ECE 285 – Project B
Total Variation

Written by Charles Deledalle on May 27, 2019.

You will have to submit a notebook `projectB.ipynb` and the package `imagetools/projectB.py`. Organize your notebook with headings (following the numbering of the questions). For writing questions, answer directly in your notebook in markdown cells. For each section, it is indicated in brackets how much it contributes to the grade.

This project focuses on image restoration with total variation. Before starting this project you will need to have gone through all assignments. Functions developed in this project will complete the `imagetools` package. We will be using the following assets

- `assets/starfish.png`
- `assets/flowers.png`
- `assets/ball.png`

1 Operators (25%)

We focus on the estimation of a clean image x_0 form its degraded observation y satisfying

$$y = \mathbf{H}x_0 + w$$

where w is a white Gaussian noise component with standard deviation σ, and $\mathbf{H}$ a linear operator. We will consider three types of linear operators: identity (denoising problem), convolution (deblurring problem), and random masking (inpainting problem).

We will need to be able to compute for any images x:

- the application of $\mathbf{H}$ to x: $x \mapsto \mathbf{H}x$,
- the application of its adjoint: $x \mapsto \mathbf{H}^*x$,
- the application of its gram matrix: $x \mapsto \mathbf{H}^*\mathbf{H}x$,
- the resolvent of its gram matrix: $x \mapsto (\mathrm{Id} + \tau\mathbf{H}^*\mathbf{H})^{-1}x$.

A linear operator will be represented by a Python object as an instance of a class that inherits from our homemade abstract class `LinearOperator` defined in `imagetools/provided.py`. Please have a look at the code. Note that `LinearOperator` has a method `norm2` that returns an approximation of the spectral norm of the operator $\|\cdot\|_2$ and `normfro` that returns an approximation of the Frobenius norm $\|\cdot\|_F$. It also has two properties `ishape` and `oshape`, the first one is the shape of the input of the operator, the second is the shape of the output. Any class that inherits from it must implement (at least):

- `__call__(self, x)`
- `adjoint(self, x)`
- `gram(self, x)`
- `gram_resolvent(self, x, tau)`

As an example, we provided `Grad` that reuses functions from the previous assignments to implement each of these methods for the gradient operator. An object can be instantiated as `H = im.Grad((n1, n2, 3))` for the gradient of a RGB image of shape $(n1, n2, 3)$.

1. In `imagetools/projectB.py`, create a class `Identity` that implements the identity operator $x \mapsto x$. An object can be instantiated as `H = im.Identity(shape)`.

2. Create a class `Convolution` that implements the convolution operator $x \mapsto \nu * x$. An object can be instantiated as `Convolution(shape, nu, separable=None)`. As we will manipulate large convolution kernels ν, all operations should be implemented in the Fourier domain. Note that during this project, we will always consider periodical boundary conditions.

   Hint: reuse functions from the assignments.

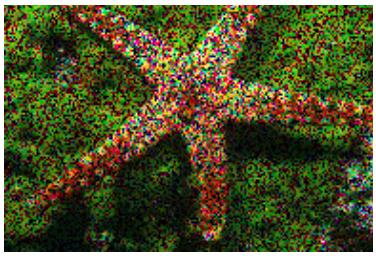
3. Create a class `RandomMasking` that implements the linear operator that sets a proportion p of arbitrary pixels to zeros. An object can be instantiated as `H = im.RandomMasking(shape, p)`.

4. In your notebook, load the image `x0 = starfish`. Create a version y for each of the three operators. For the random masking we will consider $p = .4$. For the convolution we will consider the motion kernel ν. Display the result and check that they are consistent with the following ones.

Identity

Blur

Masking



5. For the three linear operators, check that $\langle \mathbf{H}x, y\rangle = \langle x, \mathbf{H}^* y\rangle$ for any arbitrary arrays `x` and `y` of shape `H.ishape` and `H.oshape` respectively (you can generate `x` and `y` randomly).

6. Check also that $(\mathrm{Id} + \tau \mathbf{H}^*\mathbf{H})^{-1}(x + \tau \mathbf{H}^*\mathbf{H}x) = x$ for any arbitrary image `x` of shape `H.ishape`.

