Fundamentals of Image Processing

Book of Exercises
Part 1: Basics

Master 1 Computer Science - IMAge & DIGIT

Sorbonne Université

Year 2023 – 2024

Contents

1. Basic Operations

Exercise 1 — Image representations

1. What are the two main methods to represent an image in a computer? Assuming that the first pixel on an image of size $N \times M$ has index 0, how to access the pixel of coordinates (i, j) according to each of these representations? What are the coordinates of the 4-neighbors of pixel (i, j)? How to access these neighbors?

2. What is the size in bits of an image of size 256×256 where each pixel value is encoded on one octet (ignore the potential header of the image)? What is the dynamic of this image?

Exercise 2 — Basic matrix operations on images

The following matrices are representations of two images (pixel values in the range $[0, 255]$):

$I =$

100	100	100	100	100	100
100	200	200	100	100	100
100	200	200	100	100	100
100	100	100	100	100	100
100	100	100	100	100	100

$J =$

100	100	100	100	100	100
100	100	200	200	100	100
100	100	200	200	100	100
100	100	100	100	100	100
100	100	100	100	100	100

1. How would you describe intuitively the difference between these two images?

2. Calculate the pixel-wise sum of I and J. What do you observe?

3. Calculate the pixel-wise absolute difference $|I - J|$ of the two images. What do you observe?

4. How is it possible to obtain an image of the same size as I, with only a subpart of I, everything else being black?

Exercise 3 — Histogram

Let us now consider the image represented as:

$I =$

10	2	1	1	10	2	6
7	5	5	4	4	2	3
7	2	0	3	2	1	8
6	3	2	0	3	0	3
5	3	2	7	0	3	5

1. What is the minimal grey level $k_{\min}$ in I? the maximal grey level $k_{\max}$? Derive the dynamics L of I.

2. What is the definition of the histogram? Describe an algorithm to compute the histogram of an image of size $N \times M$ encoded in a vector f. Provide the corresponding `Python` code.

3. Draw the histogram of I.

4. Write an algorithm to be applied on the histogram to invert the grey levels of the image. Can the inverted image be retrieved from the modified histogram? Explain.

5. Write an algorithm to be applied on the image to binarize the image using two values k_1 and k_2, according to a threshold value S ($I_b = k_1$ if $I \leq S$ and $I_b = k_2$ otherwise). Draw the histogram of the binarized image.

6. What is a cumulative histogram? How can it be computed and drawn?

7. We want to change the grey level interval $[k_{\min}, k_{\max}]$ into $[I_1, I_2]$. Assuming that a linear normalization is performed, write the formula providing the new value of a pixel. Provide the corresponding `Python` code.

8. Calculate the stretched histogram to the interval $[2, 30]$, and to the interval $[2, 8]$.

9. Histogram equalization consists in applying a transformation $y = T(x)$ on image values providing a flat histogram. In the continuous case, this is expressed as:

$$\int_{k_{\min}}^{k_{\max}} p(x)dx = \int_{k_{\min}}^{k_{\max}} p'(y)dy \tag{1}$$

with $p'(y) = \frac{1}{k_{max} - k_{\min}}$.

For the image I above, prove that applying Equation (1) in the discrete case amounts to modify a grey level k into $k' = \text{Int}\left(\frac{k_{\max} - k_{\min}}{N \times M} H_c(k)\right)$, with:

- N the number of rows in the image;
- M the number of columns in the image;
- $H_c(k)$ the value at k of the cumulative histogram;
- `Int` the function returning the closest integer.

2. Fourier Transform

Reminder on the continuous Fourier Transform (FT)

Let us recall the definitions of 1D and 2D Fourier Transforms.

1. 1D continuous FT: let x be a continuous signal, i.e. a function such as $t \mapsto x(t), t \in \mathbb{R}$. The Fourier transform of x, denoted $\mathrm{FT}(x)$, is the function X defined as:

$$X(f) = \int_{-\infty}^{+\infty} x(t) e^{-2i\pi ft} dt, f \in \mathbb{R} \tag{2}$$

2. 2D continuous FT: let x be a continuous image, i.e. a function depending on two variables, such as $(t, u) \mapsto x(t, u), t \in \mathbb{R}, u \in \mathbb{R}$. The Fourier transform of x, denoted by $\mathrm{FT}(x)$, is the function X defined as:

$$X(f, g) = \int_{-\infty}^{+\infty} \int_{-\infty}^{+\infty} x(t, u) e^{-2i\pi(ft+gu)} dt du, f \in \mathbb{R}, g \in \mathbb{R} \tag{3}$$

	1D continuous FT		2D continuous FT	
	$x(t)$	$X(f)$	$x(t,u)$	$X(f,g)$
linearity	$x(t) + \lambda y(t)$	$X(f) + \lambda Y(f)$	$x(t,u) + \lambda y(t,u)$	$X(f,g) + \lambda Y(f,g)$
translation	$x(t - t_0)$	$X(f)\ e^{-2i\pi f t_0}$	$x(t - t_0, u - u_0)$	$X(f,g)\ e^{-2i\pi(ft_0+gu_0)}$
contraction	$x(\alpha t)$	$\frac{1}{\lvert\alpha\rvert} X(\frac{f}{\alpha})$	$x(\alpha t, \beta u)$	$\frac{1}{\lvert\alpha\rvert\lvert\beta\rvert} X(\frac{f}{\alpha}, \frac{g}{\beta})$
convolution	$x \star y(t)$	$X(f) \times Y(f)$	$x \star y(t,u)$	$X(f,g) \times Y(f,g)$
product	$x(t) \times y(t)$	$X \star Y(f)$	$x(t,u) \times y(t,u)$	$X \star Y(f,g)$

Table 1: Main properties of continuous FT.

Exercise 4 — Functional properties of continuous FT

Some useful properties of FT are summarized in Table 1. This exercise aims at proving some of them.

1. Prove that computing the 2D FT is equivalent to compute successively two 1D FT, *i.e.* $X(f, g) = \mathrm{FT}\left[u \mapsto Z(f, u)\right](g)$. Explain the function Z and provide its mathematical expression.

2. Prove that, if a 2D function x is separable (can be written as the product of two 1D functions as $x(t, u) = z(t) \times k(u)$), then we have: $X(f, g) = Z(f) \times K(g)$, with $Z(f) = \mathrm{FT}\left[z\right](f)$ and $K(g) = \mathrm{FT}\left[k\right](g)$.

3. Prove the links between convolution (denoted by $\star$) and product:
   $\mathrm{FT}\left[x \star y\right](f) = X(f) \times Y(f)$ and $\mathrm{FT}\left[t \mapsto x(t) \times y(t)\right](f) = X \star Y(f)$. Recall that the convolution of two functions x and y is the function z defined as:

$$z(t) = x \star y(t) = \int_{-\infty}^{+\infty} x(\tau) y(t-\tau) d\tau \tag{4}$$

   For the second relation, it will be useful to introduce the inverse Fourier Transform of a spectrum X: $x(t) = \int_{-\infty}^{+\infty} X(f) e^{2i\pi ft} df$.

4. Prove the rotation property of the 2D FT:
   $\mathrm{FT}\left[(t,u) \mapsto x\left(t\cos\theta + u\sin\theta, -t\sin\theta + u\cos\theta\right)\right](f,g) = X\left(f\cos\theta + g\sin\theta, -f\sin\theta + g\cos\theta\right)$

Exercise 5 — Computing the FT of some usual signals

1. From Equation (2), compute the FT of the function $x(t) = \mathrm{Rect}\left(\frac{t}{T}\right)$, where T is a constant and $\mathrm{Rect}(t)$ is defined as:

$$\mathrm{Rect}(t) = \begin{cases} 1 & \text{if } |t| \leq \frac{1}{2} \\ 0 & \text{otherwise} \end{cases} \tag{6}$$

   Derive the FT of the corresponding 2D function $x(t,u) = \mathrm{Rect}\left(\frac{t}{T}\right) \times \mathrm{Rect}\left(\frac{u}{T}\right)$.

2. Let δ denote the Dirac delta function (see Appendix 1):

$$\delta(t) = \begin{cases} 0 & \text{if } t \neq 0 \\ \infty & \text{otherwise} \end{cases}$$

   Let us recall that $\mathrm{FT}\left[t \mapsto \delta(t-t_0)\right] = e^{-2i\pi ft_0}$ and $\mathrm{FT}\left[t \mapsto e^{2i\pi f_0 t}\right] = \delta(f-f_0)$.

   Derive:

   (a) the FT of $x(t) = \cos\left(2\pi f_0 t\right)$ and $x(t) = \sin\left(2\pi f_0 t\right)$;

   (b) the FT of a 2D Dirac delta function $\delta(t-t_0, u-u_0)$;

   (c) the FT of $s_\theta(t,u) = A\cos\left[2\pi f_o\left(t\cos(\theta) + u\sin(\theta)\right)\right]$.

3. Digitization

Reminder on digital signals and images

Signals and images that are processed and analyzed computationally are **digital**. This impacts their frequential analysis in three main ways:

1. The analysis is performed on a limited temporal or spatial domain: how are the FT of a continuous signal with infinite band and the FT of a band-limited continuous signal related? (see Exercise 7).
2. Sampling of a continuous signal, *i.e.* choice of a countable set of values: what is the FT of the sampled signal? Is some information lost? in which cases? (see Exercise 6).
3. Quantization of the signal (see lecture) & Discrete Fourier Transform (DFT) (see Exercise 8) .

