ECE 285 – Project C
Wavelet based image restoration

Written by Charles Deledalle on June 7, 2019.

You will have to submit a notebook `projectC.ipynb` and the package `imagetools/projectC.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 using wavelet transforms. 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/normandy.png` • `assets/dragonfly.png` • `assets/topgun.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/projectC.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.

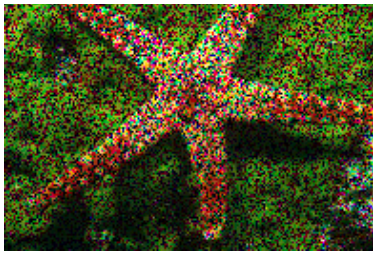
3. Create a class `RandomMasking` that implements the linear operator that sets a proportion p of arbitrary pixels to zeros. An object can be instantiated as `H = im.RandomMasking(shape, p)`.

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 Discrete Wavelet Transform (25%)

7. In `imagetools/projectC.py`, create the function

```
def dwt(x, J, h, g):
    ...
    return z
```

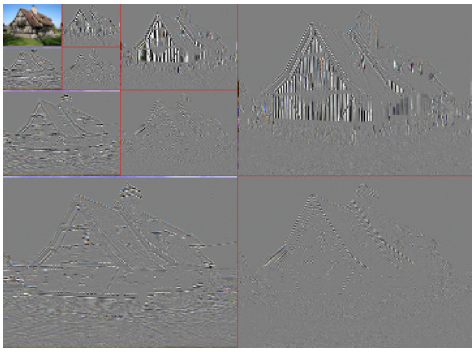
   that implements the 2d Discrete Wavelet Transform (DWT) with J scales, where `h` and `g` are high-pass and low-pass wavelet filters of shape $(n, 1)$ as provided by the homemade function `wavelet` in `imagetools/provided.py`.

   Hint: refer to Chapter 6.

8. We will use Daubechies-2 wavelets and $J = 3$ scales. In your notebook, load the image `x0 = normandy` and crop its dimension to the largest dimension compatible with the DWT by using the homemade function `im.dtw_crop`. Apply your function on this image and display its wavelet coefficients by using the homemade function `im.showdwt`. Check that your results are consistent with the following ones:

Image

Its wavelet coefficients



9. Create the function

```
def idwt(z, J, h, g)
```

implementing the 2d Inverse DWT.

10. In your notebook, check that `idwt` is the inverse of `dwt`. Check for both, the left and right invert: `dwt` ∘ `idwt` and `idwt` ∘ `dwt`. Check also whether it is the adjoint of `dwt`. How is a linear operation called when its inverse is its own adjoint?

11. Create the class `DWT` as a wrapper for the previous functions. It should inherit from `LinearOperator` and be instantiated as

```
W = im.DWT(shape, J, name='db2')
```

and also implements two methods

- `invert(self, x)` such that `x = W.invert(W(x)) = W(W.invert(x))`,
- `power(self)` that returns the results of our homemade function `dwt_power(n1, n2, J)`.

12. Write the function

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

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

$$z \mapsto \begin{cases} 0 & \text{if} \quad |z| \leqslant t \\ z - t & \text{if} \quad z > t \\ z + t & \text{otherwise} \end{cases} \tag{1}$$

Do not use loops!

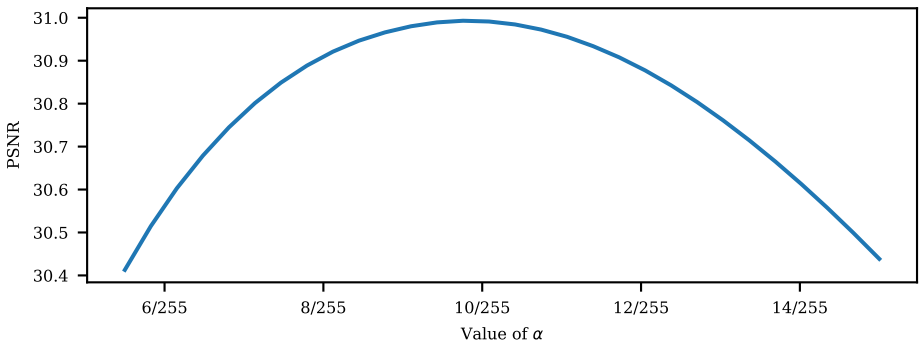
Hint: You can write it in a single line by combining `np.abs`, `np.maximum` and `np.sign`.

13. Write the function

```
def softthresh_denoise(y, sig, W, alpha=10/255)
```

that removes noise by performing soft-thresholding on the coefficients $W(y)_i$ by using the threshold $\tau_i = \frac{\sqrt{2}\sigma^2}{\lambda_i}$ where λ_i is the expected standard deviation of the corresponding clean wavelet coefficient. We will model it as $\lambda_i = \alpha p_i$ where $p_i = 2^{(j-1)}$ as provided by `W.power()` where j is the scale of the pixel index i. The scalar α is the last optional argument.

14. In your notebook, create a noisy version y of `x0 = dragonfly` (cropped to compatible dimension) with noise standard deviation $\sigma = 20/255$. Run your function with different values of α and check that your results are consistent with the following ones:



3 Undecimated Discrete Wavelet Transform (25%)

15. In `imagetools/projectC.py`, create the function

```
def udwt(x, J, h, g):
    ...
    return z
```

that implements the 2d Undecimated Discrete Wavelet Transform (UDWT) with J scales.

