Image Filtering and Fourier Transforms*

November 20, 2019

1 Image Loading and Displaying

To load and visualize images, the toolboxes provide direct commands.

```
n = 256; % Size of image
M = load_image('lena',n);

figure
imageplot(M)
```

An image is just a matrix $f \in \mathbb{R}^N$ of $N = N_0 \times N_0$ pixels and each element of the matrix denotes the *"intensity"* of that pixel.

To implement subplots within the `imageplot` command is as easy as:

```
figure
imageplot(M(1:50,1:50),'Zoom',1,2,1)
imageplot(-M,'Reversed contrast',1,2,2)
```

1.1 From Pixels to Operations of Groups of Pixels

Suppose we have an image represented as some matrix $f \in \mathbb{R}^N$, each point x_{i_1,i_2} denotes the intensity. Now we want to move from operations with one single pixel to operations with several pixels around. Some applications might be for blurring, smoothing edges...

How to smooth a signal? A simple way of smoothing a given $1-$D signal is to take the average of neighbouring values, i.e. a moving average or a weighted moving average process.

- **Moving Average:** take the N (odd) neighbouring values of a given x_j with equal weights and divide by N, thus each smoothed value

$$\tilde{x}_j = \frac{1}{N} \sum_{n=-\frac{(N-1)}{2}}^{\frac{(N-1)}{2}} x_{j+n}$$

*These are notes based on Gabriel Peyré's exceptional Numerical Tours and are for my own personal study.

- **Weighted Moving Average:** same as before but weighting each point with a given weight ω_n.

$$\tilde{x}_j = \frac{1}{N} \sum_{n=-\frac{(N-1)}{2}}^{\frac{(N-1)}{2}} \omega_n x_{j+n} \; ; \; \sum_{n=-\frac{(N-1)}{2}}^{\frac{(N-1)}{2}} \omega_n = 1$$

For an image, which would be a $2-D$ matrix, we would want to take the matrix and transform each pixel using its neighbouring values. Suppose we start with a given matrix $f \in \mathbb{R}^{9\times 9}$, i.e. a 9×9 matrix and we want to perform operations on subsets of this matrix. In this example, we want to make an operation over a 3×3 subset of the matrix with the initial pixel in the center of the matrix, and generate a new value to place it in the smoothed image $\tilde{f}$ at that precise location. That is, suppose we start with the original matrix $f \in \mathbb{R}^{9\times 9}$ and select the element $x_{2,2}$. The 3×3 subset that corresponds to the element $x_{2,2}$ of f will be the sub-matrix $K^{2,2}$

$$K^{2,2} = \begin{pmatrix} x_{1,1} & x_{1,2} & x_{1,3} \\ x_{2,1} & x_{2,2} & x_{2,3} \\ x_{3,1} & x_{3,2} & x_{3,3} \end{pmatrix}$$

We want to perform an operation over $K^{2,2}$ so that we get a corresponding smoothed value $\tilde{x}_{2,2}$ in the smoothed image $\tilde{f}$.

One possibility is to just average over the values of $K^{i,j}$, in this case, the operation would be $(1/9)\sum_{j=1}^{N} x_j$ where $x_j \in K^{i,j}$.

In Figure 1 we start with a matrix with many pixels in black, and some center pixels in white. By the process of averaging neighbours, we get the smoothed image with different shades of gray.

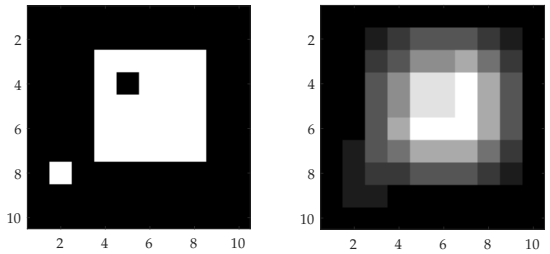


Figure 1: Smoothed Image via Neighbor Averaging

This specific step-by-step process is generalized by thinking about *kernels*.

Definition 1 *A **kernel, convolution matrix,** or **mask** is a (usually small) matrix used to tweak in some way an image. The way in which the kernel can be applied to an image is through a convolution or a cross-correlation.*

Basically, a kernel $K(i,j)$ is applied to an image pixel by pixel. Important elements are the size of the kernel, which depends on the number of neighbours k (in the example, $k = 1$), and the window size, which is equal to $2k + 1$. The number of neighbours is k since we move k elements in each direction from the central pixel. With $k = 1$, we get a kernel of size 3×3, given by the window size.

