learning as an Optimization Problem

Given a set $Y = (y_j)_{j=1}^m \in \mathbb{R}^{n\times m}$ of $m$ signals $y_j \in \mathbb{R}^m$, dictionary learning aims at finding the best dictionary $D = (d_i)_{i=1}^p$ of $p$ atoms $d_i \in \mathbb{R}^n$ to sparse code all the data.

In image denoising, each $y_j \in \mathbb{R}^n$ is a patch of size $n = w \times w$ extracted from the noisy image.

The sparse coding of a single data $y = y_j$ for some $j = 1, \dots, m$ is obtained by minimizing the following constrained optimization

$$\min_{\|x\|_0 \le k} \frac{1}{2}\|y - Dx\|^2.$$

where parameter $k > 0$ controls the amount of sparsity.

Dictionary learning performs an optimization on the dictionary $D$ and on the set of coefficients $X = (x_j)_{j=1}^m \in \mathbb{R}^{p\times m}$ where $x_j$ ($j = 1, \dots, m$) is the set of coefficients of the data $y_j$. The optimization process is written as follow:

$$\min_{D\in\mathcal{D}, X\in\mathcal{X}_k} E(X, D) = \frac{1}{2}\|Y - DX\|^2 = \frac{1}{2}\sum_{j=1}^m \|y_j - Dx_j\|^2.$$

The constraint set on $D$ is

$$\mathcal{D} = \{D \in \mathbb{R}^{n\times p}, \forall i = 1, \dots, p, \quad \|D_{\cdot,i}\| \le 1\},$$

Note that the columns of the dictionary are unit normalized.

First we propose to optimize the coefficients $X$

$$X^{(\ell+1)} \in \underset{X \in \mathcal{X}_k}{\operatorname{argmin}} E(X, D^{(\ell)}),$$

and then to update the dictioonary with the following minimization

$$D^{(\ell+1)} \in \underset{D \in \mathcal{D}}{\operatorname{argmin}} E(X^{(\ell+1)}, D).$$

### 3.1.2 Patch Definition

We want to apply the dictionary learning method to denoising. We thus consider a noisy image obtained by adding Gaussian noise to a clean, unknown image $f_0 \in \mathbb{R}^N$ where $N = n_0 \times n_0$.

We add white Gaussian noise on $f$.

Compute the dictionary $D \in \mathbb{R}^{n \times p}$. You define width $w$ of the patches, number $p$ of atoms and dimension $n = w \times w$ of the sparse data.

```
w = 10;  % Width w of the patches.
n = w*w; % Dimension n=wxw of the data to be sparse coded.
p = 2*n; % Number of atoms p in the dictionary.
m = 20*p; % Number m of patches used for the training.
k = 4; % Target sparsity
%
sd = .06; % Gaussian noise standard deviation
n0 = 256; % Image size
f0 = imresize(imread('lena.bmp'),[n0 n0]) );
f = f0 + sd*randn(n0);
imshow(f, []     );
```

The random patch locations are generated:

```
q = 3*m;
x = floor( rand(1,1,q)*(n0-w) )+1;
y = floor( rand(1,1,q)*(n0-w) )+1;
```

The lots of patches $y_j \in \mathbb{R}^n$ are extracted and stored in a matrix $Y = (y_j)_{j=1}^m$.

```
[dY,dX] = meshgrid(0:w-1,0:w-1);
Xp = repmat(dX,[1 1 q]) + repmat(x, [w w 1]);
Yp = repmat(dY,[1 1 q]) + repmat(y, [w w 1]);
Y = f(Xp+(Yp-1)*n0);
Y = reshape(Y, [n q]);
```

The mean is removed, since we are going to learn a dictionary of zero-mean and unit norm atom and those with largest energy are kept.

```
Y = Y - repmat( mean(Y), [n 1] );
[tmp,I] = sort(sum(Y.^2), 'descend');
Y = Y(:,I(1:m));
```

We consider a dictionary $D \in \mathbb{R}^{n \times p}$ of $p \geq n$ atoms in $\mathbb{R}^n$. The initial dictionary $D$ is computed by a random selection of patches, and we normalize them to be unit-norm and the initial dictionary are displayed

```
ProjC = @(D)D ./ repmat( sqrt(sum(D.^2)), [w^2, 1] );
sel = randperm(m); sel = sel(1:p);
D0 = ProjC( Y(:,sel) );
D = D0;
plot_dictionnary(D, [], [8 12]);
```

### 3.1.3 Coefficient Update

The optimization on the coefficients $X$ requires, for each $y_j = Y_{\cdot,j}$ to compute $x_j = X_{\cdot,j}$ that solves

$$\min_{\|x_j\|_0 \leq k} \frac{1}{2}\|y - Dx_j\|^2.$$

We solve this method using thr projected gradient descent method:

$$x_j \leftarrow \text{Proj}_{\mathcal{X}_k}(x_j - \tau D^*(Dx_j - y)) \quad \text{where} \quad \tau < \frac{2}{\|DD^*\|}.$$

```
select  = @(A,k)repmat(A(k,:) , [size(A,1)  1]);
ProjX  = @(X,k)X .* (abs(X) >= select(sort(abs(X), 'descend'),k));
```

### 3.1.4 Dictionary Update

The dictionary is updated by performing the following minimization

$$\min_{D \in \mathcal{D}} \frac{1}{2}\|Y - DX\|^2.$$

We also solve this minimization with a projected gradient descent

$$D \leftarrow \text{Proj}_{\mathcal{C}}(D - \tau(DX - Y)X^*)$$

where $\tau < 2/\|XX^*\|$.