2 Smoothed Total-Variation (25%)

The Total-Variation (TV) aims at reconstructing a piece-wise constant approximation of the image x_0 (refer to Chapter 4). Its discrete version minimizes, for $\tau > 0$, the following energy

$$E(x) = \frac{1}{2}\|y - \mathbf{H}x\|_2^2 + \tau \|\nabla x\|_1 \quad \text{where} \quad \|\nabla x\|_1 = \sum_{i=1}^{n_1}\sum_{j=1}^{n_2}\sum_{k=1}^{2}\sum_{c=1}^{3} |(\nabla x)_{i,j,k,c}| \tag{1}$$

7. Why does TV promote piece-wise constant solutions?

8. As a first step, we aim at solving TV by gradient descent. However, the energy is not differentiable since the absolute value is not differentiable at 0. As an alternative, we will consider a smoothed version by considering the approximation

$$|(\nabla x)_{i,j,k,c}| \approx \sqrt{|\nabla x|^2_{i,j,k,c} + \varepsilon}\ . \tag{2}$$

   for a small $\varepsilon > 0$. Show that in this case the gradient is

$$\nabla E(x) = \mathbf{H}^*(\mathbf{H}x - y) - \tau \,\mathrm{div}\left(\frac{\nabla x}{\sqrt{|\nabla x|^2 + \varepsilon}}\right) \tag{3}$$

   where the fraction is to be understood pointwise.

9. In `imagetools/projectB.py`, create a function

```
def total_variation(y, sig, H=None, m=400, rho=1, return_energy=False):
    ...
    if return_energy:
        return x, e
    else:
        return x
```

that performs m iterations of gradient descent for the smoothed total-variation with $\tau = \rho\sigma$. The argument `sig` is the noise standard deviation σ and `H` the linear operator $\mathbf{H}$ (identity if `None`). If `return_energy=True`, your function should also return a list `e` of size m of the energy $E(x^k)$ obtained at each iteration. Recall that gradient descent is

$$x^{k+1} = x^k - \gamma \nabla E(x^k) \quad \text{for} \quad 0 < \gamma < \frac{2}{L} \tag{4}$$

where $L = \sup_x \|\nabla^2 E(x)\|_2$ when E is twice differentiable. We will admit that $L = \|\mathbf{H}\|_2^2 + \frac{\tau}{\sqrt{\varepsilon}}\|\Delta\|_2$. We will consider $x^0 = \mathbf{H}^* y$, $\gamma = \frac{1}{L}$ and $\varepsilon = 10^{-3}\sigma^2$.

10. Add an optional argument `scheme`

```
def total_variation(y, sig, H=None, m=400, scheme='gd', return_energy=False)
```

and implement the Nesterov acceleration for the case where `scheme='nesterov'`. Nesterov acceleration reads as

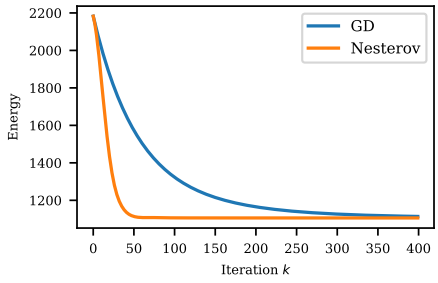
$$x^{k+1} = \tilde{x}^k - \gamma \nabla E(\tilde{x}^k) \tag{5}$$

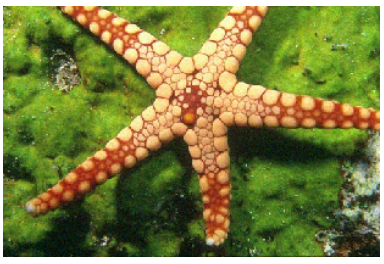
$$\tilde{x}^{k+1} = x^{k+1} + \mu^k (x^{k+1} - x^k) \tag{6}$$

We will consider $x^0 = \tilde{x}^0 = \mathbf{H}^* y$ and choose

$$\mu^k = \frac{t_k - 1}{t_{k+1}}, \quad t_{k+1} = \frac{1 + \sqrt{1 + 4t_k^2}}{2} \quad \text{and} \quad t_0 = 1 \tag{7}$$

11. Create a noisy version y of `x0 = starfish` with noise standard deviation $\sigma = 20/255$. Run your function for $\rho = 1$ and $m = 400$ iterations and compute the energy at each iteration (it should take about 1 minute for each scheme). Repeat for both schemes. Display your results and check they are consistent with the following ones.





