ECE 285 – Assignment #4

Writing part

---

1 Exercise 1 (Statistics of linear and moving average filters)

Let Y be a random vector in $\mathbb{R}^n$, with expectation $\mathbb{E}[Y] = x$ and covariance matrix $\mathrm{Var}[Y] = \sigma^2 \mathrm{Id}_n$. Let $H \in \mathbb{R}^{n \times n}$ be an arbitrary square matrix.

1. Show that $\mathbb{E}[HY] = Hx$.
2. Show that $\mathrm{Var}[HY] = \sigma^2 H H^T$.

Consider now that H is a circulant matrix, i.e., there exists $\nu \in \mathbb{R}^n$ such that

$$H = \begin{pmatrix} \nu_0 & \nu_{n-1} & \nu_{n-2} & \dots & \nu_2 & \nu_1 \\ \nu_1 & \nu_0 & \nu_{n-1} & \nu_{n-2} & \dots & \nu_2 \\ & & \ddots & & & \\ & & & \ddots & & \\ & & & & \ddots & \\ \nu_{n-1} & \nu_{n-2} & \dots & \nu_2 & \nu_1 & \nu_0 \end{pmatrix}$$

and extend ν and Y with periodical boundary conditions: $\nu_{i+n} = \nu_i$ and $Y_{i+n} = Y_i$.

3. Show that $\mathbb{E}[\nu * Y] = \nu * x$.
4. Show that $\mathrm{Cov}((\nu * Y)_i, (\nu * Y)_j) = \sigma^2 (r_{\nu\nu})_{i-j}$ where $r_{\nu\nu} = \nu \star \nu$.

   NB: In signal-processing, $(r_{\nu\nu})_\tau$ is called the auto-correlation of ν at lag τ.
5. Deduce that $\mathrm{Var}[(\nu * Y)_i] = \sigma^2 \|\nu\|_2^2$ for all $0 \leqslant i \leqslant n-1$.
6. Assume that $\nu_k \geqslant 0$ and $\sum_{k=0}^{n-1} \nu_k = 1$. Show that $\|\nu\|_2^2 \leqslant 1$. Conclude.
7. Is the residual noise after convolution: white, stationary, colored, signal and/or space dependent?

Practical part

---

2 Exercise 2 (Order-statistic filtering)

Order-statistic filters (OSF) are local filters that are only based on the ranking of pixel values inside a sliding window.

1. Create in `imstack.m`, a function

```
function xstack = imstack(x, s1, s2, boundary)
```

that creates a stack $x^{(\text{stack})}$ of size $n_1 \times n_2 \times s$ with $s = (2s_1+1)(2s_2+1)$ from the $n_1 \times n_2$ image x, such that `xstack(i,j,:)` contains all the values of `x` in the neighborhood $(-s_1, s_1) \times (-s_2, s_2)$. This function should take into account the four possible boundary conditions.

Hint: use `imshift` and only two loops for $-s_1 \leqslant k \leqslant s_1$ and $-s_2 \leqslant l \leqslant s_2$.

2. Create in `imosf.m`, a function

```
function xosf = imosf(x, type, s1, s2, boundary)
```

that implements order-statistic filters. `imosf` should first call `imstack`, next sort the entries of the stack with respect to the third dimension, and create the suitable output `xosf` according to the string `type` as follows

- `'median'`: select the median value,
- `'erode'`: select the min value,
- `'dilate'`: select the max value,
- `'trimmed'`: take the mean after excluding at least 25% of the extreme values on each side.

3. Create in `imopening.m` (resp., `imclosing.m`), the function that performs the opening (resp., closing) by the means of OSF filters.

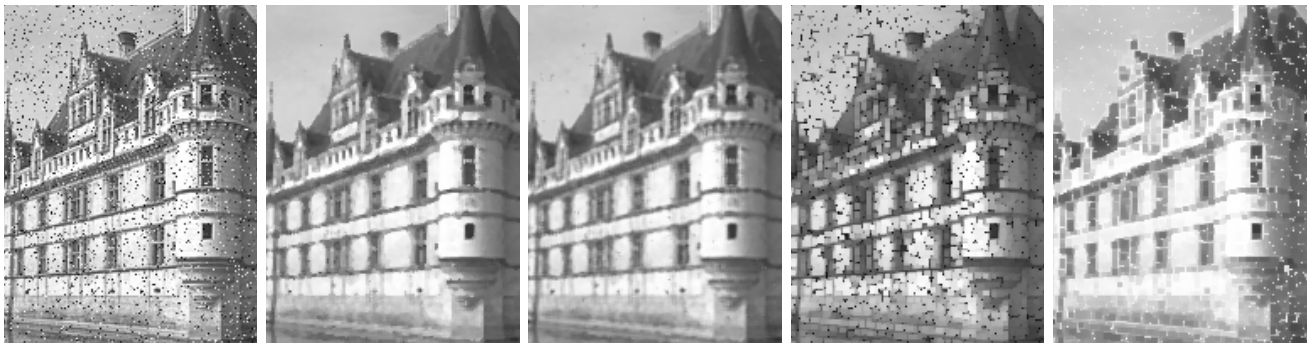
4. Download `assignment4.zip` and extract the data:

- `assignment4/castle.png`

Write a script `test_imosf.m` that loads the image `x = castle` and produces a corrupted version `y` with 10% of impulse noise as

```
p = 0.1;
L = 255;
y = x + (rand(n1, n2) < p) .* (L * rand(n1, n2) - x);
```

Apply your OSF filters and zoom on the results to check that your results are consistent with the following ones:



(a) noisy (b) median (c) trimmed mean (d) closing (e) opening

Figure 1: Results of OSF filters

3 Exercise 3 (Bilateral filter)

The bilateral filter is a denoising algorithm that reads as:

$$x_{i,j}^{(\text{bilateral})} = \frac{1}{Z_{i,j}} \sum_{k=-s_1}^{s_1} \sum_{l=-s_2}^{s_2} \varphi\left((y_{i+k,j+l} - y_{i,j})^2\right) y_{i+k,j+l} \quad \text{with} \quad Z_{i,j} = \sum_{k=-s_1}^{s_1} \sum_{l=-s_2}^{s_2} \varphi\left((y_{i+k,j+l} - y_{i,j})^2\right) .$$

where φ is the so called kernel function and $(-s_1, s_1) \times (-s_2, s_2)$ is the domain of the search window. We will choose $\varphi(\alpha) = \exp\left(-\frac{\max(\alpha - 2\sigma^2, 0)}{16h\sigma^2}\right)$, where σ^2 is the variance of the noise, assumed to be additive white and Gaussian, and $h > 0$ the filtering parameter.

1. Create a script `test_imbilateral.m` that loads the image `x = castle` and adds additive white Gaussian noise of standard deviation $\sigma = 10$ as

```
sig = 10;
y   = x0 + sig * randn(size(x));
```

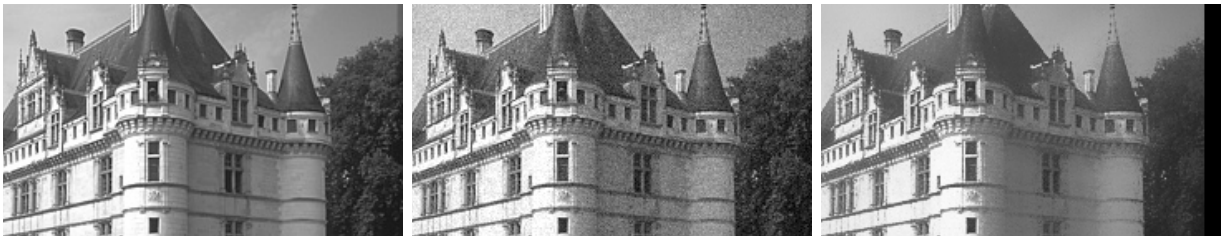
2. Create in `imbilateral_naive.m`, a function

```
function x = imbilateral_naive(y, sig, s1, s2, h)
```

that implements the bilateral filter (except around boundaries) with four loops as

```
x = zeros(n1, n2);
Z = zeros(n1, n2);
for i = (s1+1):(n1-s1)
    for j = (s2+1):(n2-s2)
        for k = -s1:s1
            for l = -s2:s2
                dist2 = (y(i+k,j+l) - y(i,j)).^2;
                % complete
x = x ./ Z;
x(Z == 0) = 0;
```

3. Complete `test_imbilateral.m` to test your function on `y` with $s_1 = s_2 = 10$ and $h = 1$. Zoom on the results to check that your functions are consistent with the following ones:



(a) Original (b) Noisy (c) Naive bilateral (2.6s)

4. Create in `imbilateral.m`, the function

```
function x = imbilateral(y, sig, s1, s2, h, boundary)
```

that implements the bilateral filter including around boundaries. The idea is again to switch the k, l loops with the i, j loops, and then make use of `imshift`. The final code should read with only two loops and deal with boundary conditions.

5. Complete `test_imbilateral.m` to test your new function. Compare the computation times.

6. Increase the noise level, and play with the search window sizes s_1 and s_2 and filtering parameter h.