Note that the orthogonal projector $\text{Proj}_{\mathcal{C}}$ is implemented in the function *ProjC* already defined.

```
dictionary_learning;
plot_dictionnary(D,X, [8 12]);
```

Test the algorithm with different noise intensities.

## 3.2 Denoising by Sparse Coding

### 3.2.1 Patch Extraction

Here we apply sparse coding in the learned dictionary to the problem of image denoising.

The method first extracts lots of patches from $f$, then perform sparse coding of each patch in $D$, and then average the overlapping patch to obtained the denoised image $f_1$.

We extract a large number $m$ of patches $Y = (y_j)_{j=1}^m \in \mathbb{R}^{n\times m}$ from the noisy image $f$.

In practice, we use an overlap factor to reduce blocking artifact and obtain good denoising performance.

```
q = 2; % Overlap factor
%
%% Space positions for the extraction of patches
 [y,x] = meshgrid(1:q:n0-w/2, 1:q:n0-w/2);
m = size(x(:),1);
Xp = repmat(dX,[1 1 m]) + repmat( reshape(x(:),[1 1 m]), [w w 1]);
Yp = repmat(dY,[1 1 m]) + repmat( reshape(y(:),[1 1 m]), [w w 1]);
```

Ensure boundary conditions.

```
Xp(Xp>n0) = 2*n0-Xp(Xp>n0);
Yp(Yp>n0) = 2*n0-Yp(Yp>n0);
```

Extract the $m$ patches $Y$.

```
Y = f(Xp+(Yp-1)*n0);
Y = reshape(Y, [n, m]);
```

Save the mean $\theta_j$ of each patch and remove it.

```
theta = mean(Y);
Y = Y - repmat( theta, [n 1] );
```

Denoising of the patches is obtained by performing a sparse coding of each patch $y_j$ in $D$

$$\min_{\|Dx_j - y_j\| \leq \epsilon} \|x_j\|_1.$$

The value of $\epsilon$ is set proportionaly to the noise level $\sqrt{n}\sigma$ that contaminates each patch.

```
rho = .95; %Proportionality factor
epsilon = rho*sqrt(n)*sigma;
```

### 3.2.2 Sparse coding problem

The sparse coding problem can written using proximal splitting schemes. We introduce an auxiliary variable $u = Dx \in \mathbb{R}^n$:

$$\min_{z=(x,u)\in\mathbb{R}^p\times\mathbb{R}^n} F(z) + G(z)$$

with $z = (x, u)$ and

$$F(x,u) = \|x\|_1 + \iota_{B_\epsilon(y)}(u) \quad \text{where} \quad B_\epsilon(y) = \{u, \|u - y\| \leq \epsilon\}$$

and

$$G(x,u) = \iota_{\mathcal{C}}(x,u) \quad \text{where} \quad \mathcal{C} = \{(x,u), u = Dx\}.$$

To minimize the sparse coding problem, we make use of a proximal splitting scheme (Douglas-Rachford algorithm) to minimize an energy of the form $F(z) + G(z)$ (see section 2.1).

In the special case of the constrained sparse coding problem, the proximal mapping of $G$ is the orthogonal projection on the convex set $\mathcal{C}$:

$$\text{Prox}_{\gamma G}(x,u) = \text{Proj}_{\mathcal{C}}(x,u).$$

It can be computed by solving a linear system of equations since

$$\tilde{u} = D\tilde{x} \quad \text{where} \quad \tilde{x} = (\text{Id} + D^*D)^{-1}(f + D^*u).$$

$\text{Proj}_{\mathcal{C}}$ is so defined by computing the inverse of $(\text{Id} + D^*D)$.

```
U = (eye(p) + D'*D)^(-1);
Replicate = @(z)deal(z, D*z);
ProjC = @(x,u)Replicate( U*( x + D'*u ) );
ProxG = @(f,u,gamma)ProjC(f,u);
```

Function $F(x,u)$ is actually a separable sum of a function that only depends on $x$ and a function that depends only on $u$:

$$F(x,u) = \iota_{B_\epsilon(y)}(u) + \|x\|_1.$$

The proximal operator of $F$ reads

$$\text{Prox}_{\gamma F}(x,u) = (\text{Proj}_{B_\epsilon(y)}(u), \text{Prox}_{\gamma\|\cdot\|_1}(x)).$$