12. Is $m = 50$ iterations enough for gradient descent? for gradient descent with Nesterov acceleration?

13. Repeat first with a random masking with $p = .4$ $\sigma = 2/255$, and next with the motion blur and $\sigma = 2/255$. Do you reach the same conclusion?

3 Advanced solvers for Total-Variation (25%)

In this section, we will investigate two other algorithms that solve our restoration problem much faster than with gradient descent even with Nesterov acceleration.

14. Write the function

```
def softthresh(z, t):
```

that for an array z implements pointwise the soft-thresholding defined as:

$$z \mapsto \begin{cases} 0 & \text{if} \quad |z| \leqslant t \\ z - t & \text{if} \quad z > t \\ z + t & \text{otherwise} \end{cases} \tag{8}$$

Do not use loops!

Hint: You can write it in a single line by combining `np.abs`, `np.maximum` and `np.sign`.

15. ADMM (Alternating Direction Method of Multipliers) is another optimization algorithm that can be used to solve our total-variation problem. It is a general technique that can be used to solve any problems of the form

$$E(x) = \frac{1}{2}\|y - \mathbf{H}x\|_2^2 + \tau\|\mathbf{\Gamma}x\|_1 \ . \tag{9}$$

In this case, ADMM reads as

$$\begin{aligned} x^{k+1} &= (\mathrm{Id}_n + \gamma\mathbf{H}^*\mathbf{H})^{-1}(\tilde{x}^k + d_x^k + \gamma\mathbf{H}^*y) \\ z^{k+1} &= \mathrm{softthresh}(\tilde{z}^k + d_z^k, \gamma\tau) \\ \tilde{x}^{k+1} &= (\mathrm{Id}_n + \mathbf{\Gamma}^*\mathbf{\Gamma})^{-1}(x^{k+1} - d_x^k + \mathbf{\Gamma}^*(z^{k+1} - d_z^k)) \\ \tilde{z}^{k+1} &= \mathbf{\Gamma}\tilde{x}^{k+1} \\ d_x^{k+1} &= d_x^k - x^{k+1} + \tilde{x}^{k+1} \\ d_z^{k+1} &= d_z^k - z^{k+1} + \tilde{z}^{k+1} \end{aligned}$$

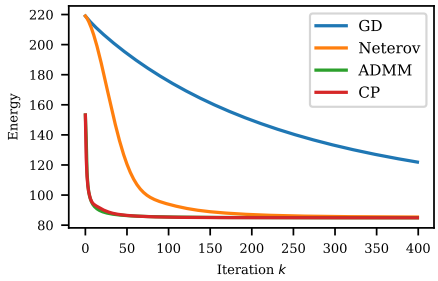
and x^k converges to a solution for any value $\gamma > 0$ and initializations $(\tilde{x}^0, \tilde{z}^0, d_x^0, d_z^0)$. Note that the x variables are images and the z variables are vector fields. Please refer to the class for more details (chapter 6). Modify your function `total_variation` to implement ADMM when `scheme='admm'`. Consider $\mathbf{\Gamma} = \nabla$, $\gamma = 1$, $\tilde{x}^0 = \mathbf{H}^*y$, $\tilde{z}^0 = \nabla\tilde{x}^0$, $d_x^0 = 0$ and $d_z^0 = 0$.