Blurring Blurring is achieved by computing a convolution $(f * g)$ with a kernel g.

A **convolution** is a mathematical operation on two functions (f, g) that produces a third function expressing how the shape of one is modified by the other. The term convolution refers to both the result function and to the process of computing it. It is defined as the integral of the product of the two functions after one is reversed and shifted.

Definition 2 *If f and g are two continuous functions, the convolution of f and g denoted $(f * g)$, is a particular kind of integral transform:*

$$(f * g) \triangleq \int_{-\infty}^{\infty} f(\tau)g(t - \tau)d\tau$$

If f and g are two discrete functions, the convolution of f and g is defined as:

$$(f * g) \triangleq \sum_{\tau=-\infty}^{+\infty} f(\tau)g(x - \tau)$$

Function g is usually called the input, and f the kernel of the convolution.

Intuitively, the convolution of two functions represents the amount of overlap between the two functions. In image processing, what we call convolution is related to the previous definition and to the product of matrices. A function of two variables $f(x, y)$ can be regarded as a matrix $A_{x,y}$, thus, we can define convolution of matrices as in Definition 3.

Definition 3 *Given two functions f and g represented as the $n \times m$ matrix A and the $k \times \ell$ matrix B, then $(f * g)$ is an $(n + k - 1) \times (m + \ell - 1)$ matrix C in which each element is defined as:*

$$c_{x,y} = \sum_u \sum_v a_{u,v} b_{(x-u+1),(y-v+1)}$$

where u and v range over all legal subscripts for a_{uv} and $b_{(x-u+1),(y-v+1)}$.

Suppose we want to construct a kernel with k neighbours and $(2k + 1)$ window size that smoothes our image F by taking averages. Let G denote the smoothed image resulting from applying a kernel K, and $G_{i,j}$ the $ij-$th element of the output image.This element can be computed as

$$G_{i,j} = \frac{1}{(2k+1)^2} \sum_{u=-k}^{k} \sum_{v=-k}^{k} F_{i-u,j-v}$$

where the term $\frac{1}{(2k+1)^2}$ allows to normalize the elements of the resulting image and is, in fact, the weight each element receives. Note, that in this specific example, each element receives the same weight. If we wanted to include different weights to different elements, these would be given by the kernel. If we let $K_{i,j}$ denote the $ij-$th element of the kernel, the $ij-$th element of the convolution $G = (F * K)$ would be computed as

$$G_{i,j} = \sum_{u=-k}^{k} \sum_{v=-k}^{k} K_{u,v} F_{i-u,j-v}$$

where $K_{u,v}$ provides the non-uniform weights.

The following snippet of code produces Figure 2. The figure shows the effects of smoothing the image with a kernel of size 9×9 in which each element is equal to $1/81$. Both functions `perform_convolution` and `conv2` yield similar results.

```
k = 9; % size of the kernel (the larger, the more blurred)
g = ones(k,k);
g = g/sum(g(:)); % normalize

fg = perform_convolution(f,g);
fg2 = conv2(f,g,'same');

% Blurred image
figure
imageplot(fg,'Blurred (Peyre)',1,3,1)
imageplot(fg2,'Blurred (conv2)',1,3,2)
imageplot(f,'Original',1,3,3)
```

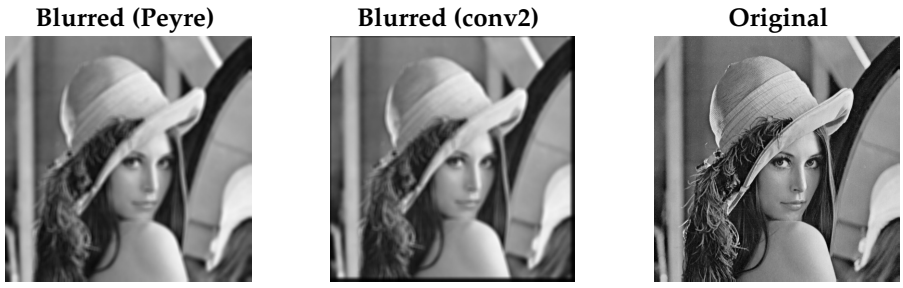


Figure 2: Blurring by use of Convolution

Median Filtering A special case of filtering is the median filtering. We could take the same sub-matrix as with the average filtering case but stack its elements as a vector and compute the median of that resulting vector. As we did for the averaging filter, we could apply this filter to all $n \times n$ regions of our initial matrix. Why would we use median filtering?

