ECE 285 – Project A
Anisotropic Diffusion Regularization

Written by Charles Deledalle on May 31, 2019.

You will have to submit a notebook `projectA.ipynb` and the package `imagetools/projectA.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 by anisotropic diffusion based regularizations. 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/fish.png`
- `assets/mushroom.png`
- `assets/moose.png`
- `assets/starfish.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)`

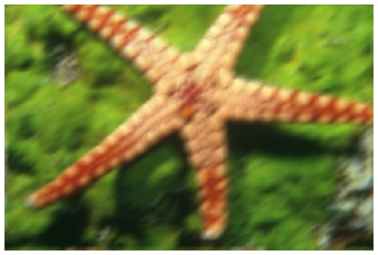
As an example, we provided `Grad` that reuses functions from the previous assignments to implement each of these methods for the gradient operator. An object can be instantiated as `H = im.Grad((n1, n2, 3))` for the gradient of a RGB image of shape $(n1, n2, 3)$.

1. In `imagetools/projectA.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.

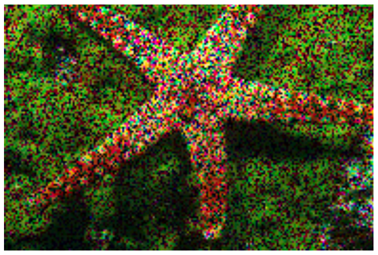
   Hint: reuse functions from the assignments.
3. Create a class `RandomMasking` that implements the linear operator that sets a proportion p of arbitrary pixels to zeros. An object can be instantiated as `H = im.RandomMasking(shape, p)`.
4. In your notebook, load the image `x0 = starfish`. Create a version 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 Anisotropic diffusion (25%)

7. Create in `imagetools/projectA.py`, a function

```python
def heat_diffusion(y, m, gamma, scheme='continuous')
```

   that produces the result of the Heat equation in complexity $O(n \log n)$ where m is the number of discrete time steps and γ the step size (refer to Chapter 4). Note that during this project, we will always consider periodical boundary conditions. You must implement the three schemes: `explicit`, `implicit` and `continuous`.

   Hint: reuse functions from the assignments.
8. In your notebook, load the image `x0 = fish` and create a noisy version `y` with Gaussian noise of standard deviation $\sigma = 10/255$. Then, run your function on `y` for `m = 100` and `gamma = 1/8`.
9. Test different schemes and discuss their convergence depending on the value of γ.
10. Create in `imagetools/projectA.py`, a function

```python
def norm2(v, keepdims=True)
```

that returns the square of the ℓ_2 norm (amplitude) at each location of the vector field v: $a_{i,j} = \|v_{i,j,:}\|_2^2$. If the image is RGB, it must return the sum of a for the three channels. The result must have shape (n_1, n_2) if `keepdims=False`, otherwise $(n_1, n_2, 1)$ for grayscale images and $(n_1, n_2, 1, 1)$ for color images.

11. Create in `imagetools/projectA.py`, a function

```python
def anisotropic_step(x, z, gamma, g, nusig, return_conductivity=False)
    ...
    if return_conductivity:
        return x, alpha
    else:
        return x
```

that implements one step of the explicit scheme for the regularized anisotropic diffusion defined by

$$\alpha^k = g(\|\nabla(\mathcal{G}_\sigma * x^k)\|_2^2)$$
$$v^k = \alpha^k \nabla x^k$$
$$x^{k+1} = x^k + \gamma \operatorname{div} v^k$$

where γ is the step size, $g : \mathbb{R} \to \mathbb{R}$ is a `Python` lambda function, and `nusig` is the Gaussian spatial kernel (separable). Note that α^k is a 2d array and v^k is a 2d vector field such that

$$\alpha^k_{i,j} = g(\|(\nabla(\mathcal{G}_\sigma * x^k))_{i,j}\|_2^2) \quad \text{and} \quad v^k_{i,j,:} = \alpha^k_{i,j}(\nabla x^k)_{i,j,:} \ .$$

Hint: refer to the class and reuse functions from the assignments.

12. Create in `imagetools/projectA.py`, a function

```python
def anisotropic_diffusion(y, m, gamma, g=None, return_conductivity=False)
```

that runs m iterations of the explicit scheme for the regularized anisotropic diffusion with Gaussian smoothing of bandwidth $\sigma = .2$ (limited to a support $[-1, 1]^2$), where y is the input image, $g(u) = \frac{10}{10+255^2 u/\sqrt{C}}$ if `g = None`, where C is the number of channels (1 for grayscale images, 3 for RGB).

Note: Don't store all iterates x^1, x^2, . . . Instead overwrite the results of iteration k at iteration $k+1$.

13. In your notebook, run your function on `y` for `m = 100` and `gamma = 1/8` with `return_conductivity=True`. Check that your results are consistent with the ones given below.

