scc0251/5830— Prof. Moacir Ponti

Teaching Assistant: Aline Becher

Assignment 3 : image restoration

Try to code the assignment by yourself. Plagiarism is not tolerated.

In this assignment you have to implement methods for image restoration. Read the instructions for each step. Use `python` with the `numpy` and `scipy` libraries.

Your program must allow the user to provide parameters in order to generate images by the following steps

1. **Parameter input**:
   - filename for the reference image f with size $n \times m$,
   - filename for the degraded image g with size $n \times m$,
   - type of filter to perform image restoration $F = 1$ (denoising) or 2 (deblurring),
   - parameter $gamma \in [0, 1]$,
   - size of the denoising filter kernel (if type 1) or size of degradation function (if type 2), $k \in [3, 5, 7, 9, 11]$,

   Afterwards, if the type of filter is 1 (denoising), get the following input:
   - mode for denoising $mode$ = `"average"` or `"robust"`,

   Otherwise, if the type of filter is 2 (deblurring), then
   - sigma for the degradation function $\sigma > 0$.
2. **Restore image** g with filter F producing the restored image $\hat{f}$.
3. **Compute** and print the RMSE comparing $\hat{f}$ with the reference image f,

Filters for image restoration

A `python` function must be coded for each one of the following image restoration methods:

1 – Adaptive denoising : this filter performs the a filter but weighing the strenght of the filter centred in each pixel by checking the dispersion of the pixels in comparison with the expected dispersion value. For this method, use only functions available at `numpy` library.

1. $\mathbf{x} = (x, y)$ is the pixel coordinate,
2. $g(\mathbf{x})$: the value of noisy image at $\mathbf{x}$,

3. disp_η : estimated dispersion measure for image noise,
4. centr_L : centrality measure for pixels in a neighbourhood $k \times k$ centred at $\mathbf{x}$
5. disp_L : dispersion measure of pixels in a neighbourhood $k \times k$ centred at $\mathbf{x}$

$$\hat{f}(\mathbf{x}) = g(\mathbf{x}) - \gamma \frac{\text{disp}_\eta}{\text{disp}_L} [g(\mathbf{x}) - \text{centr}_L]$$

There are *two modes for denoising*:

1. `"average"`: uses the arithmetic mean as centrality measure, and the standard deviation as the dispersion measure.
2. `"robust"`: uses the median as centrality measure, and the interquartile range as the dispersion measure.

In order to estimate disp_η, calculate the dispersion measure over the rectangular region related to the coordinates from 0 to 1/6 of each lateral size of the image, i.e. for the coordinates: $x = (0, \lfloor n/6 \rfloor - 1)$, $y = (0, \lfloor m/6 \rfloor - 1)$.
Border computation: since a neighbourhood is needed to compute the values, it may not be able to compute those for some border values: pixels for those coordinates must be copied from the observed image g. No padding is needed.

Error checking:

- If the dispersion measure computed for some image is $\text{disp}_\eta = 0$, then manually set it to $\text{disp}_\eta = 1$.
- If during the denoising step, any dispersion measure is $\text{disp}_L = 0$, then set it so that $\text{disp}_L = \text{disp}_\eta$.

After denoising, normalize the image so that the output image $\hat{f}$ has the same maximum value as the degraded input image g.

2 – Constrained Least Squares Filtering : this filter tries to approximate a solution via a minimization of the mean squared error, using *a priori* assumptions, in this case a Laplacian regularization. For this method, you may use functions available at `numpy` and `scipy` libraries. The `fftpack` is specially useful for computing the Fast Fourier Transforms (and the inverse) for the images and degradation functions.

The Constrained Least Squares filter is applied in the Fourier domain. Considering $G(\mathbf{u})$ to be the Fourier transform of the degraded image, and $H(\mathbf{u})$ the Fourier transform of the degradation function (the point spread function), then the restored image can be computed as:

$$\hat{F}(\mathbf{u}) = \left[\frac{H^*(\mathbf{u})}{|H(\mathbf{u})|^2 + \gamma |P(\mathbf{u})|^2} \right] \times G(\mathbf{u}),$$

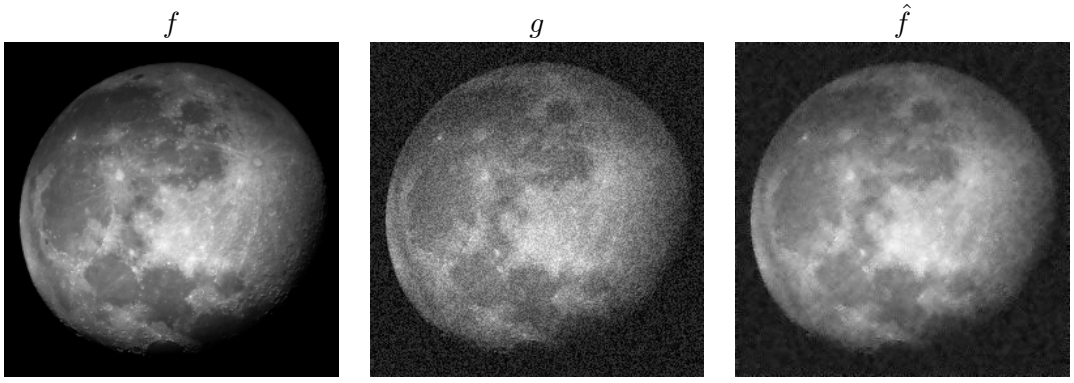
where $P(\mathbf{u})$ is the Fourier transform of a Laplacian operator:

$$p(\mathbf{x}) = \begin{bmatrix} 0 & -1 & 0 \\ -1 & 4 & -1 \\ 0 & -1 & 0 \end{bmatrix}$$

γ controls the influence of the regularization.

We are going to assume that the degradation function is a Gaussian filter with size k and standard deviation σ, given as input. Before deblurring the image, you have to create the function using the following `python` code (you can copy this source code and use in your program, but we recommend you take a time to understand it):

```
1  def gaussian_filter(k=3, sigma=1.0):
2      arx = np.arange((-k // 2) + 1.0, (k // 2) + 1.0)
3      x, y = np.meshgrid(arx, arx)
4      filt = np.exp( -(1/2)*(np.square(x) + np.square(y))/np.square(sigma) )
5      return filt/np.sum(filt)
```



$F = 1$, $\gamma = 0.8$, $k = 5$, *mode* ="robust"

$F = 2$, $\gamma = 0.00005$, $k = 5$, $\sigma = 1.0$

Figura 1: Examples of reference, degraded and restored images for the two image restoration methods

Comparing with reference

Your program must compare the generated image with a reference image r. This comparison must use the root mean squared error (RMSE). Print this error in the screen, rounding to 3 decimal places.

$$RMSE = \frac{1}{n \cdot m}\sqrt{\sum_i \sum_j (g(i,j) - R(i,j))^2}$$

Both the degraded and the reference images are stored in form of images (files PNG or JPG). You should load it as in the example below:

```
import numpy as np
import imageio
filename = str(input()).rstrip()
R = imageio.imread(filename)
```

Input/output examples

Input example 1: reference image in the file `case1.npy`, degraded image in the file `moon.jpg`, denoising adaptive with robust statistics, $F = 1$, $\gamma = 0.6$, with kernel size $k = 3$ and *mode*=`"robust"`:

```
moon.jpg
case1.png
1
0.6
3
robust
```

Input example 2: reference image in the file `case1.npy`, degraded image in the file `polygons.png`, deblurring using CLS, $F = 2$, $\gamma = 0.85$, size of the degradation function $k = 5$ and its standard deviation $\sigma = 1.0$:

```
polygons.png
case5.png
2
0.005
5
1.0
```

Output example: only the RMSE value in format `float` with 3 decimal places

Example 1 (high RMSE, indicating the generate image is too different from the reference):

```
203.786
```

Example 2 (lower RMSE, indicating a similar image and a correct result):

```
72.130
```

OBS: there will be a tolerance of around 20% for the target RMSE.

Submission

Submit your source code using the Run.Codes (only the `.py` file)

1. **Comment your code**. Use a header with name, USP number, course code, year/semestre and the title of the assignment. A penalty on the grading will be applied if your code is missing the header and comments.
2. **Organize your code in programming functions**. Use one function per restoration method, and auxiliary functions if needed. All documented/commented.