Exercise 6 — Sampling a continuous signal

The ideal sampling consists in selecting in the continuous signal a set of regularly spaced values, forming a discrete (or digital) signal $x_s(t)$.
This writes (see Appendix 2):

$$x_s(t) = \sum_{k=-\infty}^{+\infty} x(t)\delta(t - kT_s) = x(t)\text{⨿}_{T_s}(t) \tag{7}$$

where T_s is the sampling period, and $\text{⨿}_{T_s}(t)$ is called Dirac comb (or impulse train), built from Dirac delta functions for the given period T_s (see Appendix 2).

1. Prove that the FT of the sampled signal $x_s(t)$ can be written as:

$$X_s(f) = \frac{1}{T_s} \sum_{k=-\infty}^{+\infty} X(f - \frac{k}{T_s}) \tag{8}$$

   Indication: Use the properties listed in Appendix 2.
   Interpretation: The FT of the sampled signal $x_s(t)$ corresponds to the FT of the continuous signal $x(t)$ periodised with period $f_s = \frac{1}{T_s}$.

2. **Reconstruction of the continuous signal from the digital signal.**

   (a) Propose a method in the frequential domain to recover the FT $X(f)$ of the continuous signal from the FT $X_s(f)$ of the sampled signal. Which condition on $X(f)$ and f_s should hold for this method be possible? This condition is known as **Shannon theorem**.

   (b) Under the Shannon condition, it is possible to reconstruct a continuous signal $x(t)$ from the samples $x_s(kT_s)$, by inverting Equation (8). Prove that the reconstruction writes:

$$x_r(t) = \sum_{k=-\infty}^{+\infty} x(kT_s) \frac{\sin(\pi f_s(t - kT_s))}{\pi f_s(t - kT_s)} = \sum_{k=-\infty}^{+\infty} x(kT_s)\text{sinc}(\pi f_s(t - kT_s)) \tag{9}$$

3. **Spectral overlapping.**
   If the Shannon condition is not satisfied (because f_s is too low or because it is impossible to satisfy it), what is the consequence on the spectrum of the sampled signal?

   Example: sinusoidal signal
   Let $c(t) = A\cos(2\pi f_0 t)$ be a sinusoidal signal.

   (a) Let this signal be sampled at frequency f_s, yielding $c_s(k)$, $k \in \{1; N\}$. Is there a range of sampling steps that allows for the reconstruction of the signal? In that case, give this range.

   (b) The signal is now sampled at frequencies $4f_0$ and $2f_0$. Draw the spectrum of the sampled signals at these frequencies.

   (c) Now $c(t)$ is sampled at frequency $f_s = \frac{3}{2}f_0$. Draw the spectrum of $c_s(t)$. What is observed? What is the reconstructed signal if Equation (9) is applied?

Exercise 7 — Signals with limited support.

Let $x(t)$ be a continuous signal, and $x_L(t)$ the signal which is equal to $x(t)$ in the interval $\left[-\frac{L}{2}; \frac{L}{2}\right]$, $L \in \mathbb{R}$, and 0 elsewhere, *i.e.* $x_L(t) = x(t)\text{Rect}\left(\frac{t}{L}\right)$, where $\text{Rect}(t)$ is again:

$$\text{Rect}(t) = \begin{cases} 1 & \text{if } |t| \leq \frac{1}{2} \\ 0 & \text{otherwise} \end{cases} \tag{10}$$

1. Write $X_L(f) = FT\left[x_L(t)\right]$ as a function of $X(f)$.
2. Compute $X_L(f)$ for $x(t) = A\cos(2\pi f_0 t)$. Draw $|X_L(f)|$.
3. Is it theoretically possible to recover $X(f)$ from $X_L(f)$? For the example, propose a method that guarantees that $X_L(f)$ will suffer only a low degradation with respect to $X(f)$.
   Indication: The amplitude of secondary lobes of the sinus cardinal function is low in comparison to the amplitude of the main lobe. It is therefore possible to consider that the amplitude of $\text{sinc}(t)$ becomes 0 after k secondary lobes.
4. **Bonus**: Same question in 2D: compare $x(t,u)$ and $x_L(t,u) = x(t,u)\text{Rect}\left(\frac{t}{L}\right)\text{Rect}\left(\frac{u}{L}\right)$, for $x(t,u) = A\cos\left[2\pi f_0\left(t\cos(\theta) + u\sin(\theta)\right)\right]$.

Exercise 8 — DFT and frequency resolution

The aim of this exercise is to study the three discretization steps on a particular example. Let $c(t) = A\cos(2\pi f_0 t)\text{Rect}\left(\frac{t}{L}\right)$ (see Exercise 7). Here we set $L = 2T_0$.

1. Let sample $c(t)$ at frequency $f_s = 4f_0$. Draw the sampled signal $c_s(t) = \sum_k c(t)\delta(t - kT_s)$, and the digital signal $c(k)$.
2. Draw the FT of the sampled signal.

3. Let us recall that the DFT consists in sampling the FT of the sampled signal in the interval $[-\frac{f_s}{2}; \frac{f_s}{2}[$. Draw the DFT $X(h)$ of $c(k)$. Explain why the secondary lobes of the sinus cardinal do not appear.

4. Let us now sample $c(t)$ at frequency $f_s = 3.7f_0$. Compute $X(h)$ and provide a graphical representation. What do you conclude?

5. Zero-Padding: let $p(k)$ be the following discrete signal with $2N$values:

$$p(k) = \begin{cases} x(k) & \text{if } k \in \{0; N-1\} \\ 0 & \text{if } k \in \{N; 2N-1\} \end{cases} \tag{11}$$

   Compute the DFT $P(h) = DFT\,[p(k)]$, and compare it with $X(h)$. What do you conclude on the effect of zero-padding?

4. Image Filtering

Reminder on spatial and frequential filtering

Image filtering consists in processing the frequential content of an image in order, for example, to:

1. keep only the low frequencies (low-pass filter), e.g. to attenuate the noise;
2. keep only the high frequencies (high-pass filter), e.g. to detect contours (see exercise session 5);
3. select a given range of frequencies (band-pass filter).

Let $x(n, m)$ denote the image value at pixel (n, m). Frequential filtering consists in multiplying the spectrum (i.e. the FT) $X(f, g)$ by a function $H(f, g)$, yielding the filtered spectrum $X_F(f, g)$:

$$X_F(f, g) = X(f, g)H(f, g) \tag{12}$$

This operation in the frequency domain corresponds to a convolution operation ($\star$) in the spatial domain of $x(n, m)$ by the inverse FT $h(n, m)$ of $H(f, g)$:

$$x_F(n, m) = x \star h(n, m) \tag{13}$$

The function $h(n, m)$ is called impulse response of the filter, and corresponds to the filtered image of an image reduced to a Dirac impulse at $(0, 0)$.

In the digital (discrete) case, if the support of h is finite (of size d, supposed to be odd), the filter is called Finite Impulse Response (FIR) filter, and the filtering operation writes:

$$x_F(n, m) = \sum_{i=-\frac{d-1}{2}}^{\frac{d-1}{2}} \sum_{j=-\frac{d-1}{2}}^{\frac{d-1}{2}} h(i, j)x(n - i, m - j) \tag{14}$$

Filtering is then performed according to the following steps:

1. Apply a rotation of angle π of h with respect to its center: $h(n, m) \Rightarrow h(-n, -m) = g(n, m)$;
2. Center the filter at pixel p;
3. Compute the weighted sum of the image pixels and the filter coefficients $g(n, m)$;
4. This weighted sum is the value at p of the filtered image.

Exercise 9 — Filters

1. Which filter has for impulse response $h_1(u, v)$ a discrete 2D Rect ("gate") function, centered and of size 3?
2. Write the filter $h_2(u, v) =$

1	0	-1
1	0	-1
1	0	-1

as a sum of shifted Dirac functions.

3. Show that the filter h_2 is separable, and express it as $h_2 = (f_1)^T f_2$, where f_1 and f_2 are 1D convolution kernels.

4. **Boundary effect.** An image has a bounded support, hence pixels on the boundary do not have neighbors outside the support, which makes the usual application of the convolution formula impossible. What would you propose to compute a filtered value at those pixels?

5. Consider the image $I = \begin{array}{|c|c|c|}\hline 1 & 2 & 3 \\ \hline 4 & 5 & 6 \\ \hline 7 & 8 & 9 \\ \hline \end{array}$. Compute $I \star h_2$ and $(I \star f_1^T) \star f_2$. What do you conclude?

6. **Generalization**: let us consider a separable linear filter $h(n, m) = f_1(n) \cdot f_2(m)$. Prove that the 2D filter can be decomposed into two successive 1D filters.

Exercise 10 — Some filters

Let consider the following 8×8 image, with values (grey levels) ranging from 0 to 7:

0	0	0	0	0	0	0	0
0	5	5	5	5	5	5	0
0	5	7	7	7	7	5	0
0	5	7	7	7	7	5	0
0	5	7	7	7	7	5	0
0	5	7	7	7	7	5	0
0	5	5	5	5	5	5	0
0	0	0	0	0	0	0	0

1. Draw the grey value profile of line 3 of this image (lines are numbered from 0 to 7).

2. Filter this image using a linear convolution by the filter $\frac{1}{9} \times \begin{array}{|c|c|c|}\hline 1 & 1 & 1 \\ \hline 1 & 1 & 1 \\ \hline 1 & 1 & 1 \\ \hline \end{array}$ (compute the grey levels of all pixels of the filtered image). Draw the grey value profile of line 3 of this filtered image. What do you observe?