14. Compare `anisotropic_diffusion` with `heat_diffusion`. Conclude.

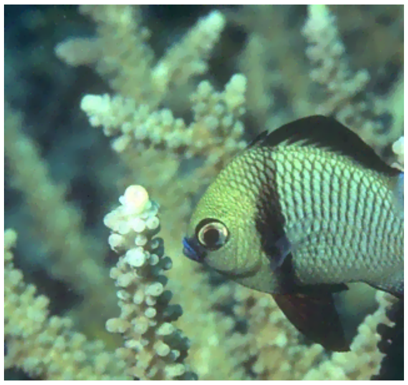
15. Add an optional argument `scheme`

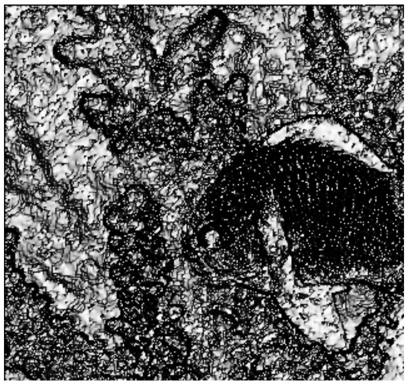
```python
def anisotropic_diffusion(y, m, gamma, g=None, return_conductivity=False,
                          scheme='explicit')
```

and implement for `scheme='implicit'` the implicit scheme as seen in class using the provided homemade function `im.cgs`. In this case, the option `return_conductivity=True` must have no effect.

16. Run the implicit scheme with $m = 5$ and $\gamma = 20/8$. Compare with the explicit scheme and conclude.

(a) Noisy image


(b) RAD


(c) Conductivity

3 Truly anisotropic diffusion (25%)

We are going to implement the explicit scheme for the truly anisotropic diffusion given by

$$\begin{aligned} x^0 &= y \\ M^k &= (\nabla \mathcal{G}_\sigma * x^k)(\nabla \mathcal{G}_\sigma * x^k)^T \\ M^k_{\text{conv}} &= \mathcal{G}_\rho * M^k \\ T^k &= h[M^k_{\text{conv}}] \\ v^k &= T^k \times \nabla x^k \\ x^{k+1} &= x^k + \gamma \operatorname{div} v^k \end{aligned}$$

for $\rho = 0.5$ (Gaussian limited to a support $[-1, 1]^2$), where M^k is a matrix field of 2×2 tensors (size: $n_1 \times n_2 \times 2 \times 2$), $(M^k_{\text{conv}})_{:,:,p,q} = \mathcal{G}_\rho * M^k_{:,:,p,q}$, $h[A]$ is a matrix spectral function that applies g to the eigenvalues of A, and $T^k \times \nabla x^k$ is a vector field obtained as the punctual matrix vector product $(T^k \times \nabla x^k)_{i,j,:,:} = T^k_{i,j,:,:} \times (\nabla x^k)_{i,j,:}$.

17. Create in `imagetools/projectA.py`, a function

```python
def tensorize(v, nurho):
    n1, n2, p = v.shape[:3]
    M = np.zeros((n1, n2, p, p))
    for k in range(p):
        for l in range(p):
            # complete
    return M
```

that takes a p dimensional vector field $n_1 \times n_2 \times p$ (or $n_1 \times n_2 \times p \times 3$ for RGB images) and returns a tensor field $M = \mathcal{G}_\rho * (vv^T)$ where `nurho` is the Gaussian spatial kernel (separable). For RGB images, vv^T must be summed over the 3 color channels.

18. Create in `imagetools/projectA.py`, a function

```python
def matrix_spectral_func(M, g):
    ...
    return T
```

that return the tensor field T (of size $n_1 \times n_2 \times p \times p$) after applying `g` to the eigenvalues of each $p \times p$ tensor of the tensor field M (of size $n_1 \times n_2 \times p \times p$).

Hint: use `numpy.linealg.svd` and read well its documentation. Do not use loops!

19. Create in `imagetools/projectA.py`, a function

```python
def truly_anisotropic_step(x, z, gamma, g, nusig, nurho,
                           return_conductivity=False)
    ...
    if return_conductivity:
        return x, T
    else:
        return x
```

that implements one step of the truly anisotropic diffusion.

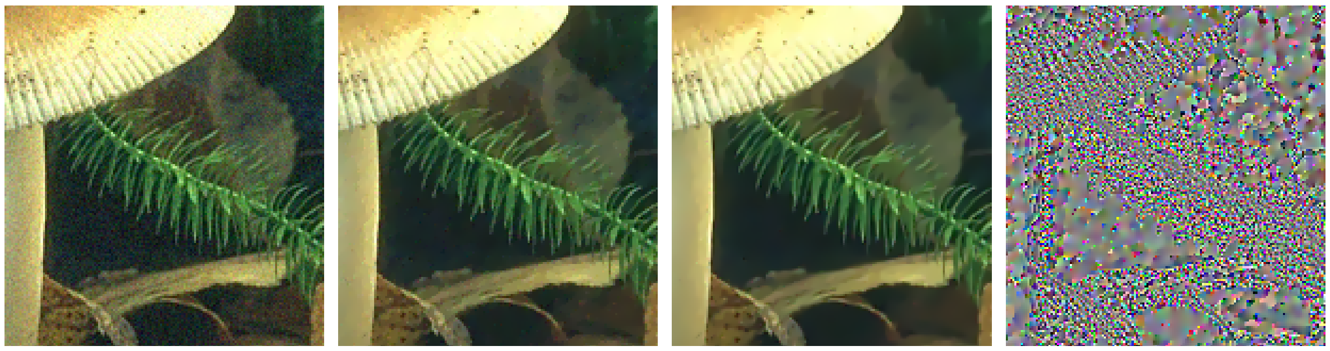
Hint: use the function from the two last questions and `numpy.matmul`.

20. Add an optional argument `variant`

```python
def anisotropic_diffusion(y, m, gamma, g=None, scheme='explicit', variant=None,
                          return_conductivity=False, variant=None)
```

and implement for `variant='truly'` the truly anisotropic diffusion, for both the explicit and implicit schemes.

21. In your notebook, load the image `x0 = mushroom` and create a noisy version `y` with Gaussian noise of standard deviation $\sigma = 10/255$. Then, run the truly anisotropic diffusion on `y` with explicit scheme, `m = 100` and `gamma = 1/8`. Display the results, compare PSNRs and check that they are consistent with the following ones.


(a) Noisy (b) AD (15s / 32.5dB) (c) TAD (52s / 33.2dB) (d) Difference

22. Rerun with implicit scheme, `m = 5` and `gamma = 20/8`, and check that your results are of similar quality but the approach is about 3 times faster.

4 Regularization for image restoration (25%)

23. In `imagetools/projectA.py`, create the function

```python
def anisotropic_diffusion_regularization(y, m, sig, H=None, g=None, ...)
```

that solves our restoration problem

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

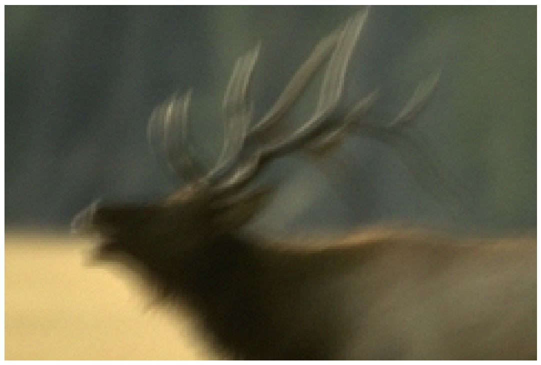
based on the tools developed in the previous section. The argument `sig` is the noise standard deviation σ, `H` the linear operator $\mathbf{H}$ (identity if `None`) and $g(u) = \frac{\sigma^2}{\sigma^2 + u/\sqrt{C}}$ if `None`. You may come up with the approach of your choice (you can refer to Chapter 4 for inspiration).

24. Discuss the mathematical foundations of your approach.

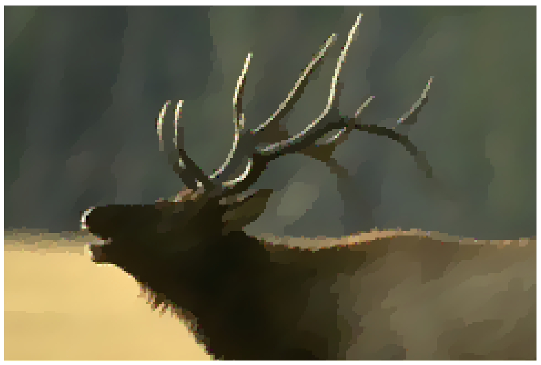
25. In your notebook, load the image `x0 = moose` and create a blurry version `y` by defining $\mathbf{H}$ as the motion blur kernel, and add a Gaussian noise of standard deviation $\sigma = 2/255$. Apply your function on y.

    Results may look like the following ones:

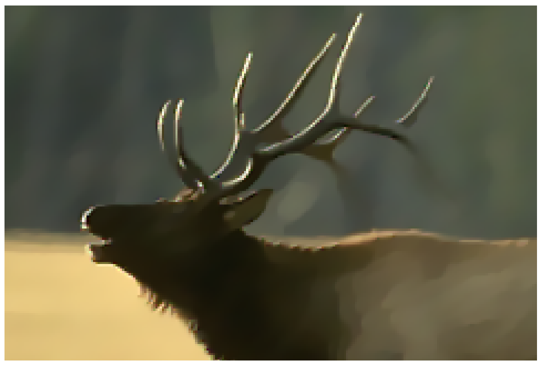
(a) Original



(b) Blurry



(c) Based on AD (32.8dB, 73s)



(d) Based on TAD (33.9dB, 230s)

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

5 Bonus (+10% max)

- Use the exact formula of the eigen decomposition of 2×2 matrices to make `matrix_spec_func` faster.
- Implement super-resolution.
- Implement and discuss further possible improvements.