- It is a **non-linear** operation.
- Reduces noise.
- Preserves edges (sharp lines)
- Main idea: use median of all pixels in a kernel instead of the mean.

Figure 3 shows the effects of applying an 11×11 median filter. Note how the edges are preserved.

Convolution vs Cross-Correlation A similar procedure to convolution is cross-correlation. The cross correlation $G = (F \otimes K)$ for a kernel K and an image F is defined as

$$G_{i,j} = \sum_{u=-k}^{k} \sum_{v=-k}^{k} K_{u,v} F_{i+u,j+v}$$

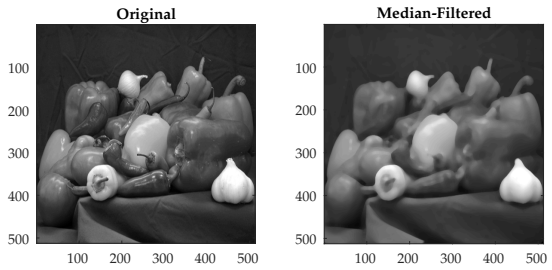


Figure 3: Example of a Median Filtered Image

The main difference with a convolution is the indices of F, recall that for the convolution, the way we index is $F_{i-u,j-v}$. Suppose we take a 3×3 kernel in which the central pixel is the origin, thus its index corresponds to $(0,0)$. Since it is 3×3, the number of neighbours we are taking is $k = 1$. Let us take a general kernel K and apply it to an impulse image F which is of size 7×7. An impulse image is one in which all elements are 0 except for the element $(4,4)$ that is 1.

$$K = \begin{pmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{pmatrix} \; ; F = \begin{pmatrix} 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{pmatrix}$$

As an example, take the $(4,3)$ element of the cross-correlation between F and K. This is equal to 6 since all elements are 0 except $F_{4,4}$. But the step by step procedure is:

$$(F \otimes K)_{4,3} = 7 \times F_{3,2} + 8 \times F_{3,3} + 9 \times F_{3,4} + 4 \times F_{4,2} + 5 \times F_{4,3} + 6 \times F_{4,4} + 1 \times F_{5,2} + 2 \times F_{5,3} + 3 \times F_{5,4} = 6$$

Instead, the convolution, would still have all elements equal to 0 except for $F_{4,4}$ but the coefficient that goes with $F_{4,4}$ in the convolution is not 6 but 4. The resulting matrices for the convolution and the cross-correlation are shown below.

$$(F * K) = \begin{pmatrix} 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 2 & 3 & 0 & 0 \\ 0 & 0 & 4 & 5 & 6 & 0 & 0 \\ 0 & 0 & 7 & 8 & 9 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{pmatrix} \; ; \; (F \otimes K) = \begin{pmatrix} 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 9 & 8 & 7 & 0 & 0 \\ 0 & 0 & 6 & 5 & 4 & 0 & 0 \\ 0 & 0 & 3 & 2 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{pmatrix}$$

Let us visualize what we have just done. Starting with the impulse image and applying that kernel to it would result in Figure 4.

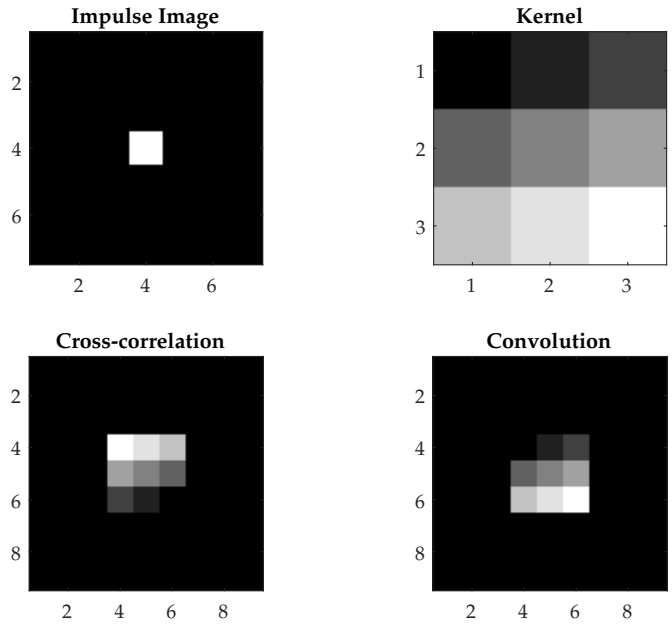


Figure 4: Impulse image, kernel, cross-correlation, and convolution

Properties of Convolution Some properties of convolution:

- **Linear and shift invariants:** behaves the same everywhere. The value of the output depends on the patttern in the image neighborhood, not the position of the neighborhood.
- **Commutative:** $(F * K) = (K * F)$.
- **Associative:** $(F * K) * G = F * (K * G)$
- **Identity:** unit impulse. $E = [\ldots, 0, 0, 1, 0, 0, \ldots]$, then $(F * E) = F$.
- **Separable:** if the filter is separable, we can convolve all rows and then all columns separately.

1.2 Fourier Transform

To measure error between an image f and its approximation f_M, we use the signal-to-noise ratio (SNR) measure

Definition 4 *The signal to noise ratio (SNR) is defined as*

$$SNR(f, f_M) = -20 \log_{10} \left(\frac{\|f - f_M\|}{\|f\|} \right)$$

which is a quantity expressed in decibels (dB). The higher the SNR, the better the quality.

Definition 5 *A basis B of a vector space V over a field F (such as the real numbers $\mathbb{R}$ or the complex numbers $\mathbb{C}$) is a linearly independent subset of V that spans V. This means that a subset B of V is a basis if it satisfies the two following conditions:*

- ***Linear independence property:*** *for every finite subset* $\{b_1, \ldots, b_n\}$ *of* B *and every* $a_1, \ldots, a_n$ *in* F*, if* $a_1 b_1 + \cdots + a_n b_n = 0$*, then necessarily* $a_1 = \cdots = a_n = 0$*;*
- ***Spanning property:*** *for every (vector)* $v \in V$*, it is possible to choose* $v_1, \ldots, v_n$ *in* F *and* $b_1, \ldots, b_n$ *in* B *such that* $v = v_1 b_1 + \cdots + v_n b_n$*.*

Definition 6 *Any n orthogonal vectors which are of unit length*

$$\langle u_i, u_j \rangle = \begin{cases} 1 & \text{if } i = j \\ 0 & \text{otherwise} \end{cases}$$

form an orthonormal basis of $\mathbb{R}^n$

Definition 7 *The Fourier orthonormal basis is defined as*

$$\psi_m(k) = \frac{1}{\sqrt{N}} e^{\frac{2i\pi}{N_0}\langle m,k \rangle}$$

where $0 \leq k_1, k_2 < N_0$ *are position indexes, and* $0 \leq m_1, m_2 < N_0$ *are frequency indexes.*

The Fourier transform $\hat{f}$ is the projection of the image on this Fourier basis.

$$\hat{f}(m) = \langle f, \psi_m \rangle.$$

The Fourier transform is computed in $O(Nlog(N))$ operation using the FFT algorithm (Fast Fourier Transform). Note the normalization by $N = \sqrt{N_0}$ to make the transform orthonormal.

The following snippet of code shows the Fourier transform of the original image, conservation of energy, and the log of the Fourier magnitude $\left(\log\left(\left\|\hat{f}(m)\right\| + \varepsilon\right)\right)$. The function `fftshift` shifts the zero-frequency component to the center of the array.

```
% Normalized Fast Fourier Transform
F = fft2(f)/n0;

% Check conservation of the image
fprintf('Energy of image: %5.5f \n',norm(f(:)))
fprintf('Energy of Fourier: %5.5f \n',norm(F(:)))

% Compute the log¡ of the Fourier magnitude for some small epsilon
epsi = 1e-2;

% Shift the zero frequency component to the center of the array
L = fftshift(log(abs(F)+1e-1));

% Display
clf;
imageplot(L,'Log(Fourier Transform)')
```

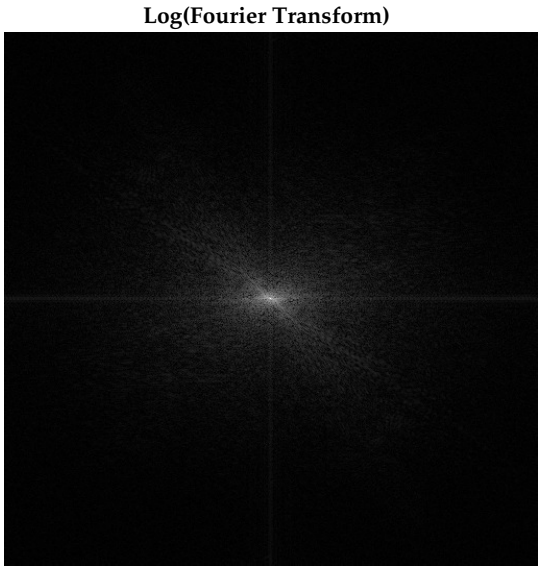


Figure 5: Log of Fourier Magnitude

1.3 Linear Fourier Approximation

An approximation is obtained by retaining a certain set of index I_M

$$f_M = \sum_{m \in I_M} \langle f, \psi_m \rangle \psi_m.$$

A linear approximation is obtained by retaining a **fixed** set I_M of $M = |I_M|$ coefficients. The important point is that I_M does not depend on the image f to be approximated.

For the Fourier transform, a low pass linear approximation is obtained by keeping only the frequencies within a square.

$$I_M = \{m = (m_1, m_2), -q/2 \leq m_1, m_2 < q/2\}$$

where $q = \sqrt{M}$.

This can be achieved by computing the Fourier transform, setting to zero the $N - M$ coefficients outside the square I_M and then inverting the Fourier transform.

Example 1 *Perform the linear Fourier approximation with M coefficients. Store the result in the variable* `fM` *and display.*

```
% Number of kept coefficients
M = n0^2/64;
% Bound of interval
q = sqrt(M);

% Compute Centered Fourier transform
F = fft2(f);

% Linear approximation pre-allocation
```

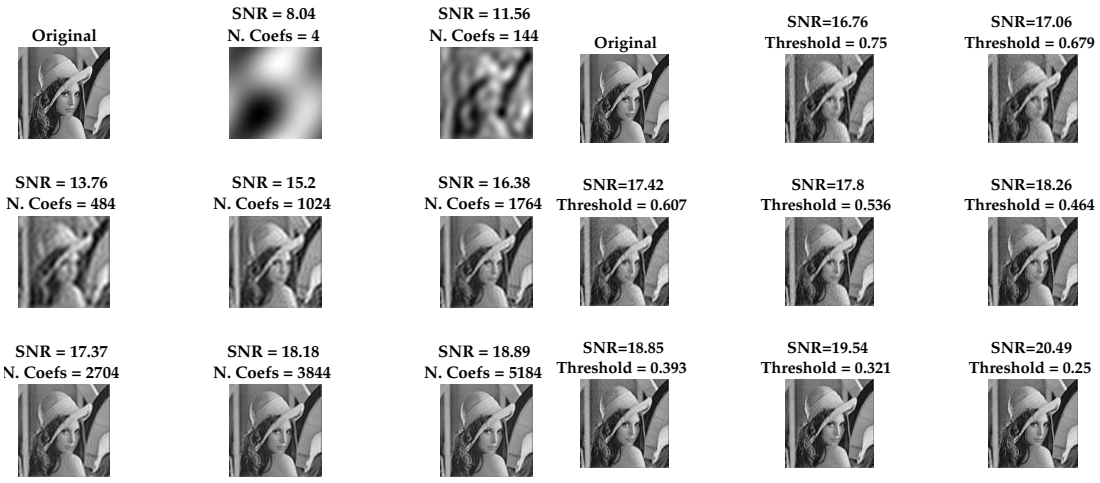
```
F1 = zeros(n0,n0);

% Choose a square in the middle of the image
sel = (n0/2-q/2:n0/2+q/2) + 1;

% Take the points of the square to the linear approx
F1(sel,sel) = F(sel,sel);

% Invert the Fourier and keep real terms
fM = real(ifft2(F1));

% Plot
figure
imageplot(f,'Original',1,2,1)
imageplot(F1,['Linear Fourier Approximation with ',num2str(snr(f,fM),4)])
```

1.4 Non-Linear Fourier Transform

A non-linear approximation is obtained by taking the M largest coefficients. This is equivalently computed using a threshold for the coefficients.

$$I_M = \{m, |\langle f, \psi_m \rangle| > T\}$$

Figures 6a and 6b show the linear and non-linear approximations for a different set of parameter values.



(a) Linear

(b) Non-linear

Figure 6: Fourier Approximations for Different Parameter Values