3. Same question with the filter $\begin{array}{|c|c|c|}\hline 0 & -1 & 0 \\ \hline -1 & 4 & -1 \\ \hline 0 & -1 & 0 \\ \hline \end{array}$

4. One wants to filter an image with a Gaussian filter of standard deviation $\sigma = 0.5$. Compute the coefficients of the filter. Draw the profile of the centerline of the filter. Same questions with $\sigma = 1$.

5. Let us consider that the Gaussian function becomes 0 after 3σ. What is the size of the Gaussian mask in the spatial domain to be used in order to obtain a filter with cut-off frequency (in the frequential domain) equal to $\frac{f_e}{2}$?

Exercise 11 — Convolution

Let us apply the convolution operator on a small region of size 20×11 (in red in the figure) of the following image. This region contains a part of the spinnaker pole, and just below a rope.

The following matrix I contains the grey levels in this region:

$I=$

196	196	199	203	200	188	195	201	204	198	200
198	212	188	206	199	202	204	201	204	187	213
199	187	206	188	209	181	213	208	208	194	186
198	218	202	198	194	210	195	195	208	163	128
184	194	187	206	207	223	202	181	149	125	81
207	195	206	217	205	195	174	151	108	70	74
205	198	187	193	190	161	133	94	80	62	210
179	196	205	186	155	119	75	66	139	223	209
204	196	185	115	91	77	60	189	202	208	188
208	159	105	95	68	90	222	214	206	199	191
146	100	87	55	151	210	210	200	192	207	201
84	69	64	207	212	212	194	211	195	196	147
60	108	219	194	198	187	188	204	191	214	208
174	218	209	195	203	190	205	192	201	201	192
200	215	201	187	204	195	203	192	210	198	148
207	194	192	184	198	197	188	205	164	161	208
176	191	200	195	191	197	207	180	176	217	198
197	190	201	200	208	195	154	171	198	198	192
193	189	205	198	216	141	192	203	194	199	196
196	217	207	186	146	207	211	188	203	188	180

1. Which are in this image the pixels belonging to the spinnaker pole and to the rope?

2. **Low-pass filtering.** Let us consider a binomial low-pass filter of size 3×3, whose convolution mask is:

$h_1=\frac{1}{16}$

1	2	1
2	4	2
1	2	1

Applying this mask on the considered 20×11 region provides the following result (not complete):

111	149	149	150	149	146	148	150	150	149	113
149	199	198	199	199	198	201	203	201	197	149
149	200	199	198	198	199	202	204	200	188	135
148	200		198	201	203	200	196	185	161	105
146	197	199	203	206	205	194	175	151	121	75
149	197	199	203	201		169	142	112	96	75
148	197	196	193	181	157	129	108	102		113
145	195	189	170	146	117	98		140	175	146
146	185	163	131	105	96	115	155	187	200	149
134	154	121	98		126	169	197	202	200	147
99	111	95	109	143	178	201	204	200	196	143
68		117	158	189	200	201	200	199	193	138
78	130	169	193	199	196	196		199	198	145
123	185	200	199	196	195	196	197	200	198	143
150	204	200	194	196	197	197	196	194	187	134
148	197	194		194	196	196	191		184	141
142	192	195	195	196	194	189	183	184	193	149
142	193	197	201	197	186	179	182	191	198	148
146	197	200	198	189		184	191	195	195	144
112	153	151	141	133	139	147	148	147	144	105

Complete the result by computing the missing values (in grey), rounded to their integer part.

3. **Contrast enhancement.** In order to enhance the contrast on the edges of the objects, the following convolution mask is applied:

$h_2 = \frac{1}{16}$

0	-1	0
-1	5	-1
0	-1	0

The resulting filtered image (incomplete) is:

586	373	408	410	410	343	382	405	417	399	589
383	291	117	252	178	238	209	188	220	126	492
412	100	265	121	283	71	277	223	226	226	395
389	309	201	200		257	155	183	325	160	210
321	186	127	221	207	301	237	208	123	162	78
451	170	244	275	216	212	189	198	90		9
441	207	133	185	236	168	161	40	-3	-273	705
290	202	271	262	189	127		-167		497	424
437	236	304	18	40	25	-263	403	268	228	332
531	186		132		-127	536	253	223	183	367
338	39		-265	210	387	224	173	152	247	460
145	-11	-262	510	292	257	149	262		217	130
-66		520	151	194	147		238	141	274	487
392	384	212	182	228	160	252	158	211	200	403
404	262	202	151		181	235	150	295	270	142
465	165	181	148	214	207	128	301			533
285	195	221	200	157	195	316	141	121	352	373
426	172	210	198	238	275		120	251	184	368
383	140	230	183	387	-105	251	270	167	219	409
570	493	427	379	121	537	468	323	445	358	516

Complete the result by computing the missing values (in grey).

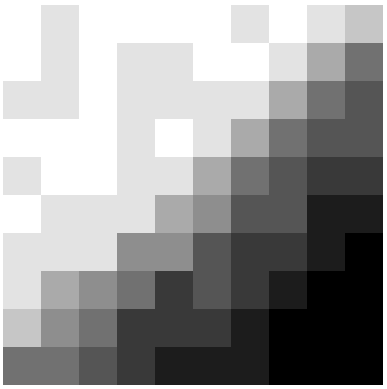
5. Filtering and Edge Detection

Exercise 12 — Sobel edge detection filter

Let I be defined as:

$$I = \begin{bmatrix} 9 & 8 & 9 & 9 & 9 & 9 & 8 & 9 & 8 & 7 \\ 9 & 8 & 9 & 8 & 8 & 9 & 9 & 8 & 6 & 4 \\ 8 & 8 & 9 & 8 & 8 & 8 & 8 & 6 & 4 & 3 \\ 9 & 9 & 9 & 8 & 9 & 8 & 6 & 4 & 3 & 3 \\ 8 & 9 & 9 & 8 & 8 & 6 & 4 & 3 & 2 & 2 \\ 9 & 8 & 8 & 8 & 6 & 5 & 3 & 3 & 1 & 1 \\ 8 & 8 & 8 & 5 & 5 & 3 & 2 & 2 & 1 & 0 \\ 8 & 6 & 5 & 4 & 2 & 3 & 2 & 1 & 0 & 0 \\ 7 & 5 & 4 & 2 & 2 & 2 & 1 & 0 & 0 & 0 \\ 4 & 4 & 3 & 2 & 1 & 1 & 1 & 0 & 0 & 0 \end{bmatrix}$$

and corresponding to the following image:



to which Sobel filters will be applied.

The impulse responses of Sobel filters (convolution masks) are defined as:

$$S_x = \begin{bmatrix} 1 & 0 & -1 \\ 2 & 0 & -2 \\ 1 & 0 & -1 \end{bmatrix} \text{ and } S_y = \begin{bmatrix} 1 & 2 & 1 \\ 0 & 0 & 0 \\ -1 & -2 & -1 \end{bmatrix}$$

1. Complete the computation of the horizontal filtering $I_x = I \star S_x$:

$$I_x = \begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & -4 & 0 & 3 & -1 & 1 & -2 & -1 & -3 & -10 & -5 & 0 \\ 0 & -3 & 1 & 1 & \mathbf{?} & 2 & 1 & -4 & -10 & -13 & \mathbf{?} & 0 \\ 0 & -1 & 2 & -1 & -3 & 1 & -2 & \mathbf{?} & -14 & -11 & -4 & 0 \\ 0 & 1 & 2 & -3 & -2 & -2 & -10 & \mathbf{?} & -12 & -6 & -1 & 0 \\ 0 & 1 & 1 & -3 & -4 & -7 & \mathbf{?} & -12 & -9 & -5 & 0 & 0 \\ 0 & -1 & -1 & -4 & -8 & \mathbf{?} & -13 & -8 & -7 & -7 & -1 & 0 \\ 0 & -3 & \mathbf{?} & -8 & -11 & -8 & -9 & -6 & -6 & -7 & -2 & 0 \\ 0 & -6 & \mathbf{?} & -10 & -11 & -4 & -4 & -7 & -6 & -4 & -1 & 0 \\ 0 & -6 & \mathbf{?} & -10 & -9 & -2 & -2 & -7 & -5 & -1 & 0 & 0 \\ 0 & \mathbf{?} & -6 & -9 & -8 & -3 & -1 & -5 & -4 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}$$

N.B: Padding with copy on the image I is used here.