We define $y = Y$ to be the set of all the patches.

```
y = Y(:,1:m);
```

And we define the projector

$$\mathrm{Proj}_{B_\epsilon(y)}(u) = y + (u-y)\max\left(1, \frac{\epsilon}{\|u-y\|}\right)$$

```
amplitude = @(a)repmat( sqrt(sum(a.^2,1)), [n 1] );
ProjB = @(u)y + (u-y) .* min(1, epsilon./amplitude(u-y) );
```

The proximal operator of the $\ell_1$ norm $\|\cdot\|_1$ is a soft thresholding:

$$\mathrm{Prox}_{\gamma\|\cdot\|_1}(x)_i = \max\left(0, \frac{\gamma}{|x_i|}\right) x_i.$$

```
ProxL1 = @(x,gamma)max(0,1-gamma./max(1e-9, abs(x))) .* x;
```

The proximal operator of $F$ is then:

```
ProxF = @(x,u,gamma)deal( ProxL1(x,gamma), ProjB(u) );
```

Convergence parameters $\mu$ and $\gamma$ and the number of iterations are be set. The Douglas-Rachford algorithm

```
mu = 1;
gamma = 1;
niter = 800;
%
tx = zeros(p,size(y,2));
tu = D*tx;
E = [];
E1 = [];
for i=1:niter
    tx1 = tx; tu1 = tu;
    % rProxG
    [tx2,tu2] = ProxG(tx1, tu1, gamma);
    tx1 = 2*tx2 - tx1; tu1 = 2*tu2 - tu1;
    % rProxF
    [tx2,tu2] = ProxF(tx1, tu1, gamma);
    tx1 = 2*tx2 - tx1; tu1 = 2*tu2 - tu1;
    % Average
    tx = (1-mu/2)*tx + mu/2*tx1;
    tu = (1-mu/2)*tu + mu/2*tu1;
    % Converging sequence
    [x,u] = ProxG(tx,tu,gamma);
end
```

Once each $x_j$ is computed, one obtains the denoised patch $\tilde{y}_j = Dx_j$. These denoised patches are then aggregated together to obtained the denoised image:

$$f_1(t) = \frac{1}{W_t}\sum_j \tilde{y}_j(t-a_j)$$

where $W_t$ is the number of patches that overlap at a given pixel location $t$. Approximated patches are denoted $Y_1 = (\tilde{y}_j)_j = DX$.

```
Y1 = reshape(D*x, [w w m]);
```

Insert back the mean.

```
Y1 = Y1 - repmat( mean(mean(Y1)), [w w] );
Y1 = Y1 + reshape(repmat( theta, [n 1] ), [w w m]);
```

To obtain the denoising, we average the value of the approximated patches $Y_1$ that overlap.

```
W = zeros(n0,n0);
f1 = zeros(n0,n0);
for i=1:m
    x = Xp(:,:,i); y = Yp(:,:,i);
    f1(x+(y-1)*n0) = f1(x+(y-1)*n0) + Y1(:,:,i);
   W(x+(y-1)*n0) = W(x+(y-1)*n0) + 1;
end
f1 = f1 ./ W;
```

Display the result. Test with different noise values.

```
clf;
imshow(f1, []);
SNR=' snr(f0,f1);
```

Compare obtained results with wavelet hard thresholding.

```
Jmin = 3;
options.ti = 1;
W = perform_wavelet_transf(f, Jmin, +1, options);
Wt = perform_thresholding(W, 2.8*sigma, 'hard')
Res = perform_wavelet_transf(Wt, Jmin, -1, options);
imshow(Res, []);
```

## References

[1] Thomas Blumensath and Mike E. Davies, *Iterative thresholding for sparse approximations*, Journal of Fourier Analysis and Applications, 2008.

[2] Antonin Chambolle and Thomas Pock, *A First-order primal-dual algorithm for convex problems with application to imaging*, Journal of Mathematical Imaging and Vision, 2011

[3] Patrick L. Combettes and Jean-Christophe Pesquet, *Proximal Splitting Methods in Signal Processing*. In: Fixed-point algorithms for inverse problems in science and engineering. Springer New York, 2011.

[4] Laurent Condat, *A primal-dual splitting method for convex optimization involving Lipschitzian, proximable and linear composite terms*, J. Optimization Theory and Applications, 2013

[5] M. Elad and M. Aharon, Image Denoising Via Sparse and Redundant representations over Learned Dictionaries, IEEE Trans. on Image Processing, December 2006.

[6] P. L. Lions and B. Mercier, *Splitting Algorithms for the Sum of Two Nonlinear Operators*, SIAM Journal on Numerical Analysis, 1979

[7] BA Olshausen, and DJ Field, *Emergence of Simple-Cell Receptive Field Properties by Learning a Sparse Code for Natural Images*, Nature, 1996.

[8] Gabriel Peyré, *Numerical Tours of Data Sciences*, `http://www.numerical-tours.com/`

[9] P. Tseng, *Convergence of Block Coordinate Descent Method for Nondifferentiable Minimization*, J. Optim. Theory Appl., 109, 2001