16. An alternative to ADMM, is the Chambolle-Pock algorithm (also known as primal-dual algorithm). It is a general technique that can also be used to solve such a problem. It reads as follows

$$\begin{aligned} \tilde{z}^{k+1} &= z^k + \kappa\mathbf{\Gamma}(v^k) \\ z^{k+1} &= \tilde{z}^{k+1} - \mathrm{softthresh}(\tilde{z}^{k+1}, \tau) \\ \tilde{x}^{k+1} &= x^k - \gamma\mathbf{\Gamma}^*(z^{k+1}) \\ x^{k+1} &= (\mathrm{Id} + \gamma\mathbf{H}^*\mathbf{H})^{-1}(\tilde{x}^{k+1} + \gamma\mathbf{H}^*y) \\ v^{k+1} &= x^{k+1} + \theta(x^{k+1} - x^k) \end{aligned}$$

and x^k converges to a solution for any choice of $\kappa > 0$ and $\gamma > 0$ satisfying $\kappa\gamma\|\mathbf{\Gamma}\|_2^2 < 1$, and any initializations (x^0, z^0, v^0). Note that the x and v variables are images and z is a vector field. Modify your function `total_variation` to implement Chambolle-Pock algorithm when `scheme='cp'`. Consider $\mathbf{\Gamma} = \nabla$, $\gamma = \theta = 1$, $\kappa = 1/\|\mathbf{\Gamma}\|_2^2$, $x^0 = \mathbf{H}^*y$ and $z^0 = v^0 = 0$.

17. Create a blurry version y of `x0 = flowers` with noise standard deviation $\sigma = 2/255$ and motion blur. Run TV on y with ADMM and Chambolle-Pock algorithm with $m = 400$ and $\rho = 1$. Compare the speed of these algorithms with gradient descent and Nesterov acceleration. Check that your results are consistent with the following ones.



18. What are the advantages of Chambolle-Pock algorithm compared to ADMM? (If you do not know, start the next section, the answer should become clear)

4 Total Generalized Variation (25%)

The Total Generalized Variation (TGV) aims at reconstructing a piece-wise affine approximation of the image x_0. A simplified and discrete version of it minimizes, for $\tau > 0$ and $\zeta > 0$, the following energy

$$E(x) = \frac{1}{2}\|y - \mathbf{H}x\|_2^2 + \tau \min_z \ (\|\nabla x - \zeta z\|_1 + \|\mathrm{div}\, z\|_1) \tag{10}$$

where z is a vector field.

19. Show that when $\zeta = 0$, TGV is equivalent to TV.

20. Provide your interpretation on why does TGV promote piece-wise affine solutions?

21. Show that TGV can be rewritten, for $X = \begin{pmatrix} x \\ z \end{pmatrix}$, as

$$E(X) = \frac{1}{2}\|y - \bar{\mathbf{H}}X\|_2^2 + \tau\|\mathbf{\Gamma}X\|_1 \quad \text{with} \quad \bar{\mathbf{H}} = \begin{pmatrix} \mathbf{H} & 0 \end{pmatrix} \quad \text{and} \quad \mathbf{\Gamma} = \begin{pmatrix} \nabla & -\zeta\mathrm{Id} \\ 0 & \mathrm{div} \end{pmatrix}. \tag{11}$$

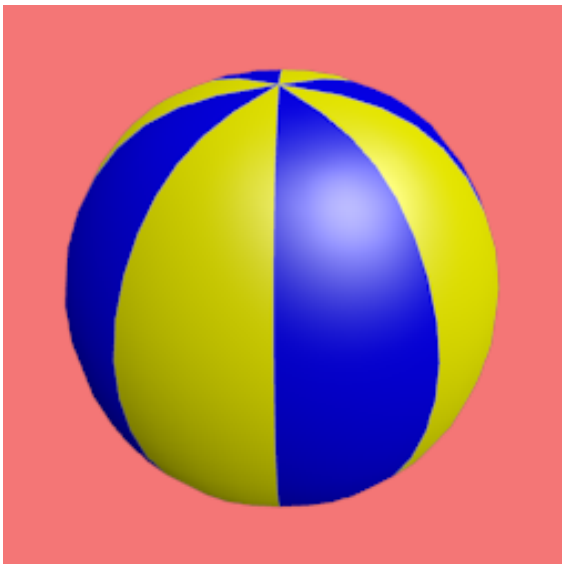
22. In `imagetools/projectB.py`, create the function

```
def tgv(y, sig, H=None, zeta=.1, rho=1, m=400, return_energy=False)
```

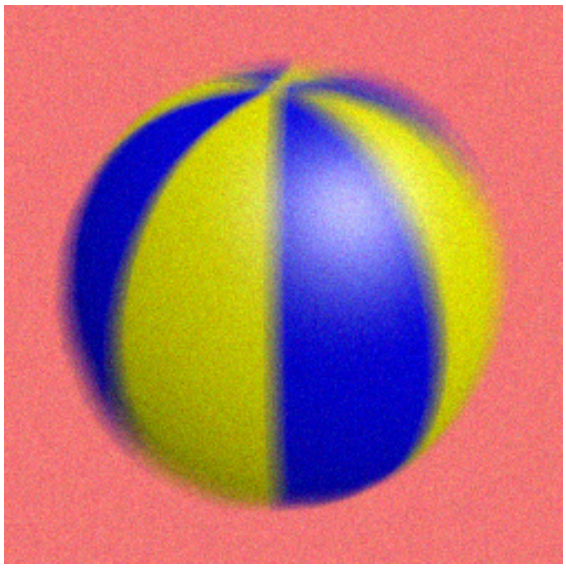
that solves TGV for an arbitrary operator $\mathbf{H}$ and $\tau = \rho\sigma$. You can choose to implement the algorithm of your choice (but choose wisely!). We will consider $\zeta = .1$.

23. Create a blurry version y of `x0 = ball` with noise standard deviation $\sigma = 10/255$ and motion blur. Compare TV and TGV with default parameters.

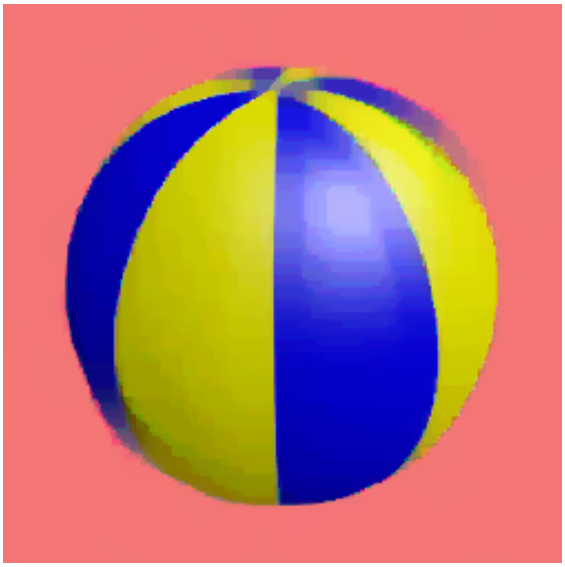
    Results may look like the following ones:



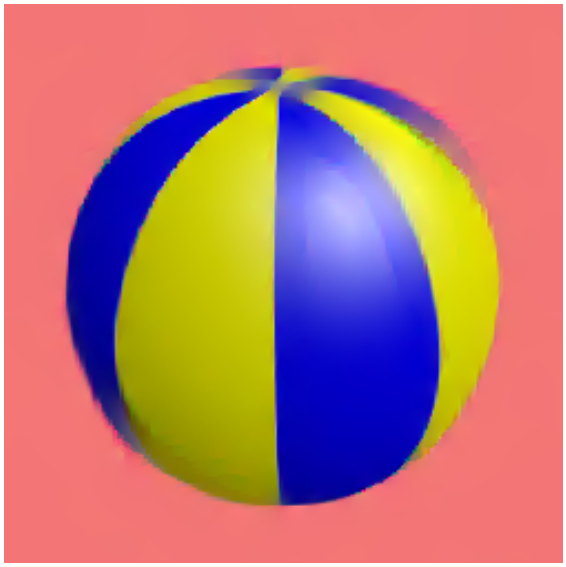
(a) Original



(b) Blurry



(c) Based on TV (26.06 dB)



(d) Based on TGV (26.10 dB)

5 Bonus (+10% max)

- In denoising, for different noise levels and parameters, compare your implementation of TV with the ones of Scikit image: `skimage.restoration.denoise_tv_*`.
- Implement super-resolution.
- We have implemented anisotropic TV (refer to Chapter 4). Implement isotropic TV.
- Implement and discuss further possible improvements.