2. Complete the computation of the vertical filtering $I_y = I \star S_y$:

$$I_y = \begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & -1 & -3 & -3 & 0 & 1 & -3 & -8 & -11 & 0 \\ 0 & -3 & -1 & -1 & \mathbf{?} & -4 & -3 & -4 & -10 & -15 & \mathbf{?} & 0 \\ 0 & 1 & 2 & 1 & 1 & 1 & -4 & \mathbf{?} & -14 & -11 & -6 & 0 \\ 0 & 1 & 2 & 1 & 0 & -2 & -8 & \mathbf{?} & -12 & -8 & -5 & 0 \\ 0 & -1 & -3 & -3 & -4 & -9 & \mathbf{?} & -10 & -7 & -7 & -8 & 0 \\ 0 & -1 & -3 & -6 & -10 & \mathbf{?} & -11 & -8 & -5 & -5 & -7 & 0 \\ 0 & -5 & \mathbf{?} & -12 & -15 & -14 & -9 & -6 & -6 & -5 & -4 & 0 \\ 0 & -6 & \mathbf{?} & -14 & -13 & -10 & -6 & -5 & -6 & -4 & -1 & 0 \\ 0 & -14 & \mathbf{?} & -8 & -7 & -6 & -6 & -5 & -3 & -1 & 0 & 0 \\ 0 & \mathbf{?} & -6 & -3 & -2 & -3 & -3 & -1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}$$

3. Complete the computation of the norm of the gradient $||\overrightarrow{G}|| = \sqrt{I_x^2 + I_y^2}$ (rounded to the closest integer):

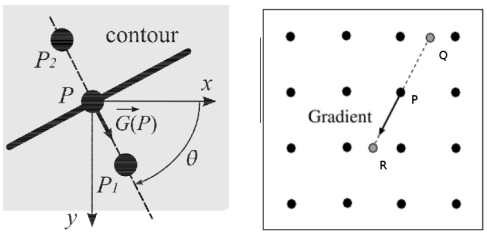
$$||\overrightarrow{G}|| = \begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 4 & 0 & 3 & 3 & 3 & 2 & 1 & 4 & 13 & 12 & 0 \\ 0 & 4 & 1 & 1 & ? & 4 & 3 & 6 & 14 & 20 & ? & 0 \\ 0 & 1 & 3 & 1 & 3 & 1 & 4 & ? & 20 & 16 & 7 & 0 \\ 0 & 1 & 3 & 3 & 2 & 3 & 13 & ? & 17 & 10 & 5 & 0 \\ 0 & 1 & 3 & 4 & 6 & 11 & ? & 16 & 11 & 9 & 8 & 0 \\ 0 & 1 & 3 & 7 & 13 & ? & 17 & 11 & 9 & 9 & 7 & 0 \\ 0 & 6 & ? & 14 & 19 & 16 & 13 & 8 & 8 & 9 & 4 & 0 \\ 0 & 8 & ? & 17 & 17 & 11 & 7 & 9 & 8 & 6 & 1 & 0 \\ 0 & 15 & ? & 13 & 11 & 6 & 6 & 9 & 6 & 1 & 0 & 0 \\ 0 & ? & 8 & 9 & 8 & 4 & 3 & 5 & 4 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}$$

4. Propose a threshold value to binarize the image and extract the contours from $||\overrightarrow{G}||$.

Exercise 13 — Suppression of non-maxima

The problem of edges detection can be formulated as the selection of local maxima of the gradient amplitude in the direction of the gradient. A discrete filter approximating the gradient is first applied, and then a post-processing step allows suppressing the non-maxima points.

Let $\overrightarrow{G(P)}$ be the image gradient of a pixel P. One has to check whether P is a local maximum of $||\overrightarrow{G(P)}||$ in the direction of the gradient (Figure (a) below).



(a) (b)

Two algorithms are considered to do this:

1. Discretization of orientations. Propose an algorithm to determine the discrete direction to be assigned to point P (considering 8-neighborhood).

2. Sub-pixel interpolation. Propose a method based on the computation of the gradient of the two neighbors that are in the direction of the gradient.

   Indication: Determine the coordinates of points Q and R in Figure (b) above, and compute the gradient module using linear interpolation.

Exercise 14 — Multi-resolution pyramid and aliasing

We assume a multi-resolution pyramid built from a Gaussian filtering. The aim here is to estimate the parameters of the filter so as to not induce aliasing effect during sub-sampling. It is assumed that the original image (with the best resolution) has been correctly sampled.

Recall that the FT of a Gaussian function is a Gaussian function with inverted standard deviation ($\sigma_f = \frac{1}{\sigma_s \pi}$):

$$TF\left[e^{-b^2(t^2+u^2)}\right] = \frac{\pi}{b^2} e^{-\frac{\pi^2(f^2+g^2)}{b^2}}$$

1. By using a Gaussian filter, is it theoretically possible to cancel totally the aliasing effect? Explain.

2. Assuming that in practice the Gaussian filter has a cut-off frequency at 3σ, determine the standard deviation of the Gaussian mask to be applied in the spatial domain, so as to guarantee that the convolution $I \star G$ does not introduce aliasing when moving to the next resolution level.

Appendix 1: Dirac delta function

Intuitively, the Dirac delta function can be interpreted as follows: $\delta(t) = \lim_{T \to 0} \left[\frac{1}{T} \text{Rect}\left(\frac{t}{T}\right) \right]$.

Exercise 15 — Properties of the Dirac delta function

1. $\int_{-\infty}^{+\infty} \delta(t)dt = 1$.
2. $x(t)\delta(t - t_0) = x(t_0)\delta(t - t_0)$
3. $x \star \delta_{t_0}(t) = x(t - t_0)$ with $\delta_{t_0}(t) = \delta(t - t_0)$
4. scaling property: $|\alpha|\delta(\alpha t) = \delta(t)$
5. $\text{FT}\left[\delta_{t_0})\right](f) = e^{-2i\pi f t_0}$ and $\text{FT}\left[t \mapsto e^{2i\pi f_0 t}\right] = \delta(f - f_0)$

Appendix 2: Sampling

The Dirac distribution $\delta(t)$ is defined as:

$$\delta(t) = \begin{cases} 0 & \text{if } t \neq 0 \\ \infty & \text{otherwise} \end{cases} \tag{15}$$

Dirac Comb

The Dirac comb is defined as:
$\text{⨿}_{T_e}(t) = \sum_{k=-\infty}^{+\infty} \delta(t - kT_e)$.

Sampling model

The ideal sampling of a 1D signal $x(t)$ with periode T_e is:

$$x_e(t) = \sum_{k=-\infty}^{+\infty} x(t)\delta(t - kT_e) = x(t)\text{⨿}_{T_e}(t) \tag{16}$$

With this model, we have $\lim_{T_e \to 0} x_e(t) = x(t)$ and $\lim_{T_e \to 0} \int_{-\infty}^{+\infty} x_e(t)dt = \int_{-\infty}^{+\infty} x(t)dt$.

Exercise 16 — FT of Dirac distribution and other usual functions

Signal	FT	Signal	FT
1	$\delta(f)$	$\delta(t)$	1
$e^{2i\pi f_0 t}$	$\delta(f - f_0)$	$\delta(t - t_0)$	$e^{-2i\pi f t_0}$
$\text{⨿}_T(t) = \sum_{k=-\infty}^{+\infty} \delta(t - kT)$	$\frac{1}{T}\text{⨿}_{\frac{1}{T}}(f) = \frac{1}{T} \sum_{k=-\infty}^{+\infty} \delta(f - \frac{k}{T})$		
$\text{Rect}\left(\frac{t}{T}\right)$	$T\text{sinc}\left(\pi f T\right)$	$f_0\text{sinc}\left(\pi f_0 t\right)$	$\text{Rect}\left(\frac{f}{f_0}\right)$

Practical works

Practical work materials (Jupyter notebooks and data) are to be retrieved from the course's web site: `https://frama.link/mR68kcMB`.

Practical work 1: introduction and image enhancement

- Quick start for Python (10 minutes!) : https://www.stavros.io/tutorials/python/
- Quick start for Numpy : https://numpy.org/devdocs/user/quickstart.html#
- For Matlab users: Numpy is very similar but with some important difference, see http://mathesaurus.sourceforge.net/matlab-numpy.html.
- Keep in mind that in Python, exception of variable of scalar type, all is reference and affectation is not a copy.

Short introduction to image processing with Python

Help: use the function `help()` to get information on a Python objet.

Images are stored as arrays that is the default type of the `numpy` module. Defaut type of array elements is `float64` according to the IEEE754 norm. Special float values are defined: infinity (`inf`) and undefined (`nan`, *not a number*), and some numerical constants, such as π.

[ ]:
```
# import numpy
import numpy as np

# predefined constants
print(np.inf,np.nan,np.pi)

# some values
print( 1., 1e10, -1.2e-3)
```

Creating an array: several ways.

1. From a list of values (formally any Python iterable object). Elements of an array have the same **type**, determined by Numpy:

[ ]:
```
V = np.array([1,2,3])
M = np.array([[1,2,3],[4,5,6.]])
print ("V is of type",V.dtype)
print ("M is of type",I.dtype)
```

2. Without values: Numpy has constructors such as `empty()`, `zeros()`, `ones()`... Shape should be given (see below). Important: `empty()` does not initialize array elements.

[ ]:
```
I = np.zeros((3,4))
print(I)
J = np.empty((4,3))
print(J)
```

3. From a sequence, prefer `arange()` from numpy to `range()` from python.

[ ]:
```
print(np.arange(10))
print(np.arange(0,10,2))
print(np.arange(9,-1,-.5))
```

Shape of an array

Shape decribes the number of elements for each dimension. A vector is of dimension 1, a matrix is of dimension 2. Superior dimensions are possible. Shape is not size that is the number of elements of an array. Type of shape is always a tuple of integers. With previous example:

[ ]:
```
print(I.shape, I.size)
print(J.shape, J.size)
print(V.shape, V.size)
```

An important function/method is `reshape()` to change the shape of an array. Typical usage of `reshape()` is to transform a vector into a matrix or reciprocally.

[ ]:
```
K = np.arange(12)).reshape((3,4))
print(K)
print(np.reshape(K,(12)))
print(K.reshape((2,2,3)))
```

Elements of an array

Access element by indices: two syntaxe are possible, the first given in the example is prefered. Negative index is possible with the same meanning of Python list.