Hint: refer again to Chapter 6.

16. In your notebook, test again your function on `x0 = dragonfly` and display its undecimated wavelet coefficients by using `im.show` (you can make one subplot per channel). Again use Daubechies-2 wavelets and $J = 3$. Describe the result.

17. Create the function

```
def iudwt(z, J, h, g)
```

implementing the 2d Inverse UDWT.

18. In your notebook, check for both, the left and right invert: `iudwt` $\circ$ `udwt` and `udwt` $\circ$ `iudwt`. Is the UDWT invertible? Check also whether `iudwt` is the adjoint of `udwt`.

19. Implement `udwt_create_fb`, `fb_apply` and `fb_adjoint` as introduced in Chapter 6.

20. Create the class `UDWT` as a wrapper for the previous functions. It should inherit from `LinearOperator` and be instantiated as

```
W = im.UDWT(shape, J, name='db2', using_fb=True)
```

where the flag `using_fb` allows your wrapper to switch between the recursive and the filter bank implementation. As for DWT, implement also the two methods

- `invert(self, x)` corresponding to its pseudo-inverse,

- `power(self)` that returns the results of our homemade function `udwt_power(J)`.

21. Run `softthresh_denoise` on the noisy image y with W the undecimated wavelet transform. Compare the results computation time of both, the recursive and the filter bank implementation. Compare that you have a gain of about 1dB compared to the results with the DWT.

4 Wavelet based image restoration (25%)

We now aim at reconstructing an approximation of the image x_0 by sparse analysis regularization of wavelet coefficients. This consists of minimizing the following energy

$$E(x) = \frac{1}{2}\|y - \mathbf{H}x\|_2^2 + \tau\|\Lambda^{-1/2}\mathbf{W}x\|_1 \quad \text{where} \quad \Lambda^{+1/2} = \mathrm{diag}(\lambda_1, \ldots, \lambda_n)\ . \tag{2}$$

where we will choose $\tau = \sqrt{2\sqrt{\frac{m}{n}}\sigma^2}$ where m is the dimension of y and n the dimension of $\mathbf{W}x$.

22. ADMM (Alternating Direction Method of Multipliers) is an optimization algorithm that can be used to solve our optimization problem as

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

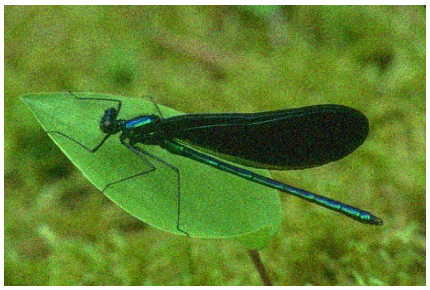
where $\gamma > 0$. The iterates x^k converge to a solution for any value $\gamma > 0$ and initializations $(\tilde{x}^0, \tilde{z}^0, d_x^0, d_z^0)$. Note that the x variables are images and the z variables are wavelet coefficients. Please refer to the class for more details (chapter 6). In `imagetools/projectC.py`, create the function

```
def sparse_analysis_regularization(y, sig, W, H=None, m=40, alpha=10/255,
                                   gamma=1, return_energy=False):
    ...
    if return_energy:
        return x, e
    else:
        return x
```

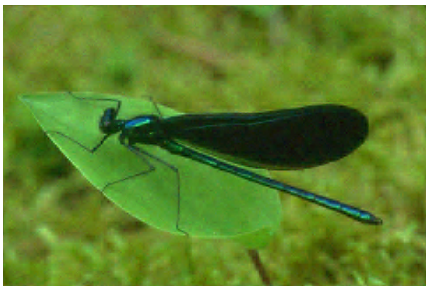
that performs m iterations of ADMM. The argument `sig` is the noise standard deviation σ and `H` the linear operator $\mathbf{H}$ (identity if `None`). If `return_energy=True`, your function should also return a list `e` of size m of the energy $E(x^k)$ obtained at each iteration. Consider the initialization $\tilde{x}^0 = \mathbf{H}^*y$, $\tilde{z}^0 = \mathbf{W}\tilde{x}^0$, $d_x^0 = 0$ and $d_z^0 = 0$.

23. In your notebook, run `sparse_analysis_regularization` on the noisy image y of `x0 = dragonfly` with default parameters, first with $\mathbf{W}$ the DWT, and next with the UDWT. Interpret the results. About how many iterations are required before the energy plateaus? Check that your results are consistent with the following ones

Noisy (22.11dB)



Softtresh DWT (31.01dB)



Softtresh UDWT (32.43dB)

Sparse Analysis Regularization UDWT (32.85dB)

24. In your notebook, load the image `x0 = topgun`, crop it to a compatible dimension, and create a blurry version `y` by defining $\mathbf{H}$ as the motion blur kernel, and add a Gaussian noise of standard deviation $\sigma = 2/255$. Apply your function on y (choose default parameters but with `gamma=100`). Check the evolution of the loss.

    Results must look like the following ones:

Blurry (21.03dB)

SAR with DWT (27.40dB)

SAR with UDWT (29.89dB)

25. Repeat the experiment but with a random masking of 40%.

5 Bonus (+10% max)

• In denoising, for different noise levels and parameters, compare your wavelet implementation with the one of Scikit image: `skimage.restoration.denoise_wavelet`.

• Implement super-resolution.

• Study the influence of the wavelet family and the number of scales.

• Implement and compare with sparse synthesis regularization.

• Implement and discuss further possible improvements.