[ ]:
```
I = np.arange(12).reshape((3,4))
print(I[1,2])
print(I[0][0])
print(I[-1,0])
```

Access by group of indices using the operator : allows to extract subarray. General syntaxe is `start:end:step` and it is very powerfull:

[ ]:
```
print('extract the first line')
print(I[0,:])
print(I[0,0:])
print(I[0,::])
print(I[0,::1])

print('extract center of the array')
print(I[1:3,1:3])

print('extract elements with even indices')
print(I[::2,::2])

print('print the horizontal mirror of an array')
print(I[:,::-1])
```

Array arithmetic

Operators and functions can be applied to arrays. Mostly, operations are element-wise (i.e. applied element by element). The consequence is arrays should have the same shape. One operand can

also be scalar in most of time.

[ ]:
```
A = np.arange(12).reshape((3,4))
B = 2 * A + 1
C = A + B
D = np.cos(2*np.pi*A/12)

print (D)
print (D**2)
print (D>0)
```

Array may be viewed as matrix, we can make some linear algebraic manipulation. For example, `np.matmul()` is the matrix multiplication. It can be used to build matrix from vector. An example, using the transpose operator `T`.

[ ]:
```
L = np.arange(1,6).reshape((1,5))
# transpose of L. Warning: C remains a reference to L
C = L.T
# This could be better if your want to touch L
C = L.T.copy()

print("A 5*5 matrix:")
print(np.matmul(C,L))

print("A dot product, but result is a matrix:")
print(np.matmul(L,C))
print(np.matmul(L,C)[0,0])

print("dot() is prefered with vectors:")
V = np.arange(1,6)
print(V.dot(V))
print(np.dot(V,V))
```

Images

We make use of PIL module (https://pillow.readthedocs.io/en/stable/reference/Image.html) to load and write an image and easily converted to Numpy array. Be careful: array type depends on image.

[ ]:
```
from PIL import Image

# reading an image and convert to array
myimage = np.array(Image.open('image.png'))

# write an image (alternative format) from an array
Image.fromarray(myimage).save('image.jpg')
```

Array can be displayed as an image using Matplotlib module. Here a short example:

[ ]:
```
import matplotlib.pyplot as plt

# minimal example:
plt.imshow(myimage)
plt.show()

# with more controls:
w,h=400,400
plt.figure(figsize=(w/80,h/80))  # optional, to control the size of figure (unit:
 ↪ pixel)
plt.gray() # optional call to display image using a gray colormap
plt.title('This is an image') # optional: add a title
plt.axis('off') # optional: remove axes
plt.imshow(myimage)
plt.show()
```

See also: - https://matplotlib.org/3.1.1/tutorials/introductory/images.html - https://matplotlib.org/gallery/images_contours_and_fields/image_demo.html#sphx-glr-gallery-images-contours-and-fields-image-demo-py).

Exercice 1

In this exercice, we work with image `img/moon.png`. If possible give two solutions : one with loops (for, while, ...) and one without loops.

1. Write and test a function `openImage()` getting an image filename as argument and returning the array of pixel values.

[ ]:
```
from PIL import Image
import numpy as np

def openImage(fname):
    """ str -> Array
    (notation above means the function gets a string argument and returns an␣
 ↪Array object)
    """
```

2. Write and test a function `countPixels()` getting an array and an integer `k` as arguments and returning the number of pixels having the value `k`.

[ ]:
```
def countPixels(I,k):
    """ Array*int -> int"""
```

3. Write and test a function `replacePixels()` getting an array and two intergers and replacing pixels having `k1`value to `k2` value and returning the new array. Be aware to not modify `I`.

[ ]:
```
def replacePixels(I,k1,k2):
    """ Array*int*int -> Array """
```

4. Write and test a function `normalizeImage()` getting an array and two integers `k1` and `k2` and returning an array with elements normalized to the interval $[k_1, k_2]$.

```
[ ]: def normalizeImage(I,k1,k2):
         """ Array*int*int -> Array """
```

5. Write and test a function `inverteImage()` getting an array and returning and arry having inverted pixel values (i.e. the transform $k \mapsto 255 - k$

```
[ ]: def inverteImage(I):
         """ Array -> Array """
```

6. Write and test a function `computeHistogram()` getting an array and returning its histogram. Type of histogram can be an array or a list. It is forbidden to use an histogram method from a Python module. Is it possible to compute the histogram without explicitely visiting array pixels?

```
[ ]: def computeHistogram(I):
         """ Array -> list[int] """

     # use comments to answer to a verbal question
```

7. Write and test a function `thresholdImage()` getting an array `I` and an integer `s` and returning an array having elements set to 0 if corresponding element of `I` is lower than `s` or 255 otherwise.

```
[ ]: def thresholdImage(I,s):
         """ Array*int -> Array """
```

8. Using previous functions, give a series of instructions to read then to display an image, plot the histogram (one can use `plot()` or `bar()` from `matplotlib.pyplot` module), inverse the image and display it, plot its histogram.

```
[ ]: import matplotlib.pyplot as plt

     ## your code start below
```

9. Give a series of instructions to read and display an image, plot the histogram, normalize the image to the interval $[10, 50]$, compute the new histogram, display the image and the histogram. Remark: `imshow()` normalizes image. To avoid this and see the effect of the normalization, use `imshow()` with parameters `vmin=0,vmax=255`. Comment the results.

```
[ ]:
```

10. Same question than 9. remplacing the normalization by a thresholding with parameter $s = 127$.

```
[ ]:
```

Exercice 2 - generate images

1. Create the array `I` 4 by 4 corresponding to the following image:
   Black pixels have value 0, white pixels value 255, and grey pixels value 127. Display the image using `imshow()` and plot the histogram.

[ ]:

2. We want to generate a matrix having random values. Functions `rand()` and `randn()` from `numpy.matlib` module generate array of given shape with random values following respectively a uniform distribution on $[0, 1[$ and a normal distribution. Create an array of shape 512 by 512 having **integer** elements following an uniform distribution in the set $\{0, 1, \cdots, 255\}$. We also want to create an array following a gaussian distribution with a mean of 128 and a standard deviation of 16 and with **integer** values. Display the images and their histogramms. Discuss the results.

[ ]:
```
import numpy.matlib
```

Exercice 3: image manipulation

In this exercice, we work with image `img/pout.png`.

1. Read and display this image

[ ]:

2. Examine the histogram. Determine the extrema of the image. What can you say about the quality of this image?

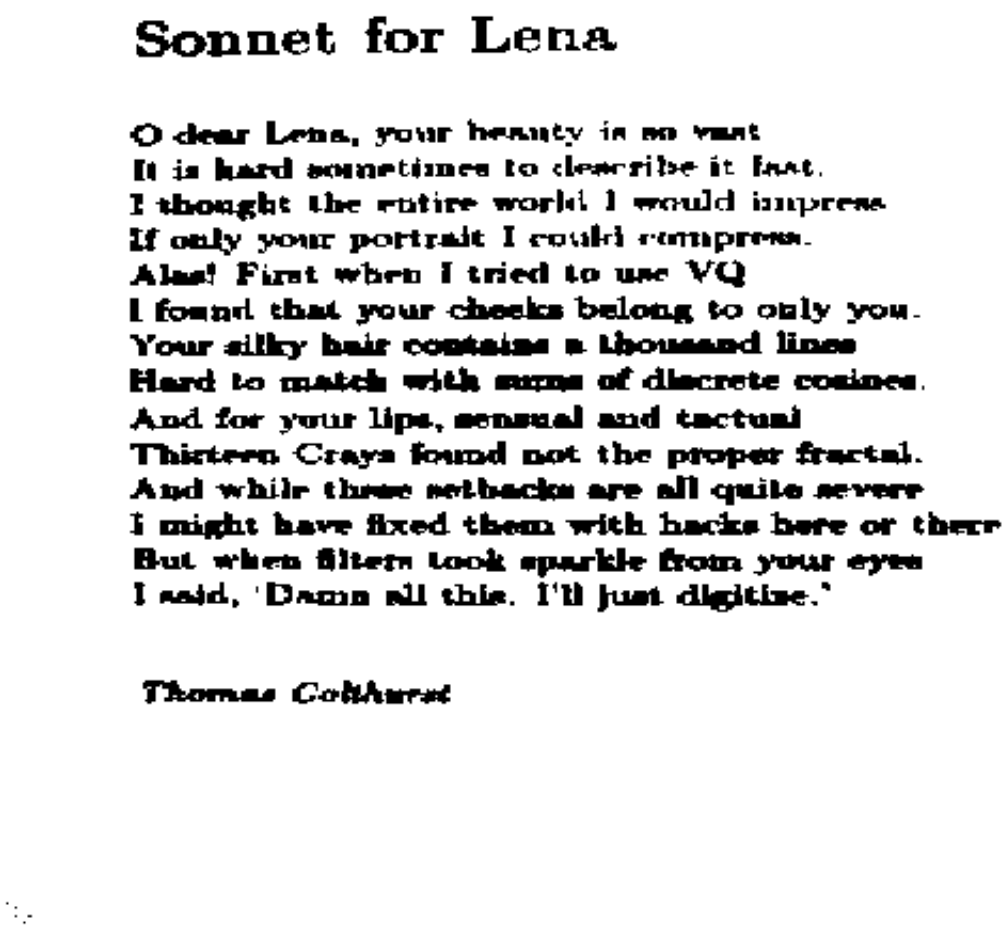
[ ]:

3. Using functions from Exercice 1, write the function `histogramEqualization()` getting one image, its histogram, applying an histogram equalization and returning the new image. Test this function on `pout.png` and discuss the result.

[ ]:
```
def histogramEqualization(I,h):
    """ Array * (list[int] -> Array """
```

Practical work 2 : Fourier transform

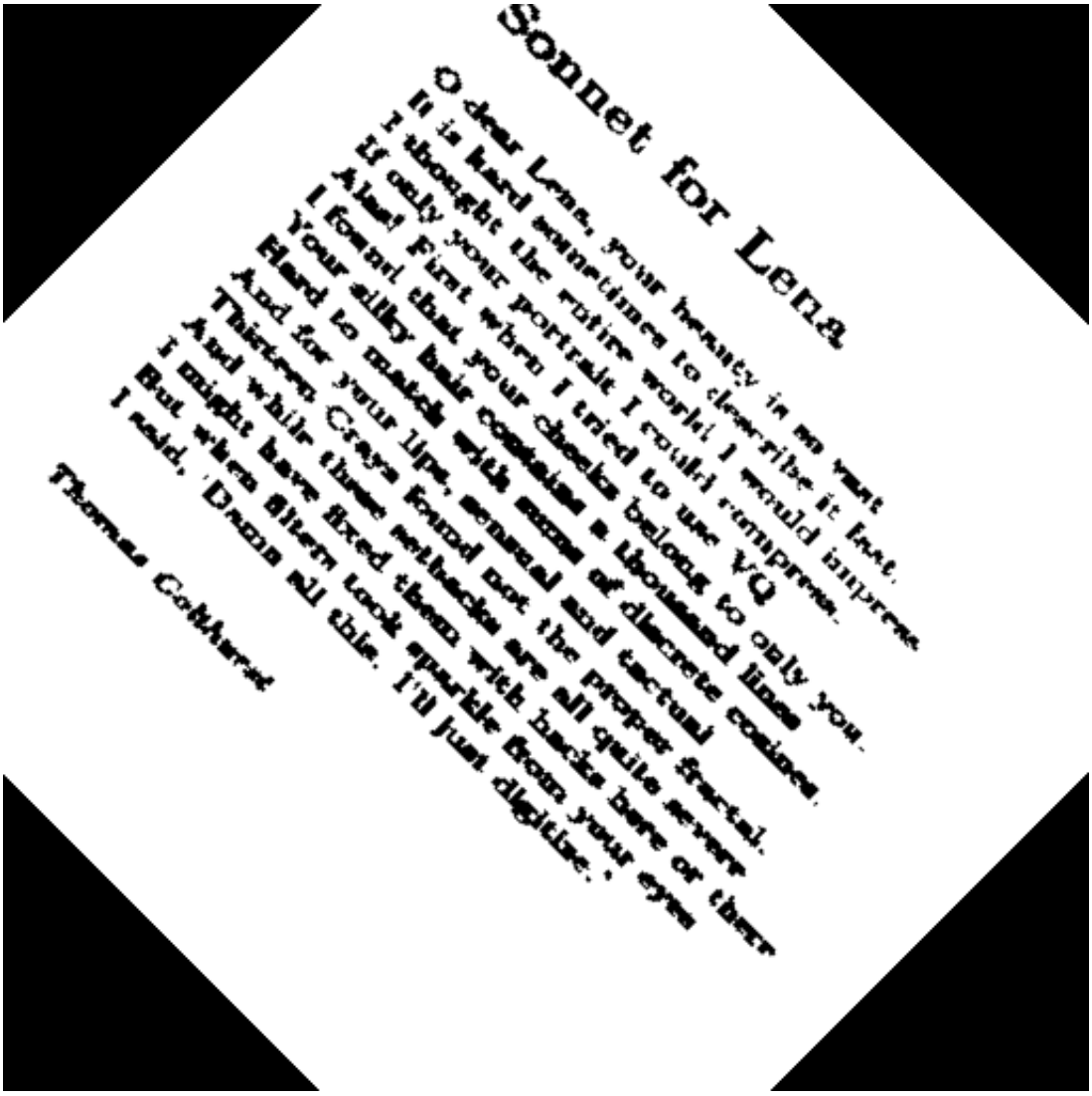
This practial work is dedicated to the study of the discrete Fourier transform applied on the two following images:



Sonnet for Lena

O dear Lena, your beauty is so vast
It is hard sometimes to describe it fast.
I thought the entire world I would impress
If only your portrait I could compress.
Alas! First when I tried to use VQ
I found that your cheeks belong to only you.
Your silky hair contains a thousand lines
Hard to match with sums of discrete cosines.
And for your lips, sensual and tactual
Thirteen Crays found not the proper fractal.
And while these setbacks are all quite severe
I might have fixed them with hacks here or there
But when filters took sparkle from your eyes
I said, 'Damn all this. I'll just digitize.'

Thomas Colthurst



and analyze the properties of their spectrum. To this end, we make use of the following functions provided by the module `numpy.fft`:

- `fft2()` to compute the Fourier transform on an image
- `fftshift()` to center the low frequencies
- `abs()` (from `numpy`) to compute the module of a complexe array

In most of cases, high frequencies have lower energy compare to low frequencies. We will use a logarithmic scale by applying $\log(1 + abs(TF))$ to display the spectrum.

[1]:
```
import numpy as np
from numpy.fft import fft2,fftshift
from PIL import Image

son = np.array(Image.open('img/son.png'))
sonrot = np.array(Image.open('img/sonrot.png'))
```

Exercice: properties of Fourier transform applied on natural images

1. Write the following functions:

- `computeFT(I)` returning the Fourier transform of image `I`,
- `toVisualizeFT(If)` returning the centered module of a complex array `If` (the Fourier transform of an image),
- `toVisualizeLogFT(If)` similar to the previous function but use a logarithmic scale.

[ ]:
```
def computeFT(I):
    """ Array -> Array[complex] """

def toVisualizeFT(If):
    """ Array[complex] -> Array[float] """

def toVisualizeLogFT(If):
    """ Array[complex] -> Array[float] """
```

2. Write a series of instructions that

- compute the Fourier transform of `son` and `sonrot`,
- compute and display the module using a logarithmic scale,
- threshold the module with a parameter of 1.10^5 (use the function of TME1)
- display the thresholded spectrum

[ ]:
```
import matplotlib.pyplot as plt

# your code below
```

3. Interpretation: discuss the results obtained on thresholded FT module. What property of the Fourier transform is shown ?

4. Write the function `blend()` getting two images, one float parameter $\alpha \in [0, 1]$, calculating $\alpha I_1 + (1 - \alpha) I_2$ and returning the result.

[ ]:
```
def blend(I1,I2,alpha):
    """ Array**2*float -> Array """
```

5. Apply the previous function on images `son` and `sonrot` and $\alpha = \frac{1}{2}$, compute the Fourier transform, threshold the module and visualize the result.

[ ]:
```

```

6. Compare the latter result with those of question 2: what property of the Fourier transform is shown? What is the behaviour of α in the resulting spectrum?

7. We want to determine the text orientation in image `sonrot` and produce a new image with horizontal text. Write the function `rectifyOrientation()` that:

- computes the FT module of image given in parameter and threshold it at 3×10^5,
- from thresholded module determines the main orientation using the function `mainOrientation()`

- produces the rectified image applying a rotation with a suitable angle using `rotateImage()`

[ ]:
```
def mainOrientation(I):
    """ Array -> tuple[Iori,float]
        return image of orientation (32 bins) and the main orientation (degree)␣
 ↪from a Fourier transform module
    """
    n, m = I.shape

    size = 32
    x = np.array(range(size))
    ori = np.vstack((np.cos(np.pi*x/size), np.sin(np.pi*x/size))).T

    Iori = np.zeros((n, m))
    orients = np.zeros((size))

    for i in range(1,n+1):
        for j in range(1,m+1):
            if I[i-1, j-1] > 0:
                v = np.array([j-m/2, -i + n/2])
                if i > n/2:
                    v = -v
                    prod = np.matmul(ori, v)
                    maxi = prod.max()
                    if maxi > 0:
                        imax = np.nonzero(prod == maxi)
                        Iori[i-1, j-1] = imax[0]
                        orients[imax] += 1

    maxori = np.nonzero(orients == orients.max())[0][0]
    return (Iori, 180*maxori/size - 90)

def rotateImage(I,a):
    """ Array*float -> Array
        return a rotation of angle a (degree) of image I
    """
    return np.array(Image.fromarray(I).rotate(a, expand=True, fillcolor=127))

####### your code below
```

8. Experiment `rectifyOrientation()` on `sinrot`, and on a rotation of `img/port.jpg` (using `rotateImage()`) with various rotation angles.

[ ]:
```

```

Practical work 3: 2D sampling and aliasing

Properties studied in 1D apply in 2D. The following results can be admitted: - given a regular grid, a sampling of a continuous 2D signal can be modelled as follow:

$$x_s(t,u) = x(t,u)C(t,u) \quad \text{with} \quad C(t,u) = \sum_{k=-\infty}^{+\infty}\sum_{l=-\infty}^{+\infty} \delta(t-kT_s, u-lT_s)$$

C is the analog of Dirac comb (also called impulse train) in 2D - spectrum of x_s writes:

$$X_s(f,g) = \frac{1}{T_s^2}\sum_{k=-\infty}^{+\infty}\sum_{l=-\infty}^{+\infty} X(f-kf_s, g-lf_s)$$

2D sampling then implies a periodisation of the spectrum for the two dimensions - it is possible to reconstruct the original signal from the sampled signal if 2D Shannon condition is verified (band limited signal) with:

$$x_r(t,u) = \sum_{k=-\infty}^{+\infty}\sum_{l=\infty}^{+\infty} x_s(kT_s, lT_s)\,\mathrm{sinc}(\pi f_s(t-kT_s))\,\mathrm{sinc}(\pi f_s(u-lT_s)) \tag{1}$$

so called Shannon interpolation.

Exercice 1: aliasing and windowing of 2D signals

Given the following signal:

$$s_\theta(t,u) = A\cos(2\pi f_0(t\cos\theta + u\sin\theta))$$

Here an example with $\theta = \frac{\pi}{4}$:

The goal of this exercice is to study the limit conditions of sampling of this image in order to avoid aliasing.

[ ]:
```
import numpy as np
from numpy.fft import fft2, fftshift
import matplotlib.pyplot as plt
# for interactive ploting, see surf() below
%matplotlib notebook
from matplotlib import cm
from matplotlib.colors import Normalize
from mpl_toolkits.mplot3d import Axes3D

def sinusoid2d(A, theta, size, T0, Te):
    """ double**2*int*double**2 -> Array[double] """
    ct = np.cos(theta/180*np.pi)
    st = np.sin(theta/180*np.pi)
    x, y = np.meshgrid(np.arange(0, size, Te), np.arange(0, size, Te))
    return A*np.cos(2*np.pi*(y*ct - x*st)/T0)

def shannonInterpolation(I, Ts, size):
```

```
    """ Array*int*double -> Array[double] """
    n, m = I.shape
    x, y = np.meshgrid(np.arange(0, size), np.arange(0, n))
    Y = np.sinc(x/Ts-y)
    x, y = np.meshgrid(np.arange(0, size), np.arange(0, m))
    X = np.sinc(x/Ts-y)
    return np.matmul(X.T, np.matmul(I, Y))

def imshow(I,title=None):
    """ display an image """
    plt.figure(figsize=(500//80,500//80))
    plt.gray()
    plt.imshow(I)
    if title: plt.title(title)
    plt.show()

def surf(Z,title=None):
    """ 3D plot of an image """
    X,Y = np.meshgrid(range(Z.shape[1]), range(Z.shape[0]))
    fig = plt.figure(figsize=(600/80,600/80))
    if title: plt.title(title)
    # ax = fig.gca(projection='3d')
    ax = plt.subplot(projection='3d')
    ax.plot_surface(X, Y, Z, cmap=cm.coolwarm, linewidth=0, antialiased=False)
    plt.show()
```

1. We provide the function `sinusoid2d(A, theta, L, T0, Ts)` that allows to sample signal s_θ with a sampling period of `Ts` (the grid is regular with the sample sampling value for directions u and t). Paremeters `A`, `theta`, `L` and `T0` respectively control the amplitude, orientation and period ($T_0 = \frac{1}{f_0}$) of signal s_θ. Generate a pseudo continuous signal s_{45} with `A=1`, `theta = 45`, `L = 512`, `T0 = 64` and `Ts=1`.

[ ]:

2. (a) What is the maximal frequency of previous signal s_{45} in direction t (denoted $f_t^{\max}$) and direction u (denoted $f_u^{\max}$)? Let $f_m = \max(f_t^{\max}, f_u^{\max})$. Explain why f_m can be used to derive the limit frequency (in sens of Shannon) for the sampling of s_{45}.

2. (b) Sample s_{45} with $f_s = 16 f_m$ and display the sampled signal.

[ ]:

2. (c) Compute the Fourier transform of the sampled signal and display frequencies. One can use `surf()` function for an interactive 3D plot.

[ ]:

2. (d) Comment the spectrum:
   - verify the presence of the two Dirac peaks
   - for various values of T_s, observe changes in the spectrum. Compare with the spectrum of the continuous signal (s_{45}). What is the origin of theses differences?
   - (Bonus question):
     - Why, aside the two Dirac peaks, there are somes structures? Explain the origin of these lobes.
     - Increase T_0 in order to obtain a unique peak. Explain the origin of this fusion. Verify the limit value of T_0 for which the two peaks interfer.

[ ]:

3. (a) Sample s_{45} with $f_s = 4f_m$ and display the sampled signal.

[ ]:

3. (b) Write a function `error()` implementing the relative average error $\epsilon_r = \frac{1}{2AL^2}\sum_{k=0}^{L}\sum_{l=0}^{L}|x_r(k,l) - x_d(k,l)|$.

[ ]:

3. (c) Reconstruct the sampled signal. Display original and reconstructed signal. Print the relative average error between the original and reconstructed images. What is the origin of this error?

[ ]:

4. Same question than 3. with $f_s = \frac{3}{2}f_m$. Comment the effects of aliasing.

[ ]:

5. Consider the continuous signal with an orientation of $\theta = 10$. What is the value of f_m? With a sampling of $\frac{3}{2}f_s$ what is the additional drawback appearing after the reconstruction? Explain.

[ ]:

6. Bonus question: write a function `shannonInterpolationLoop()` implementing equation (1) using two loops, in a C way. Compare and discuss the run time of this function and `shannonInterpolation()` on a small signal ($L = 64$). Runtime can be measured using `tic()` and `tac()` functions.

[1]:
```
from time import process_time
mytime = 0
def tic():
    """ NoneType -> NoneType """
```

```
    global mytime
    mytime = process_time()
def tac():
    """ NoneType -> int """
    global mytime
    print (process_time()-mytime)
    mytime = process_time()

### your code starts below
```

Exercice 2: aliasing on natural images

In this exercice, we study aliasing on image `img/barbara.png`. Aliasing occurring with subsample of image, we propose to write a code that implements a subsample (using function `subSample2()` of factor 2 on the image.

[ ]:
```
from PIL import Image

barbara = np.array(Image.open('img/barbara.png'))

def subSample2(I):
    """ Array -> Array """
    return I[::2,::2]
```

1. Explain what is a subsample of factor 2 and the impact when applied on an image.
2. Write a code that

- iterates the subsampling process
- at each iteration, computes the Fourier transform of the subsampled image

  Display subsampled images and their spectrum. Describe and interpret the effects of aliasing. Why aliasing is a drawback ?

[ ]:
```

```

3. Bonus question: same question with the image `img/mandrill.png`.

[ ]:
```
mandrill = np.array(Image.open('img/mandrill.png'))
### your code and comments start below
```

Practical work 4: Frequency filtering, color

[1]:
```
import numpy as np
from numpy.fft import fft2,ifft2,fftshift
import matplotlib.pyplot as plt
from PIL import Image

def imshow(I,title=None,size=500):
    """ display an image with a specific size """
    plt.figure(figsize=(size//80,size//80))
    plt.gray()
    plt.imshow(I)
    if title: plt.title(title)
    plt.show()
```

Exercice 1 - Frequency filtering

1. Compute than display the centered module of Fourier transform of `img/mandrill.png` (use functions seen in previous lessons).

[ ]:
```

```

2. Write a function `idealLowPassFilter(n,m,fc)` returning an ideal low pass filter with frequency cutoff f_c and size $n \times m$. Recall: this function set to 1 pixels at Euclidian distance f_c from the center (null frequency).

[ ]:
```

```

3. Write a function `lowPass(I,fc)` performing a low pass filtering of an image I. The function should

- compute the centered Fourier transform of I
- multiply point-by-point the spectrum with the ideal low filter produced by `idealLowPassFilter()`
- uncenter the filtered spectrum and apply the inverse Fourier transform (use function `ifft2()` from module `numpy.fft`)
- return the real part of filtered image

[ ]:
```

```

4. Experiment this function on `img/mandrill.png` and `img/lena.jpg` with various values of cut off f_c.

- give two effects that appears when f_c decreases,
- propose two applications of this filtering.

[ ]:
```

```

Exercice 2 - Linear filtering (convolution)

1. Given a kernel convolution of size $d \times d$, d being odd. How many lines and columns should be added to each side of the image to apply this filter? The image is supposed surrounded by zero values.
2. Write a function `imagePad(I,h)` getting an image and a kernel, returning a new image padded with zeros according to question 1. It is not allowed to use a module implementing the padding.

[ ]:

3. Write a function `conv2(I,h)` getting an image and a kernel and returning the convolution of I by h. The function should return an image having the same shape than I. It is not allowed to use a module implementing the convolution.

[ ]:

4. Try this function on mean filter of size 3×3, 5×5 and 7×7. Discuss the results.

[ ]:

5. Display the transfert function of these mean filters. For a better visualization, use the zero-padding technique to obtain a filter with a large size (for instance 256×256). Use `imshow()` and `toVisualizeLogFT()`.

[ ]:

6. Interpretation: what is the analytic expression of the transfert function of a mean filter? Is it an ideal low pass filter?
7. Bonus question: perform the same study for the Gaussian kernel. Determine σ in order to have filter of size 3×3, 5×5, and 7×7.

[1]:
```
def gaussianKernel(sigma):
    """ double -> Array
        return a gaussian kernel of standard deviation sigma
    """
    n2 = int(np.ceil(3*sigma))
    x,y = np.meshgrid(np.arange(-n2,n2+1),np.arange(-n2,n2+1))
    kern = np.exp(-(x**2+y**2)/(2*sigma*sigma))
    return  kern/kern.sum()
### your answer start below
```

Exercice 3: anti aliasing filtering

1. Give a code that subsamples of factor 2 (use function `subSample2()` given in TME3) the image `img/barbara.png`.

[ ]:

2. Give a code that subsamples of factor 2 (use function `subSample2()`) the image `img/barbara.png` after applying an low pass filter (use `antiAliasingFilter()`). As comment, recall the principle of filtering in the frequency domain.

[ ]:
```
def antiAliasingFilter(n,m):
    """ int*int -> Array """
    n2, m2 = n//2, m//2
    rn, rm = n//4, m//4
    A = np.zeros((n, m))
    A[rn:rn+n2, rm:rm+m2] = 1
    return A
### your answer start below
```

3. Describe and analyze the filtering of Barbara with and without the anti aliasing filter. What information is lost for the two filtered images ?

[ ]:

Exercice 4: color image

1. Read images `img/clown.bmp` and `img/clown_lumi.bmp` as two arrays named I_1 and I_2. Display these images examine their shape. What difference there are between them?

[ ]:

2. The first image is an array of dimension 3. Explain the signification of each dimension. From this image create 3 images I_R, I_G, I_B of dimension 2. Display these three images and explain what you see.

[ ]:

3. Create a new image I_3 of dimensions 3, the first dimension contains the value of I_R, the second the value of I_B and the third the values of I_G. Try another combinations. Remark: color images are better handled by `imshow()` if pixel values range in $[0, 1]$.

[ ]:

4. Write a code that allows the see the first channel with red color scales, the second channel in green color scales, and the blue channel in blue color scales.

[ ]:

Practical work 5: edge detection

The goal of this practial work is to experiment various edge detectors. Attention is given to the following points: 1. comparison between the first and second order detectors 2. study of the impact of smoothing 3. removing non maxima answers of the detectors 4. evaluation in term of robustness and localization

[9]:
```
# Useful modules
import numpy as np
from PIL import Image
from scipy.signal import convolve2d
import matplotlib.pyplot as plt

# Useful functions for this work
def orientation(Ix, Iy, Ig):
    """ Array[n,m]**3 -> Array[n,m]
        Returns an image of orientation.
    """
    n, m = Ix.shape
    x = np.arange(4)*np.pi/4
    ori = np.stack((np.cos(x), np.sin(x)), axis=1)
    O = np.zeros(Ix.shape)
    for i in range(n):
        for j in range(m):
            if Ig[i, j] > 0:
                v = np.array([Ix[i, j], -Iy[i, j]])/Ig[i, j]
                if Iy[i, j] > 0: v = -v
                prod = np.matmul(ori, v)
                maxi = prod.max()
                imax = np.nonzero(prod == maxi)
                O[i, j] = imax[0][0]+1
    return O

def gaussianKernel(sigma):
    """ double -> Array
        return a gaussian kernel of standard deviation sigma
    """
    n2 = int(np.ceil(3*sigma))
    x,y = np.meshgrid(np.arange(-n2,n2+1),np.arange(-n2,n2+1))
    kern =  np.exp(-(x**2+y**2)/(2*sigma*sigma))
    return kern/kern.sum()


def imshow(I, title=None, size=500, axis=False):
    """ display an image, with title, size, and axis """
    plt.figure(figsize=(size//80, size//80))
    plt.gray()
    plt.imshow(I)
```

```
    if not axis: plt.axis('off')
    if title: plt.title(title)
    plt.show()

def imshow_hot(I, title=None, size=500, axis=False):
    """ display an image, with title, size, and axis """
    plt.figure(figsize=(size//80, size//80))
    plt.hot()
    plt.imshow(I)
    if not axis: plt.axis('off')
    if title: plt.title(title)
    plt.show()

def niceDisplay14(affichages,titres=None):
    """ list[Array]*list[str] -> NoneType
        display from 1 up to 4 images or vectors with optionnal titles
        2D arrays are displayed as image with imshow()
        1D arrays are displayed as curve with plot()
    """
    if not type(affichages) == type([]):
        affichages = [affichages]

    if titres is None:
        titres = ['',]*len(affichages)

    if not type(titres) == type([]):
        titres = [titres]

    nb_affichages = len(affichages)
    if nb_affichages >4 or nb_affichages < 1 :
        raise ValueError('niceDisplay_14 : affichage should be a list of length␣
↪1 up to 4')

    if nb_affichages != len(titres):
        raise ValueError('niceDisplay_14 : titres must have same length than␣
↪affichage')

    courbes = False
    for i in range(0,nb_affichages):
        s = plt.subplot(101+10*nb_affichages+i)
        s.set_title(titres[i])
        if len(affichages[i].shape)==2 and affichages[i].shape[0] > 1 and␣
↪affichages[i].shape[1] > 1:
            # on affiche une image
            s.imshow(affichages[i], cmap="gray",interpolation='nearest',␣
↪aspect='equal')
        else :
```

```
            # il s'agit d'une seule ligne, à afficher comme une courbe
            plt.plot(affichages[i])
            courbes=True

    agrandissement_h = nb_affichages
    agrandissement_v = nb_affichages*2 if courbes else nb_affichages
    params = plt.gcf()
    plSize = params.get_size_inches()
    params.set_size_inches( (plSize[0]*agrandissement_v,␣
↪plSize[1]*agrandissement_h) )
    plt.show()
```

Exercice 1: comparison between first and second order detectors

The function `conv2()`, written in TME4, is a little bit slow. We propose to use `convolve2d()` available in module `scipy.signal`. We use this fonction with parameter `mode` set to `'same'` (see `help(convolve2d)`).

1. **Sobel filters** are a couple of filters approximating, by finite difference, the gradient of an image, and defined by:

$$S_x = \begin{pmatrix} 1 & 0 & -1 \\ 2 & 0 & -2 \\ 1 & 0 & -1 \end{pmatrix} \qquad S_y = \begin{pmatrix} 1 & 2 & 1 \\ 0 & 0 & 0 \\ -1 & -2 & -1 \end{pmatrix}$$

   Write a function `SobelDetector(I,s)` that computes gradient of I, the norm of the gradient, and returns the norm threshold with a value s.

[5]:
```
def SobelDetector(I, s):
    """ Array*double -> Array """
```

2. **Laplacian filter** approximates, by finite difference, the seconde derivatives of an image. It is defined by:

$$L = \begin{pmatrix} 0 & 1 & 0 \\ 1 & -4 & 1 \\ 0 & 1 & 0 \end{pmatrix}$$

   Write a function `LaplacianDetector(I,s)` that computes the Laplacian of I and returns the zero crossings of the Laplacian as an image. Zero crossing occurs if the sign of Laplacian changes in a neighborrow. To identify a zero crossing for a pixel (i, j):

- consider a 3×3 neighborrow I_L and compute the minimal and maximal values over I_L,
- zero crossing occurs if $\max(I_L) > 0$, $\min(I_L) < 0$, and $\max(I_L) - \min(I_L) > s$

[ ]:
```
def LaplacianDetector(I, s):
    """ Array*double -> Array """
```

3. **Comparison between first and second order detectors**. Experiment the two detectors on `img/lena.png`. Find the thresholds for both detectors (for example, 70 and 70) that allow to detect approximately the same edges.

[ ]:

Exercice 2: Non maximum suppression

This exercice addresses the problem of a unique localization of edges from a map determined by first order filters. This can be obtained by applying a non maximum suppression technique: the norm of an edge gradient should be maxima in the direction of the gradient. To this end, we provide the function `orientation(Ix,Iy,Ig)` that determines from spatial derivatives I_x, I_y (using your favorite filters) and the gradient norm I_g the orientation of the gradient for each pixels. Possible orientations are 0, $\frac{\pi}{4}$, $\frac{\pi}{2}$ and $\frac{3\pi}{4}$ respectively coded by values 1, 2, 3 and 4 (0 stands for null gradient). Orientation of gradient are determined in the direct orthonormal basis.

1. Display the output of function `orientation()` applied on image `img/tools.png`. Verify values according to the different local orientation configurations (use `imshow_hot()` in interactive mode in the nootebook to see pixels value pointed by the mouse screen).

[ ]:
```
%matplotlib notebook
#### your answer below
```

Explain the different values given to a pixel by `orientation()`:

Your answer: ...

2. Write a function `G=nms(Ig, Ior)` getting the gradient norm of an image, and the gradient orientation and returning an image of norm gradient for pixels being a local maxima of gradient norm in the direction of gradient or 0 otherwise. In other words, pixel `(i,j)` is a local maxima if `G[i,j] > 0` and `G[i,j]` gives the gradient norm value at this pixel.

[ ]:
```
def nms(Ig, Ior):
    """ Array**2 -> Array """
```

3. Experiments function `nms()` on images `img/tools.png` and `img/lena.png` after apply or not a gaussian filter (use the function `gaussianKernel()` given in the previous TME).

[ ]:
```
%matplotlib inline
### your answer below
```

4. Conclude on the effect of smoothing (value of σ) on the edge detection process and on the size of edges.

[ ]:

Exercice 3: Effects of smoothing in edge detection

Edge detectors are high pass filters and then amplify noise. To avoid this issue, we apply low pass filter, such as Gaussian filter, as preprocess. In this exercice, we use the image `img/lena.png`.

1. Smooth image `lena` by convolving with G_σ, the gaussian kernel of standard deviation $\sigma = 2$.

[ ]:

2. On the smoothed image apply Sobel and Laplacian edge detectors. Find threshold values (around 10 for Laplacian and 200 for Sobel) in order the two detectors approximately detect the same contours. After smoothing, what are the main difference between the two detectors.

[ ]:

3. Let vary $\sigma \in [\sigma_1, \sigma_2]$ (σ_1 and σ_2 to be determined) and analyze results obtained for both detectors. What are the effects of smoothing on the noise ? on the localisation of contours ?

[ ]:

4. Replace the gaussian filter by a constrast enhancer filter of impulse response $\begin{pmatrix} 0 & -1 & 0 \\ -1 & 5 & -1 \\ 0 & -1 & 0 \end{pmatrix}$ (other choices are possible). Discuss the results.

[ ]:

5. Apply a multiresolution decomposition up to a size 8×8. Apply the two detectors for each resolution. Discuss the results.

[ ]: