Image Analysis: An Introduction to Super Resolution using Wavelet

Guillaume Lemaître
Heriot-Watt University, Universitat de Girona, Université de Bourgogne
g.lemaitre58@gmail.com

***Abstract*—Principle of super resolution is to obtain high resolution images from one or several low resolution images. First methods allows to recover high resolution images using several low resolution images. However, a challange was to reconstruct these high resolution images from only one low resolution image. In order to recover the missing data, the idea is to interpolate vanished data. In this paper, different methods are presented based on wavelet interpolation. An histotical overview will be given so that the reader catch the scientific evolution way taken by the researchers to face the super resolution issues.**

I. Introduction

Super resolution refers to techniques which allow to enhance the resolution of images. Before to go further, super resolution methods presented in this literature review will be only based on wavelet theory [2], [8], [10] and more precisely on wavelet interpolation.

In order to solve the super resolution issues which is an inverse problem, a foward model have to be constructed. In the different papers which will be described, the foward model used is the following:

$$\mathbf{f}_k = DCE_k\mathbf{x} + \mathbf{n}_f \tag{1}$$

where D is the downsampling operator, C is the blurring operator, E_k is the affine transforms to acquire each low resolution image and $\mathbf{n}_f$ is an additive noise. $\mathbf{x}$ is the high resolution unknown image while $\mathbf{f}_k$ are the different low resolution images. Figure 1 presents the different steps of the forward model to go from high resolution image to low resolution images.

C. Ford and D. M. Etter proposed a method to interpolate missing values for one dimensional signal [1]. Inspired by this work, N. Nguyen and P. Milanfar proposed an extension for interlaced one dimensial signal and two dimensional images [3], [5], [4]. These methods can be considered as conventional methods since the super resolution image is computed giving multiple low resolution images. Each low resolution image imposes a set of linear constraints on the unknown high resolution image. If the number of low resolution images is enough (where each image gives different information due to the subpixel shifts), the set of equations will be larger than the number of unknowns and the system is determined.

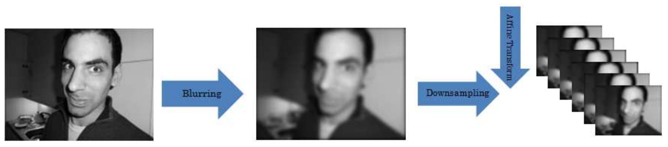


Figure 1. Forward transformation to go from high resolution image to low resolution images

Instead of obtain super resolution image from several low resolution images, a challenge was to generate super resolution image from only with one low resolution image. Previous methods [1], [3], [5], [4] will lead to an under determined system because the number of unknowns will be larger than the number of linear constraints given by the low resolution image. In order to get round these difficulties of under determined system affected by the restricted number of low resolution image, D. Glasner et al. proposed a trick based on the property of patch redundancy inside a single image [13].

Recently, due of several important results by D. Donoho, E. Candes, J. Romberg and T. Tao [15], [16], [17], [18], [19] in the field of compressed sensing, under determined linear system could be solved and give sparse solution. The wavelet domain being almost sparse, these results could be used to infer super resolution issues. J. Yang and al. proposed a method using the sparse representation of the wavelet domain in order to generate a super resolution image from only one low resolution image [6], [7].

In this paper, previous presented works will be discussed in details. Super resolution from multiple low resolution images will be presented in the first section. The advance using only a single frame will be introduced in the second section while a presentation of super resolution using compressed sensing and sparse representation properties will be given in the third part. The last section will be dedicated to the different applications of super resolution.

II. Super resolution from multiple low resolution images

This part is organized as follow: the first section will be dedicated to the one dimensional case and the second section will describe the extension of the two dimensional case.

A. One dimensional case

The method presented in this section was proposed by N. Nguyen and P. Milanfar [3]. This work is based on the

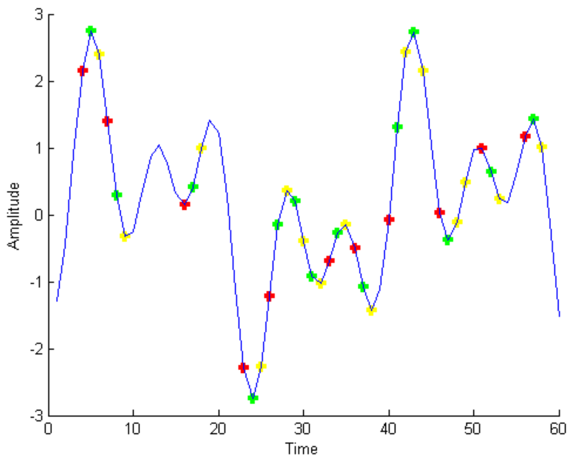


Figure 2. Example of interlaced data. Points in red can be considered as a frame as well as points in green and in yellow. The signal in blue is the original signal.

multiresolutional basis fitting reconstruction (MBFR) method from C. Ford and D. M. Etter [1]. The hypothesis assumed by N. Nguyen and P. Milanfar was to consider the data organized as interlaced sampling structure which will simplify the complexity of the algorithm. For a one dimensional case, interlaced sampling can be represented as in figure 2.

Data are not only sampling randomly but can be represented as a set of ”frames”. The transformation from one frame to another is a simple translation. Hence, each frame gives information (imposes linear constraints). The principle of super resolution will be to find the missing data so that after reconstruction, the original signal will be recovered.

In order to retrieve the ”high resolution” signal, wavelet coefficients have to be computed. Projecting the function onto a sufficient number of subspaces will lead to have a sufficient number of wavelet coefficients in order to reconstruct the original signal at every points wanted in the finest scale. Approximation coefficients will lead to obtain a coarse-scale approximation of the original signal while detail coefficients will allow to find out the details missing after the calculation of the coarse-scale approximation.

The approximations coefficients extract from multiple low resolution ”frames” are approximately equal to the approximation coefficients of the high resolution signal. Finding these approximation coefficients, the details coefficients can be retrieve.

1) Overview of multiresolution analysis and discrete wavelet transform: Before to enter into details in the MBFR method, an overview of the multiresolution analysis and discrete wavelet transform will be given. A multiresolution analysis in $L^2(\mathbb{R})$ is defined by [8]:

$$... \subset V_2 \subset V_1 \subset V_0 \subset V_{-1} \subset ... \subset V_{j+1} \subset V_j \subset ...$$

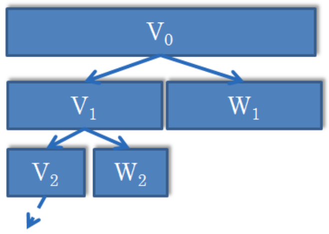


Figure 3. Scheme of multiresolution analysis in discrete wavelet transform

$$\begin{aligned}\overline{\bigcup_{j\in\mathbb{Z}} V_j} &= L^2(\mathbb{R}) &(2)\\ \bigcap_{j\in\mathbb{Z} V_j} &= \{0\} &(3)\\ \forall j\in\mathbb{Z} \textit{ if } f(x)\in V_j &\Leftrightarrow f(2^{-1}x)\in V_{j+1} &(4)\\ \forall k\in\mathbb{Z} \textit{ if } f(x)\in V_0 &\Leftrightarrow f(x-k)\in V_0 &(5)\end{aligned}$$

Moreover, W_j can be defined as the orthogonal complement of V_j as:

$$\begin{aligned}V_{j-1} &= V_j \oplus W_j &(6)\\ L^2(\mathbb{R}) &= \bigoplus_{j\in\mathbb{Z}} W_j &(7)\end{aligned}$$

The multiresolution analysis can be considered as shown in figure 3. The space V_j is the approximation space spanned by the scaling function $\varphi_{j,n}(x) = 2^{-\frac{j}{2}}\varphi(2^{-j}x - n)$ while the space W_j is the detail space spanned by the scaling function $\psi_{j,n}(x) = 2^{-\frac{j}{2}}\psi(2^{-j}x - n)$ [1].

Considering this multiresolution analysis, if a function $f(x)$ resides in the space V_0, this function can be decomposed as follow:

$$f(x) = \sum_n a_{J,n}\varphi_{J,n}(x) + \sum_{j=1}^{J}\sum_n d_{j,n}\psi_{j,n}(x) \qquad (8)$$

2) Computation of the coarse-scale approximation of the signal: As remind in the section II-A1, a function can be decomposed as shown in equation (8) for all x. In the case where data are missing, only few samples $x_k \in \{0, 1, 2, ..., p\}$ are available. Hence the equation (8) can be written as:

$$\begin{aligned}
f(x_0) &= \sum_n a_{J,n}\varphi_{J,n}(x_0) + \sum_{j=1}^{J}\sum_n d_{j,n}\psi_{j,n}(x_0) \\
f(x_1) &= \sum_n a_{J,n}\varphi_{J,n}(x_1) + \sum_{j=1}^{J}\sum_n d_{j,n}\psi_{j,n}(x_1) \\
f(x_2) &= \sum_n a_{J,n}\varphi_{J,n}(x_2) + \sum_{j=1}^{J}\sum_n d_{j,n}\psi_{j,n}(x_2) \quad (9) \\
f(x_p) &= \sum_n a_{J,n}\varphi_{J,n}(x_p) + \sum_{j=1}^{J}\sum_n d_{j,n}\psi_{j,n}(x_p)
\end{aligned}$$

This last decomposition was the base of the work describe in [1]. However, N. Nguyen and P. Milanfar included the concept of interlaced data [3] presented on figure 2. Hence, the data available are $x_k \in \{0, 1, 2, ..., p\}$ but n times where n will be the number of "frames". So equation (10) can be written:

$$\begin{aligned}
f^i(x_0) &= \sum_n a_{J,n}\varphi^i_{J,n}(x_0) + \sum_{j=1}^{J}\sum_n d_{j,n}\psi^i_{j,n}(x_0) \\
f^i(x_1) &= \sum_n a_{J,n}\varphi^i_{J,n}(x_1) + \sum_{j=1}^{J}\sum_n d_{j,n}\psi^i_{j,n}(x_1) \\
f^i(x_2) &= \sum_n a_{J,n}\varphi^i_{J,n}(x_2) + \sum_{j=1}^{J}\sum_n d_{j,n}\psi^i_{j,n}(x_2) \quad (10) \\
f^i(x_p) &= \sum_n a_{J,n}\varphi^i_{J,n}(x_p) + \sum_{j=1}^{J}\sum_n d_{j,n}\psi^i_{j,n}(x_p)
\end{aligned}$$

where i is the low resolution "frame" considered.

In vector notation, the system is given by:

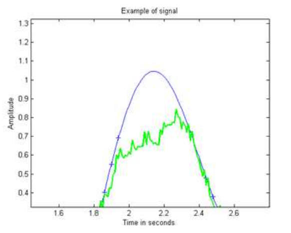
$$\mathbf{f}^i_s = \Phi^{s(i)}_J \mathbf{a}_J + \sum_{j=1}^{J} \Psi^s_{j(i)} \mathbf{d}_j \qquad (11)$$

where Φ^s_J is the matrix of shifts of the scaling function samples at the level J associated with the sample index x_k of the i^{th} frame f^i_s. Ψ^s_j is the matrix of shifts of the wavelet function samples at the level j associated with the sample index x_k of the i^{th} frame f^i_s.

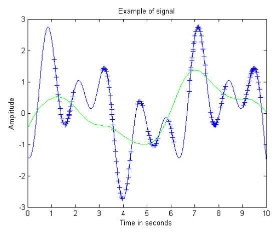
Due of the missing data, the detail coefficients $\mathbf{d}_j$ cannot be computed. However, the approximation coefficients can be approximate discarding the detail part. So the system (11) can be written as follow:

$$\mathbf{f}^i_s \approx \Phi^s_J \mathbf{a}_J \qquad (12)$$

In order to solve this problem in regularized least square sense in the wavelet domain, the number of linear constraints has to be larger than the number of unknowns. So the number of "frames" have to be large enough and as discuss by C. Ford and D. M. Etter, J is chosen to be the minimum (or finest) resolution for which the system is determined [1]. Thus, an approximation of the approximation coefficients is given by:



(a) Case when the value of λ is too small. The solution will be under-regularized and will oscillate.



(b) Case when the value of λ is too large. The solution found will not "trust" enough the samples given and will be too flat.

Figure 4. Properties of the parameter λ

$$\hat{\mathbf{a}}_J = (\Phi^{sT}_J \Phi^s_J + \lambda I)^{-1} \Phi^{sT}_J \mathbf{f}^i \qquad (13)$$

Taking advantage of the interlaced data properties contributed by [3], the equation (13) can be written as:

$$\hat{\mathbf{a}}_J = \left(\sum_{i=1}^{n} \Phi^{s(i)T}_J \Phi^{s(i)}_J + \lambda I \right)^{-1} \sum_{i=1}^{n} \Phi^{s(i)T}_J \mathbf{f}^i \qquad (14)$$

where I is the identity matrix with the same size as $\Phi^{s(i)T}_J \Phi^{s(i)}_J$. The parameter λ is the parameter of regularization. Larger λ is, more regular (flat) will be the solution. The properties of the parameter λ is shown on figure 4. When the value of λ is too small (figure 4(a)), the solution will be under-regularized and will oscillate. When the value of λ is too large (figure 4(b)), the solution found will not "trust" enough the samples given and will be too flat.

Once the approximation coefficients $\hat{\mathbf{a}}_J$ are computed, a coarse-scale approximation of the original signal can be computed:

$$\hat{\mathbf{f}}_0 = \Phi_J \hat{\mathbf{a}}_J \qquad (15)$$

where Φ_J is the matrix of shifts of the scaling function at level J for all x of the high resolution grid. Results of approximation is presented on figure 5.

After have computed the coarse-scale approximation signal, it will be possible to compute the detail coefficients from the coarsest to the finest scale. This calculation is explained in the next section.

3) Computation of the detail part of the signal: The following equation can be deduced from the equation (11):

$$\sum_{j=1}^{J} \Psi^s_j \mathbf{d}_j = \mathbf{f}^i_s - \Phi^s_J \mathbf{a}_J \qquad (16)$$

Replacing $\mathbf{a}_J$ by the approximation computed before $\hat{\mathbf{a}}_J$ and taking profit of the interlaced data sampling, the equation (16) can be written as follow:

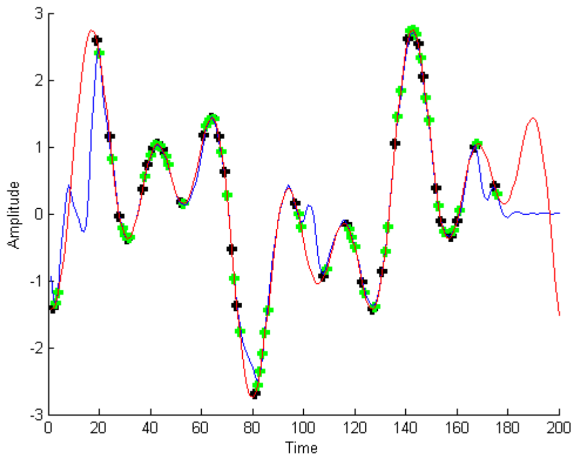


Figure 5. The original signal is represented in red. The coarse-scale approximation found using regularized least square is shown in blue. Black and green dots are the data used to infer the approximation signal. Points having the same color belongs to the same ”frame”.

$$\sum_{j=1}^{J} \Psi_j^s \mathbf{d}_j = \mathbf{f}_s^i - \Phi_J^{s(i)} \hat{\mathbf{a}}_J \tag{17}$$

Hence,

$$\begin{aligned} \Psi_J^s \mathbf{d}_J &\approx \mathbf{f}_s^i - \Phi_J^{s(i)} \hat{\mathbf{a}}_J && (18)\\ \mathbf{e}_0^i &\approx \mathbf{f}_s^i - \Phi_J^{s(i)} \hat{\mathbf{a}}_J && (19) \end{aligned}$$

In the same way as in the previous section II-A2, a solution can be inferred in regularized least square sense and will lead to formalize as follow:

$$\hat{\mathbf{d}}_J = \left(\sum_{i=1}^{n} \Psi_J^{s(i)T} \Psi_J^{s(i)} + \lambda I \right)^{-1} \sum_{i=1}^{n} \Psi_J^{s(i)T} \mathbf{e}_0^i \tag{20}$$

where $\Psi_J^{s(i)}$ is the matrix of shifts of the wavelet function samples at the level J associated with the sample index x_k of the i^{th} error frame e_s^i.

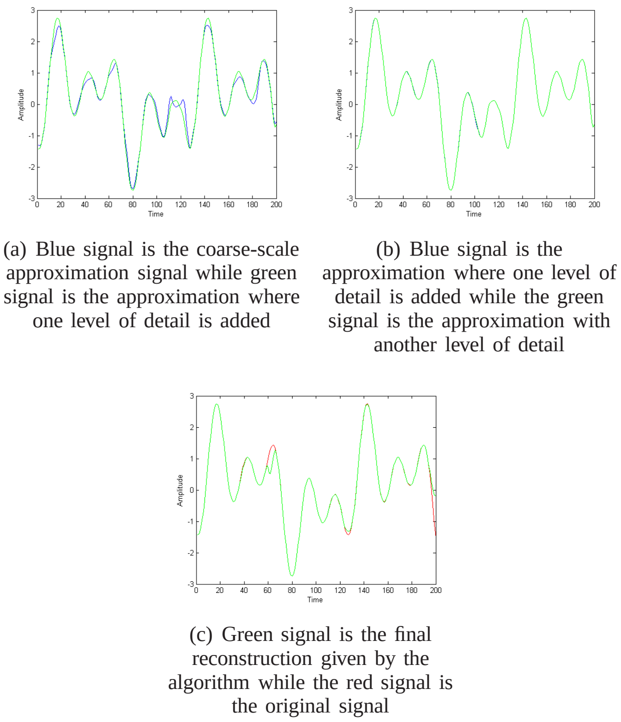
Once the detail coefficients are found, an approximation signal can be update as follow:

$$\hat{\mathbf{f}}_1 = \hat{\mathbf{f}}_0 + \Psi_J \hat{\mathbf{d}}_J \tag{21}$$

Results of the approximation signal are shown on figure 6(a).

Replacing $\hat{\mathbf{f}}_0$ by $\hat{\mathbf{f}}_1$ and repeating the same manipulation at the next finer scale $(J-1)$, it will be possible to add one level of detail more and so on until achieving the finest scale possible. Results of the approximation signal are shown on figure 6(b) while final results are presented on figure 6(c).

A Matlab implementation is given in appendix A.

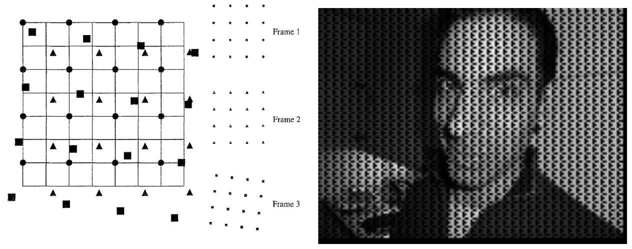


(a) Blue signal is the coarse-scale approximation signal while green signal is the approximation where one level of detail is added

(b) Blue signal is the approximation where one level of detail is added while the green signal is the approximation with another level of detail

(c) Green signal is the final reconstruction given by the algorithm while the red signal is the original signal

Figure 6. Results for different level of details and comparison with the original signal



(a) Interlaced data in the two dimensions case

(b) Example of interlaced data after registration of low resolution images where black pixels will have to be interpolated

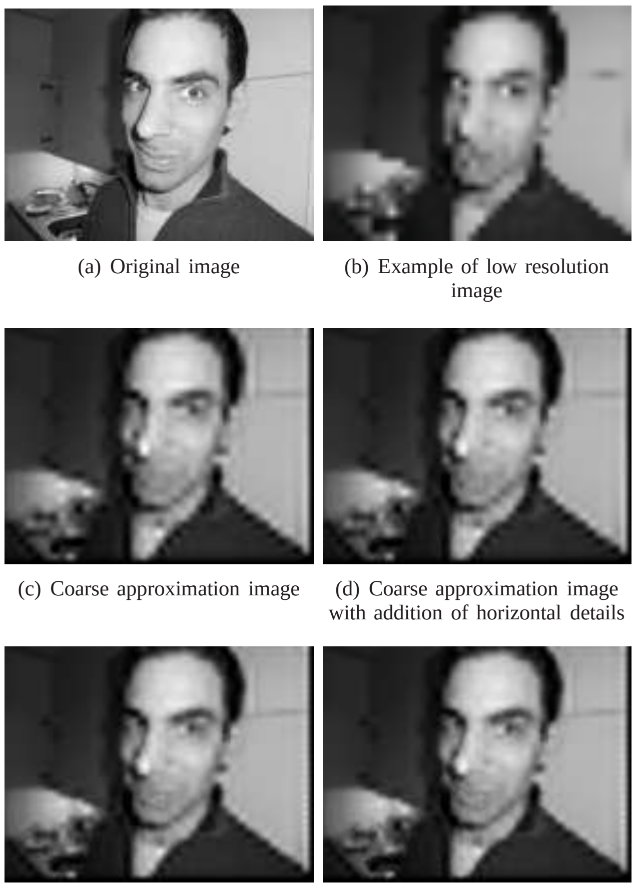
Figure 7. Interlaced data in the two dimensions case

B. Two dimensional case

N. Nguyen and P. Milanfar extend the one dimensional case presented in the previous section (II-A) to the two dimensional case for images [3], [5], [4].

They assumed in the same way that the data given by low resolution images are interlaced sampling as shown in figure 7.

Each low resolution image has to be translated to give enough new information about the scene to create the high resolution image. As previously, the method is based on multiresolutional analysis and discrete wavelet transform. So first, a complement of discrete wavelet analysis for the two dimensional case will be given. Then, the interpolation for interlaced two dimensional images will be introduced.


(a) Original image (b) Example of low resolution image

(c) Coarse approximation image (d) Coarse approximation image with addition of horizontal details

(e) Coarse approximation image with addition of horizontal details and vertical details (f) Coarse approximation image with addition of horizontal details, vertical details and diagonal details

Figure 8. Results obtained with the algorithm described

C. Multiresolution analysis and discrete wavelet transform for two dimensional images

The multiresolution analysis and wavelet transform for two dimensional images are just an extension of the one dimensional case. In this section, only modifications compare to the one dimensional case will be presented.

As presented in [8], [10], [12], the decomposition of an image in the wavelet domain is about the same. The main changes lie in the two dimensions of the wavelet family (scaling and wavelet function). Now, the translation is not only in one dimension but in the two direction (x and y).

Hence, the main change is in the equation (8) which will be now:

$$f(t,s) = \sum_{k,l} a_{J,k,l}\varphi_{J,k,l}(t,s) + \sum_{j=1}^{J}\sum_{k,l} d_{j,k,l}\psi_{j,k,l}(t,s) \quad (22)$$

In the case of wavelet families which are separable, Mallat deduces that the two dimensional space $V_j^{(2)}$ is equivalent to the tensor product of the one dimensional space V_j [2], [10], [12]:

$$V_j^{(2)} = V_j \otimes V_j \quad (23)$$

And the two dimensional scaling function can be decomposed into two one dimensional scaling functions(horizontal and vertical):

$$\varphi_{j,k,l}(t,s) = \varphi_{j,k}(t)\varphi_{j,l}(s) \quad (24)$$

The two dimensional wavelet function will be decomposed into three dimensional wavelet function (horizontal, vertical and diagonal):

$$\begin{array}{rcl} \psi^h_{j,k,l}(t,s) & = & \varphi(s)_{j,k}\psi(t)_{j,l} \\ \psi^v_{j,k,l}(t,s) & = & \psi(s)_{j,k}\varphi(t)_{j,l} \\ \psi^d_{j,k,l}(t,s) & = & \psi(s)_{j,k}\psi(t)_{j,l} \end{array} \quad (25)$$

Thus the equation 22 can be written as:

$$\begin{aligned} f(t,s) = & \sum_{k,l} a_{J,k,l}\varphi_{j,k}(t)\varphi_{j,l}(s) \\ & + \sum_{j=1}^{J}\sum_{k,l} d^h_{j,k,l}\varphi(s)_{j,k}\psi(t)_{j,l} \\ & + \sum_{j=1}^{J}\sum_{k,l} d^v_{j,k,l}\psi(s)_{j,k}\varphi(t)_{j,l} \\ & + \sum_{j=1}^{J}\sum_{k,l} d^d_{j,k,l}\psi(s)_{j,k}\psi(t)_{j,l} \end{aligned} \quad (26)$$

Hence, re-writing the equation (11) for the two dimensional case:

$$\begin{array}{rcl} \mathbf{f}^i_s & = & (\Phi^{s(i)}_{Jt} \otimes \Phi^{s(i)}_{J})\mathbf{a}_J \\ & + & \sum_{j=1}^{J}(\Phi^{s(i)}(s)_{jt} \otimes \Psi^{s(i)}(t)_{js})\mathbf{d}^h_j \\ & + & \sum_{j=1}^{J}(\Psi^{s(i)}(s)_{jt} \otimes \Phi^{s(i)}(t)_{js})\mathbf{d}^v_j \\ & + & \sum_{j=1}^{J}(\Psi^{s(i)}(s)_{jt} \otimes \Psi^{s(i)}(t)_{js})\mathbf{d}^d_j \end{array} \quad (27)$$

where Φ are the matrices of shifts of the different scaling functions samples associated with the i^{th} frame f^i_s. Ψ are the matrices of shifts of the different wavelets functions samples associated with the i^{th} frame f^i_s. The symbol $\otimes$ represents the Kronecker or tensor product. This product is used in order to have a matrix with all possibilities of translation after merging two matrices (scaling or wavelet functions).

1) Computation of the coarse-scale approximation of the image: As shown in the one dimensional case (section II-A2), a coarse-scale approximation image can be computed in the same way and the equation (12) can be written as:

$$\mathbf{f}_s^i \approx (\Phi_{Jt}^{s(i)} \otimes \Phi_J^{s(i)})\mathbf{a}_J \tag{28}$$

The approximation coefficients $\mathbf{a}_J$ can find in a regularized least square sense and as shown in equation (14):

$$\begin{aligned}\hat{\mathbf{a}}_J &= \left(\sum_{i=1}^{n}(\Phi_{Jt}^{s(i)T}\Phi_{Jt}^{s(i)}) \otimes (\Phi_{Js}^{s(i)T}\Phi_{Js}^{s(i)}) + \lambda I\right)^{-1} \\ &\times \sum_{i=1}^{n}\left(\Phi_{Jt}^{s(i)T} \otimes \Phi_{Js}^{s(i)T}\right)\mathbf{f}^i \end{aligned}\tag{29}$$

Once the approximation are computed, an coarse-scale approximation image can be computed as in the one dimensional case (equation (15)):

$$\hat{\mathbf{f}}_0 = \left(\Phi_{Jt}^{(i)T} \otimes \Phi_{Js}^{(i)T}\right)\hat{\mathbf{a}}_J \tag{30}$$

Figure 8(c) shows the result of a coarse approximation image.

When the approximation coefficients have been computed, the detail coefficients can be computed too.

2) Computation of the detail part of the image: As in the one dimensional case, detail coefficients are computed using the residuals (error between the approximation and the different low resolution images). The difference in the two dimensional case compared to the one dimensional is that three different kind of details will be computed at each scale level: horizontal, vertical and diagonal. Thus, first horizontal detail coefficients will be computed as:

$$\begin{aligned}(\Phi^{s(i)}(s)_{jt} &\otimes \Psi^{s(i)}(t)_{js})\mathbf{d}_J^h \approx \mathbf{f}_s^i - \left(\Phi_{Jt}^{s(i)T} \otimes \Phi_{Js}^{s(i)T}\right)\hat{\mathbf{a}}_J \\ \mathbf{e}_h^i &\approx \mathbf{f}_s^i - \left(\Phi_{Jt}^{s(i)T} \otimes \Phi_{Js}^{s(i)T}\right)\hat{\mathbf{a}}_J \end{aligned}\tag{31}$$

In the same way as in the section II-A3, the horizontal detail coefficients will be computed using regularized least square methods:

$$\begin{aligned}\hat{\mathbf{d}}_J^h &= \left(\sum_{i=1}^{n}(\Phi_{Jt}^{s(i)T}\Phi_{Jt}^{s(i)}) \otimes (\Psi_{Js}^{s(i)T}\Psi_{Js}^{s(i)}) + \lambda I\right)^{-1} \\ &\times \sum_{i=1}^{n}\left(\Phi_{Jt}^{s(i)T} \otimes \Psi_{Js}^{s(i)T}\right)\mathbf{e}_h^i \end{aligned}\tag{32}$$

Then, details will be added to the coarse approximation to give a new approximation which will be finer:

$$\hat{\mathbf{f}}_1 = \hat{\mathbf{f}}_0 + (\Phi^{(i)}(s)_{jt} \otimes \Psi^{(i)}(t)_{js})\hat{\mathbf{d}}_J^h \tag{33}$$

Figure 8(d) shows a coarse approximation image where the horizontal details were added.

Replacing $\hat{\mathbf{f}}_0$ by $\hat{\mathbf{f}}_1$ and recomputing the residuals, the vertical and diagonal details will be obtain:

$$\begin{aligned}(\Psi^{s(i)}(s)_{jt} &\otimes \Phi^{s(i)}(t)_{js})\mathbf{d}_J^v \approx \mathbf{f}_s^i - \hat{\mathbf{f}}_0^{s(i)} \\ \mathbf{e}_v^i &\approx \mathbf{f}_s^i - \hat{\mathbf{f}}_0^{s(i)} \end{aligned}\tag{34}$$

$$\begin{aligned}\hat{\mathbf{d}}_J^v &= \left(\sum_{i=1}^{n}(\Psi_{Jt}^{s(i)T}\Psi_{Jt}^{s(i)}) \otimes (\Phi_{Js}^{s(i)T}\Phi_{Js}^{s(i)}) + \lambda I\right)^{-1} \\ &\times \sum_{i=1}^{n}\left(\Psi_{Jt}^{s(i)T} \otimes \Phi_{Js}^{s(i)T}\right)\mathbf{e}_v^i \end{aligned}\tag{35}$$

Replacing $\hat{\mathbf{f}}_0$ by $\hat{\mathbf{f}}_1$:

$$\begin{aligned}(\Psi^{s(i)}(s)_{jt} &\otimes \Psi^{s(i)}(t)_{js})\mathbf{d}_J^d \approx \mathbf{f}_s^i - \hat{\mathbf{f}}_0^{s(i)} \\ \mathbf{e}_d^i &\approx \mathbf{f}_s^i - \hat{\mathbf{f}}_0^{s(i)} \end{aligned}\tag{36}$$

$$\begin{aligned}\hat{\mathbf{d}}_J^d &= \left(\sum_{i=1}^{n}(\Psi_{Jt}^{s(i)T}\Psi_{Jt}^{s(i)}) \otimes (\Psi_{Js}^{s(i)T}\Psi_{Js}^{s(i)}) + \lambda I\right)^{-1} \\ &\times \sum_{i=1}^{n}\left(\Psi_{Jt}^{s(i)T} \otimes \Psi_{Js}^{s(i)T}\right)\mathbf{e}_d^i \end{aligned}\tag{37}$$

Results with vertical and diagonal details added are shown respectively on figure 8(e) and figure 8(f).

A Matlab implementation is given in appendix B.

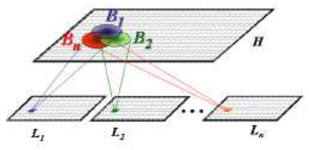
III. Super resolution from a single low resolution image

In the previous section II, in order to obtain an overdetermined system, several low resolution images where needed and each low resolution image give linear constraints as shown in figure 9(a). D. Glasner et al. proposed a method where only a single low resolution image will allow to generate a super resolution image [13].

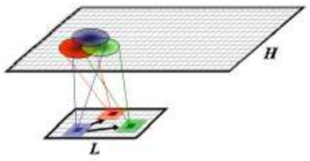
A. Patch redundancy inside a single frame at different scales

Glasner et al. find out that considering a patch in the original image, similar patch can be found inside the same image (figure 9(b)). Pushing the concept further, it will be possible to find similar patches at different scales (lower scale). This redundancy is shown in figure 9.

Glasner et al. analysed on a wide images database the percentage of recurrent patches through different scales. Results are shown in figure 10. Figure 10(a) shows that the percentage of recurrent patches at different scales is very important even at a low scale. In order to see if details so high frequency

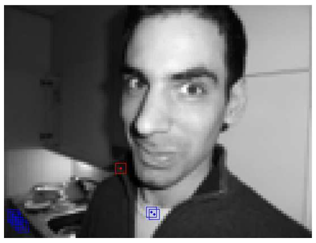


(a) Super resolution using multiple low resolution images. At the same location for each low resolution image, the pixel considered will give information to generate the high resolution image

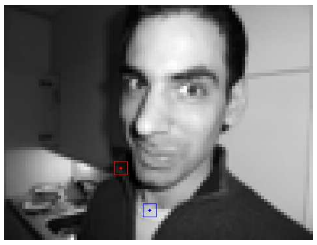


(b) Super resolution using recurrent patches properties. Due of the redundancy of patches inside the same image, different locations inside the image can give linear constraints in order to generate the high resolution image

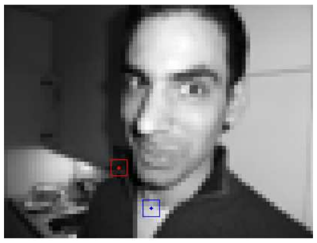
(c) Patch redundancy at the original scale. Red patch is the model patch while blue patches are the nearest neighbour of the model patch



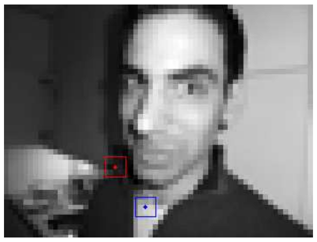
(d) Patch redundancy at the scale 1.25^{-1}. Red patch is the model patch while blue patches are the nearest neighbour of the model patch



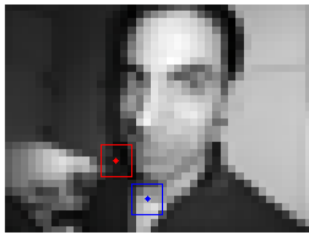
(e) Patch redundancy at the scale 1.25^{-2}. Red patch is the model patch while blue patches are the nearest neighbour of the model patch



(f) Patch redundancy at the scale 1.25^{-3}. Red patch is the model patch while blue patches are the nearest neighbour of the model patch

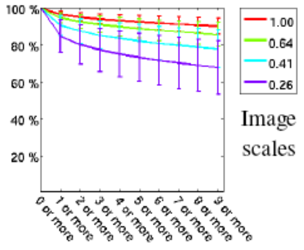


(g) Patch redundancy at the scale 1.25^{-4}. Red patch is the model patch while blue patches are the nearest neighbour of the model patch

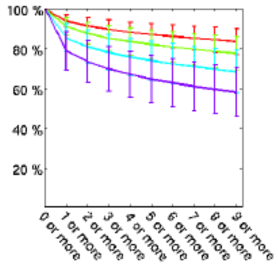


(h) Patch redundancy at the scale 1.25^{-6}. Red patch is the model patch while blue patches are the nearest neighbour of the model patch

Figure 9. Patch redundancy at different scales



(a) Percentage of recurrent patches at different scale for all images



(b) Percentage of recurrent patches at different scale for only high texture location in the images

Figure 10. Patch redundancy at different scales

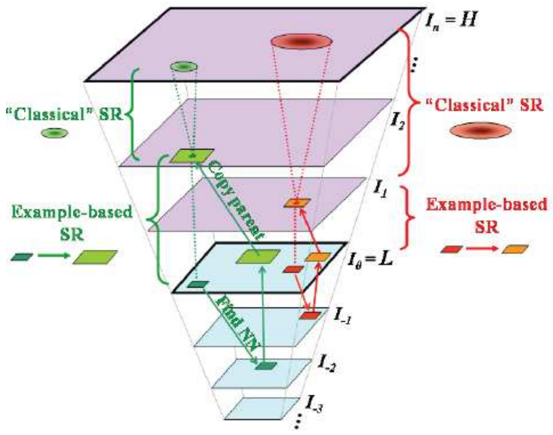


Figure 11. Combining method proposed by Glasner et al. [13]

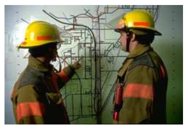
could be recover from these redundancies, they analysed the redundancy patches on the high texture location of the image at different scales. The results are shown on the figure 10(b). The percentage of recurrent patches at low scale only regarding high texture is still large. They concluded that the redundancy of patches will add linear constraints using the same image at different scales.

B. *Scheme of super resolution using patch redundancy*

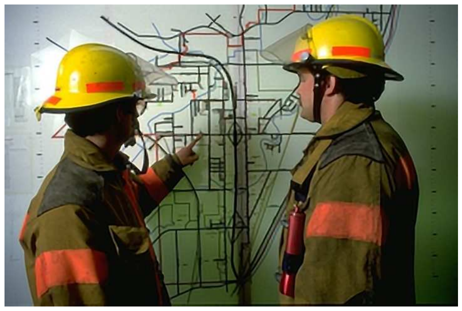
The method proposed by Glasner et al. is illustrated in figure 11. Patches are defined in the low resolution image (dark red and dark green patches). For a scale smaller than the low resolution image, similar patches of the initial patches are found (light red and light green patches). These last patches will bring more linear constraints in order to solve a classical super resolution problem. The information of these patches will be weighted by the similarity with orginal patches. These low scale patches will be copied in order to estimate the super resolution image. An example of result of this method is shown on figure 12.

IV. Super resolution using compressed sampling and sparse representation

The method proposed by Glasner et al. [13] gives good results. However, the purpose of this method was not to break



(a) Low resolution image



(b) Super resolution generated by the algorithm

Figure 12. Results of the method proposed by Glasner et al. [13]

the issue of under determined system which cannot be solved.

Recently, due of several important results by D. Donoho, E. Candes, J. Romberg and T. Tao [15], [16], [17], [18], [19] in the field of compressed sensing, under determined linear system could be solved and give sparse solution. This approach gives a new impetus to the field of super resolution.

Due of the non-sparsity of the spatial domain of the image, compressed sensing cannot be apply directly. However, generally images in the wavelet domain are mostly sparse. Hence, compressed sensing and l_1 normalization can be used to recover high resolution image from a single low resolution image.

The method which will be presented was introduced by J. Yang [6]. In this section, only the part corresponding to the resolution of the super resolution issues will be presented. Hence, the preparation of dictionaries will not be explained. For more information regarding the dictionaries creation, the reader can consult the article [6], [7].

Compressive sensing can give sparse solution using the wavelet domain. Considering a high resolution image X, X can be decomposed of several patches x. Hence, the high resolution image can be decomposed in the wavelet domain as:

$$x = D_h \alpha \tag{38}$$

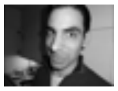
where α represents the wavelet coefficients and D_h represents the dictionary used to decompose the high resolution image. The idea of the algorithm proposed is to approximate α knowing that the vector should be sparse.

In fact, only low resolution image is available. Hence, the goal will be to infer each high resolution patch from each low resolution image using the wavelet coefficients. Hence, α have to be approximated such that α is the sparsest as possible and the difference between the estimation and the real data is as small as possible. This problem can be formalized as:

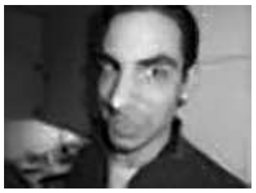
$$\min \|\alpha\|_{l_1} \text{ s.t. } \|FD_l\alpha - Fy\|_2^2 \leq \epsilon \tag{39}$$

where α are the wavelet coefficients, D_l is the dictionary for low resolution decomposition, y is the patch in the low

(a) Original image



(b) Low resolution image



(c) High resolution image reconstruct using the algorithm

Figure 13. Results of the method proposed by J. Yang et al. [6], [7]

resolution image and F is a linear feature extractor operator.

Another constraint is that all different patches found in the high resolution image have to be as consistent as possible each other. Hence, the overlap between each patch will be taken in consideration as each new patch will be compatible with the previous patch computed. This constraint is added to the equation (39) and can be written:

$$\begin{aligned} \min \|\alpha\|_{l_1} \quad \text{s.t.} \quad & \|FD_l\alpha - Fy\|_2^2 \leq \epsilon_1 \\ & \|PD_h\alpha - w\|_2^2 \leq \epsilon_2 \end{aligned} \tag{40}$$

where α are the wavelet coefficients, D_l is the dictionary for low resolution decomposition, y is the patch in the low resolution image and F is a linear feature extractor operator. The matrix P extracts the region of overlap between the current patch and the precious computed patch and w is the matrix containing the values of the previous patch computed.

As in the section II, a regularized linear estimator will be used using a parameter λ which will have the same kind of effect on the solution. This parameter will balance between the sparsity level of the solution and the fidelity of the approximation to y. Hence, the system which will be solved is the following:

$$\min \lambda \|\alpha\|_{l_1} + \frac{1}{2}\left\|\bar{D}\alpha - \bar{y}\right\|_2^2 \tag{41}$$

where $\bar{D} = \begin{bmatrix} FD_l \\ \beta PD_h \end{bmatrix}$ and $\bar{y} = \begin{bmatrix} Fy \\ \beta w \end{bmatrix}$. β will balance between matching the low resolution input and finding high resolution patch which is compatible with its neighbor.

To reconstruct the high resolution patch, the estimation $\hat{\alpha}$ found can be injected inside the equation (38). Results of this algorithm are presented on figure 13.

V. Application

Thus, the advantage of super resolution algorithm is that using only one or several low resolution images, a high resolution image of the scene can be computed.

Hence, it is very obvious that super resolution algorithm can be used to speed up and enhance the quality of images for all type of real-time applications. Moreover, the cost of the sensors produced to acquire low resolution images is less

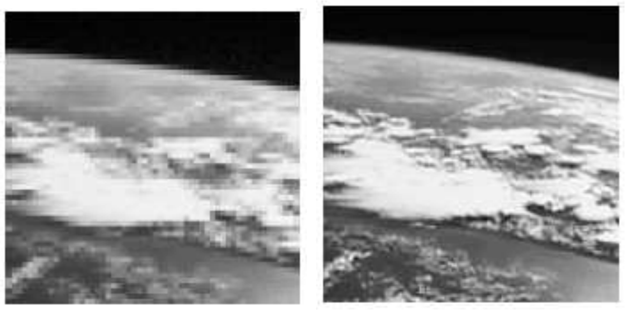

(a) Low resolution image (b) High resolution image computed

Figure 14. Results of the super resolution in astronomy given by R. Willet et al.[14]

expensive than high resolution sensors. The architecture of sensors will be less complex. Miniaturization of elements inside sensors will be less important leading to a financial gain.

Nowadays, super resolution was used also in astronomy. Images acquired in astronomy are usually low resolution and blurred due of physics limitations. Results of super resolution image are presented on figure 14

VI. Conclusion

In this paper, an historical overview of super resolution was given. First, a classical super resolution algorithm was presented. In this case, multiple low resolution images were needed in order to generate a high resolution image. We focus more in details on this method than the other methods presented after that. Then, methods using only one low resolution image was presented. The first method was a method which use a patches based super resolution at different scale in order to find more linear equations so that to have an over determined system. The second solution is based on the discoveries in the field of compressed sensing which allow to find a sparse solution to under determined system which is the case of super resolution in the wavelet domain. Finally, we presented a short section regarding the applications of the super resolution.

References

[1] C. Ford and D. M. Etter, "Wavelet basis reconstruction of nonuniformly sampled data," *IEEE Transactions on Circuits and Systems*, vol. 45, pp. 1165–1168, 1998.

[2] S. G. Mallat, "A theory for multiresolution signal decomposition: the wavelet representation," *IEEE Transactions on Pattern Analysis and Machine Intelligence - II: Analog and Digital Signal Processing*, vol. 11, pp. 674–693, 1989.

[3] N. Nguyen and P. Milanfar, "A wavelet-based interpolation-restoration. method for superresolution (wavelet superresolution)," *IEEE Transactions on Circuits and Systems*, vol. 19, pp. 321–338, 2000.

[4] N. Nguyen, P. Milanfar, and G. H. Golub, "A computationally efficient superresolution image reconstruction algorithm," *IEEE Transactions on Image Processing*, vol. 10, no. 4, pp. 573–583, 2001.

[5] N. Nguyen and P. Milanfar, "An efficient wavelet-based algorithm for image superresolution," in *ICIP*, 2000.

[6] J. Yang, J. Wright, T. Huang, and Y. Ma, "Image super-resolution as sparse representation of raw image patches," *Computer Vision and Pattern Recognition, IEEE Computer Society Conference on*, vol. 0, pp. 1–8, 2008.

[7] ——, "Image super-resolution via sparse representation," *IEEE Transactions on Image Processing*, vol. 19, pp. 2861 – 2873, 2010.

[8] F. Truchetet, *Ondelettes pour le signal numerique*, Hermes, Ed. Hermes, 1998.

[9] ——, *Traitement lineaire du signal numerique*, Hermes, Ed. Hermes, 1998.

[10] S. Mallat, *Une exploration des signaux en ondelettes*, E. de l'ecole polytechnique, Ed. Ellipses Diffusion, 2000.

[11] C. McGregor, J. Nimmo, and W. Stothers, *Fundamentals of university mathematics*, Horwood, Ed. Horwood, 2007.

[12] S. Mallat, *A wavelet tour of signal processing*, A. Press, Ed. Academic Press, 2008.

[13] D. Glasner, S. Bagon, and M. Irani, "Super-resolution from a single image," in *ICCV*, 2009. [Online]. Available: http://www.wisdom.weizmann.ac.il/ vision/SingleImageSR.html

[14] R. Willet, I. Jermyn, R. Nowak, and J. Zerubia, "Wavelet-based super-resolution in astronomy," 2004.

[15] E. Candes, J. Romberg, and T. Tao, "Robust uncertainty principles: Exact signal reconstruction from highly incomplete frequency information," *IEEE Transactions on Information Theory*, vol. 52, pp. 489 – 509, 2006.

[16] ——, "Stable signal recovery from incomplete and inacurate measurements," *Communications on Pure and Applied Mathematics*, vol. 59, pp. 1207 – 1223, 2006.

[17] E. Candes and T. Tao, "Near-optimal isignal recovery from random projections and universal encoding strategies?" *IEEE Transactions on Information Theory*, vol. 52, pp. 5406 – 5245, 2006.

[18] E. Candes and J. Romberg, "Sparsity and incoherence in compressive sampling," *Inverse Problem*, vol. 23, pp. 969 – 986, 2007.

[19] E. Candes and T. Tao, "The dantzig selector: Statistical estimation when p is much smaller than n," *Annal of Statistics*, vol. 35, pp. 2392 – 2404, 2007.

[20] M. F. Duarte, M. A. Davenport, D. Takhar, J. N. Laska, T. Sun, K. F. Kelly, and R. G. Baraniuk, "Single-pixel imaging via compressive sampling," *IEEE Signal Processing Magazine*, vol. 1, pp. 83 – 91, 2008.

APPENDIX A
IMPLEMENTATION OF ONE DIMENSIONAL CASE SUPER RESOLUTION

A. *Main file*

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%% Guillaume Lemaitre
%%% -------------------------
%%% This function is an implementation of the 1D super resolution presented
%%% by:
%%% C. Ford and D. M. Etter, Wavelet Basis Reconstruction of Nonuniformly
%%% Sampled Data, IEEE TRANSACTIONS ON CIRCUITS AND SYSTEMSII: ANALOG AND
%%% DIGITAL SIGNAL PROCESSING, VOL. 45, NO. 8, AUGUST 1998
%%% and
%%% Nhat Nguyen and Peyman Milanfar, A Wavelet-Baset
%%% interpolation-restoration method for super resolution (Wavelet Super
%%% Resolution), Circuits System Signal Process, VOL. 19, NO. 4, 2000, pg
%%% 321-338
%%% -------------------------
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

clear all;
close all;
clc;

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%% Paramters
%%% Total numbers of points
nbPoints = 200;
t0 = 0;
tmax = 10;
%%% Factor of downsampling
factorDW = 4;
%%% Number of images which have to be considered
nbImages = 3;
%%% Color vector
colorVector = ['+r'; '+g' ; '+y' ; '+c' ; '+m' ; '+k'];
%%% Wavelet family
filter = 'db6';
lengthSupport = 11;
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%% Generation of a synthetic signal
%%% Original Signal:
xOriginal = 1:nbPoints;
%fOriginal = cos((2*pi*xOriginal).*((tmax-t0)/nbPoints)).*(sqrt(xOriginal).*((tmax-t0)/nbPoints))
% + sin((5*pi*xOriginal).*((tmax-t0)/nbPoints) + 10).*(sqrt(xOriginal).*((tmax-t0)/nbPoints));
fOriginal = sin(xOriginal.*((tmax-t0)/nbPoints)) + sin(5*xOriginal.*((tmax-t0)/nbPoints) + 10)
 + cos(3*xOriginal.*((tmax-t0)/nbPoints) + 10);

figure(1);
subplot(321); plot(xOriginal,fOriginal);
title('Original synthetic signal');

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%% Compute the different frames
%%% Compute the downsampling signal
[xLowResolution,fLowResolution] = frameCreation(xOriginal,fOriginal,factorDW);

for i = 1:factorDW
    subplot(322);hold on;plot(xLowResolution(i,:),fLowResolution(i,:), num2str(colorVector(i,:)));
end
title('Different frames after random sampling');
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%


%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%% Wavelet reconstruction using only Low Resolution data
%%% ---------------------------------------------------------------------
%%% In this first part, we will only compute a coarse approximation of the
%%% signal
%%% ---------------------------------------------------------------------
%%% ----------------------------------------------------------------------
%%% Parameters
%%% Compute the scaling and wavelet function
[phi,psi,xval] = wavefun(filter,10);
%%% Scale where we will estimate the coarse signal
J = 2;

k = floor(-lengthSupport + 2ˆ(-J)*xOriginal(1)) + 1:2ˆ(-J)*xOriginal(length(xOriginal));

%%% Preallocation
GJs = zeros(length(xLowResolution), length(k));

for i = 1:nbImages

    %%% ----------------------------------------------------------------------
    %%% Compute the matrix contaning the shift of the scaling function at
    %%% the different points knew

    for j = 1:length(xLowResolution)
        count = 1;
        for k2 = k(1):k(length(k))
            [phis,psis,xvals] = computeScaleTranslateWavelet(phi,psi,xval,J,k2);
            ind = find((xLowResolution(i,j)==xvals));
            if(¬isempty(ind))
                GJs(j,count) = phis(ind);
            end
            count = count + 1;
        end
    end

    %%% ----------------------------------------------------------------------
    %%% Compute the approximation coefficient in a Regularized Least-Squares
    %%% sense
    lambda = 0.01;

    if (i == 1)
        leftPart = GJs'*GJs + lambda.*eye(size(GJs'*GJs));
        rightPart = GJs'*fLowResolution(i,:)';
    else
        leftPart = leftPart + (GJs'*GJs + lambda.*eye(size(GJs'*GJs)));
        rightPart = rightPart + (GJs'*fLowResolution(i,:)');
    end

end

approCoefficient = (leftPart\rightPart);

%%% ----------------------------------------------------------------------
%%% Compute the coarse approximation
%%% Preallocation
GJ = zeros(length(xOriginal), length(k));

%%% ----------------------------------------------------------------------
%%% Compute the matrix contaning the shift of the scaling function at
%%% the different points wanted

for j = 1:length(xOriginal)
    count = 1;
    for k2 = k(1):k(length(k))
        [phis,psis,xvals] = computeScaleTranslateWavelet(phi,psi,xval,J,k2);
        ind = find(xOriginal(j)==xvals);
        if(¬isempty(ind))
            GJ(j,count) = phis(ind);
        end
        count = count + 1;
    end
end

%%% ----------------------------------------------------------------------
%%% Computation of the approximation curve
f0 = GJ*approCoefficient;

%%% ----------------------------------------------------------------------
%%% Plot the result of the approximation
subplot(323);
for i = 1:nbImages
    %%% Plot the point used for the approximation
    hold on;
    plot(xLowResolution(i,:),fLowResolution(i,:), num2str(colorVector(i,:)));
end
%%% Plot the coarse curve
hold on;
plot(xOriginal,f0);
title('Coarse approximation and data used to obtain this approximation');
%%% Plot the approximation curve and the original curve
subplot(324);
%%% Plot the original curve
plot(xOriginal,fOriginal, 'r');
%%% Plot the coarse curve
hold on;
tmp = f0;
plot(xOriginal,tmp);
title('Coarse approximation (blue) - Original approximation (red)');

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%% Wavelet reconstruction - Reconstruction of the different level of
%%% details

for i = J:-1:1
    clear leftPart;
    clear rightPart;

    %%% Preallocation
    Hjs = zeros(length(xLowResolution), length(k));

    %%% Compute the error between the approximation and the signal at the
    %%% sampling points
    error = zeros(nbImages,length(xLowResolution));
    for j = 1:nbImages
        for a = 1:length(xLowResolution(j,:))
            error(j,a) = fLowResolution(j,a) - f0(xLowResolution(j,a));
        end
    end

    %%% ------------------------------------------------------------------
    %%% Compute the matrix containg the shift of the wavelet function at
    %%% the different points knew
    for l = 1:nbImages

        for j = 1:length(xLowResolution)
            count = 1;
            for k2 = k(1):k(length(k))
                [phis,psis,xvals] = computeScaleTranslateWavelet(phi,psi,xval,i,k2);
                ind = find((xLowResolution(l,j)==xvals));
                if(¬isempty(ind))
                    Hjs(j,count) = phis(ind);
                end
                count = count + 1;
            end
        end

        %%% ---------------------------------------------------------------
        %%% Compute the approximation coefficient in a Regularized Least-Squares
        %%% sense
        lambda = 0.001;

        if (l == 1)
            leftPart = Hjs'*Hjs + lambda.*eye(size(Hjs'*Hjs));
            rightPart = Hjs'*error(l,:)';
        else
            leftPart = leftPart + (Hjs'*Hjs + lambda.*eye(size(Hjs'*Hjs)));
            rightPart = rightPart + (Hjs'*error(l,:)');
        end
    end

    detailsCoeff = (leftPart\rightPart);


    %%% ----------------------------------------------------------------------
    %%% Compute the coarse approximation
    %%% Preallocation
    Hj = zeros(length(xOriginal), length(k));

    %%% ----------------------------------------------------------------------
    %%% Compute the matrix contaning the shift of the scaling function at
    %%% the different points wanted

    for j = 1:length(xOriginal)
        count = 1;
        for k2 = k(1):k(length(k))
            [phis,psis,xvals] = computeScaleTranslateWavelet(phi,psi,xval,i,k2);
            ind = find(xOriginal(j)==xvals);
            if(¬isempty(ind))
                Hj(j,count) = phis(ind);
            end
            count = count + 1;
        end
    end

    %%% ----------------------------------------------------------------------
    %%% Computation of the details curve
    f1 = f0 + Hj*detailsCoeff;
    f0 = f1;
end

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%% Plot the details on the last plot
subplot(325);
plot(xOriginal,tmp,'b');
hold on;
plot(xOriginal,f1,'g');
title('Coarse approximation (blue) - Final Estimation (green)');

subplot(326);
plot(xOriginal,fOriginal,'r');
hold on;
plot(xOriginal,f1,'g');
title('Final Estimation (green) - Original approximation (red)');
```

B. Function to compute the different wavelet family

```
function [phist,psist,xvalst] = computeScaleTranslateWavelet(phi,psi,xval,scale,translation)

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%% Guillaume Lemaitre
%%% -------------------------
%%% This function allows to compute the scaling function and the wavelet
%%% function at different scale and translation
%%% -------------------------
%%% Input:
%%% phi: original scaling function
%%% psi: original wavelet function
%%% xval: time discretisation vector
%%% scale: factor of rescaling
%%% translation: factor of translation
%%% -------------------------
%%% Output:
%%% phist: scaling function to the new scale and translated
%%% psist: scaling function to the new scale and translated
%%% xvalst: time value connected to the scaling and translated function
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

xvalst = 2ˆ(scale).*xval;
xvalst = xvalst + repmat(2ˆ(scale)*translation , [ 1 , length(xval) ]);
phist = 2ˆ(-scale/2).*phi;
psist = 2ˆ(-scale/2).*psi;
```

APPENDIX B
IMPLEMENTATION OF TWO DIMENSIONAL CASE SUPER RESOLUTION

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%% Guillaume Lemaitre
%%% -------------------------
%%% This function is an implementation of the 2D super resolution presented
%%% by:
```

```
%%% Nhat Nguyen and Peyman Milanfar, A Wavelet-Baset
%%% interpolation-restoration method for super resolution (Wavelet Super
%%% Resolution), Circuits System Signal Process, VOL. 19, NO. 4, 2000, pg
%%% 321-338
%%% -------------------------
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

clear all;
close all;
clc;

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%% Parameters
%%% Add directories
addpath('data/original');
addpath('data/Low Resolution');

%%% Factor of downsampling
factorDW = 4;
%%% Number of images which have to be considered
nbImages = 10;
%%% Color vector
colorVector = ['+r'; '+g' ; '+y' ; '+c' ; '+m' ; '+k'];
%%% Wavelet family
filter = 'db4';
lengthSupport = 7;
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%% Intialisation

%%% Read the low resolution images
for i = 1:(factorDW*factorDW)
    imagesLR(:,:,i) = im2double(imread(['data/Low Resolution/image' , num2str(i) , '.bmp']));
    if (i == 1)
        %%% Compute the size of the LR images
        [heigthLR , widthLR] = size(imagesLR(:,:,1));
    end
    %%% Compute the pixel correspondance of each image
    for j = 1:heigthLR
        for k = 1:widthLR
            %%% For the horizontal pixel
            corrIndexLR(j,k,i,1) = ((j - 1)*factorDW + floor((i - 1)/factorDW)) + 1;
            %%% For the vertical pixel
            corrIndexLR(j,k,i,2) = ((k - 1)*factorDW + mod((i - 1),factorDW)) + 1;
        end
    end
end

imwrite(imagesLR(:,:,1),'LR.bmp','bmp');

%%% Read original picture
%%% Read the image
originalImage = im2double(rgb2gray(imread('data/original/image.jpg')));
subplot(232);
imshow(originalImage);
title('Original Image');

%%% Create a random array to select n low resolution images
tmp = randperm(factorDW*factorDW);
indexImagesLR = tmp(1:nbImages);
indexImagesLR = sort(indexImagesLR);
clear tmp;

%%% Compute the quasi image
count = 1;
for i = 1:nbImages
    for j = 1:heigthLR
        for k = 1:widthLR
            %%% Compute the position in a row
            row = ((j - 1)*factorDW + (floor((indexImagesLR(i) - 1)/factorDW))+1);
            col = ((k - 1)*factorDW + mod((indexImagesLR(i) - 1),factorDW))+1;

            count = count + 1;
            %%%%%%%%%%%%%%%%%%%%%%
            quasiHighResolutionImage(((j - 1)*factorDW + ...
                (floor((indexImagesLR(i) - 1)/factorDW))+1) ...
                , ((k - 1)*factorDW + mod((indexImagesLR(i) - 1),factorDW))+1) = ...
                imagesLR(j,k,indexImagesLR(i));
        end
    end
end

imwrite(quasiHighResolutionImage,'quasi.bmp','bmp');

%%% Plot an example of low resolution image
subplot(231);
imshow(imagesLR(:,:,1));
title('Example of low resolution images');
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%% Wavelet reconstruction using only Low Resolution data
%%% ----------------------------------------------------------------------
%%% In this first part, we will only compute a coarse approximation of the
%%% signal
%%% ----------------------------------------------------------------------

%%% ----------------------------------------------------------------------
%%% Parameters
%%% Compute the scaling and wavelet function
[phi,psi,xval] = wavefun(filter,10);
%%% Scale where we will estimate the coarse signal
J = 2;
lambda = 0.0001;

%%% Compute the translator vector for the horizontal and vertical direction
kHorizontal = floor(-lengthSupport + 2^(-J)) + 1:2^(-J)*(size(imagesLR(:,:,1),1)*factorDW);
kVertical = floor(-lengthSupport + 2^(-J)) + 1:2^(-J)*(size(imagesLR(:,:,1),2)*factorDW);

%%% Preallocation
GJhs = zeros(size(imagesLR(:,:,1),1), length(kHorizontal));
GJvs = zeros(size(imagesLR(:,:,1),2), length(kVertical));

for i = 1:nbImages

    %%% ----------------------------------------------------------------------
    %%% Compute the matrix containing the shift of the scaling function at
    %%% the different points knew
    %%% Horizontal scaling function

    for j = 1:size(GJhs,1)
        count = 1;
        for k2 = kHorizontal(1):kHorizontal(length(kHorizontal))
            [phis,psis,xvals] = computeScaleTranslateWavelet(phi,psi,xval,J,k2);
            ind = find(corrIndexLR(j,1,indexImagesLR(i),1)==xvals);
            if(¬isempty(ind))
                GJhs(j,count) = phis(ind);
            end
            count = count + 1;
        end
    end

    %%% Vertical scaling function

    for j = 1:size(GJvs,1)
        count = 1;
        for k2 = kVertical(1):kVertical(length(kVertical))
            [phis,psis,xvals] = computeScaleTranslateWavelet(phi,psi,xval,J,k2);
            ind = find(corrIndexLR(1,j,indexImagesLR(i),2)==xvals);
            if(¬isempty(ind))
                GJvs(j,count) = phis(ind);
            end
            count = count + 1;
        end
    end


    if (i == 1)
        leftsum = kron((GJhs'*GJhs),(GJvs'*GJvs)) + lambda.*eye(size(kron((GJhs'*GJhs),(GJvs'*GJvs))));
        tmp = imagesLR(:,:,indexImagesLR(i))';
        rightsum = kron(GJhs',GJvs')*tmp(:);
    else
        leftsum = leftsum + kron((GJhs'*GJhs),(GJvs'*GJvs)) + ...
```

```
                 lambda.*eye(size(kron((GJhs'*GJhs),(GJvs'*GJvs))));
        tmp = imagesLR(:,:,indexImagesLR(i))';
        rightsum = rightsum + kron(GJhs',GJvs')*tmp(:);
    end

end

%%% Computation of the approximation coefficient
approCoeff = (leftsum\rightsum);

%%% ----------------------------------------------------------------------
%%% Compute the coarse approximation
%%% Preallocation
GJh = zeros(size(imagesLR(:,:,1),1)*factorDW, length(kHorizontal));
GJv = zeros(size(imagesLR(:,:,1),2)*factorDW, length(kVertical));

%%% ----------------------------------------------------------------------
%%% Compute the matrix contaning the shift of the scaling function at
%%% the different points wanted
%%% Horizontal scaling function

for j = 1:size(GJh,1)
    count = 1;
    for k2 = kHorizontal(1):kHorizontal(length(kHorizontal))
        [phis,psis,xvals] = computeScaleTranslateWavelet(phi,psi,xval,J,k2);
        ind = find(j==xvals);
        if(¬isempty(ind))
            GJh(j,count) = phis(ind);
        end
        count = count + 1;
    end
end

%%% Vertical scaling function

for j = 1:size(GJv,1)
    count = 1;
    for k2 = kVertical(1):kVertical(length(kVertical))
        [phis,psis,xvals] = computeScaleTranslateWavelet(phi,psi,xval,J,k2);
        ind = find(j==xvals);
        if(¬isempty(ind))
            GJv(j,count) = phis(ind);
        end
        count = count + 1;
    end
end

%%% ----------------------------------------------------------------------
%%% Computation of the approximation curve
f0 = (kron(GJh,GJv)*approCoeff);
coarseApproximation = reshape(f0,widthLR*factorDW,heigthLR*factorDW)';

%%% Plot the result of the approximation
subplot(233);
imshow(coarseApproximation);
title('Coarse approximation');

imwrite(coarseApproximation,'coarse.bmp','bmp');

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%% Wavelet reconstruction - Reconstruction of the different level of
%%% details

lambda = 0.1;
for scale = J:-1:1

%%% Compute the translator vector for the horizontal and vertical direction
kHorizontal = floor(-lengthSupport + 2ˆ(-J)) + 1:2ˆ(-J)*(size(imagesLR(:,:,1),1)*factorDW);
kVertical = floor(-lengthSupport + 2ˆ(-J)) + 1:2ˆ(-J)*(size(imagesLR(:,:,1),2)*factorDW);

%%% Preallocation
Gjhs = zeros(size(imagesLR(:,:,1),1), length(kHorizontal));
Gjvs = zeros(size(imagesLR(:,:,1),2), length(kVertical));
Hjhs = zeros(size(imagesLR(:,:,1),1), length(kHorizontal));
Hjvs = zeros(size(imagesLR(:,:,1),2), length(kVertical));
Gjh = zeros(size(imagesLR(:,:,1),1)*factorDW, length(kHorizontal));
Gjv = zeros(size(imagesLR(:,:,1),2)*factorDW, length(kVertical));
```

```
Hjh = zeros(size(imagesLR(:,:,1),1)*factorDW, length(kHorizontal));
Hjv = zeros(size(imagesLR(:,:,1),2)*factorDW, length(kVertical));

clear leftsum;
clear rightsum;

    %%% Create some low resolution images of the approximation image to
    %%% compute the low resolution errors image
    %%% Preallocation
    approImagesLR = zeros(size(imagesLR));
    for a = 0:(factorDW - 1)
        for b = 0:(factorDW - 1)
            col = 1;
            row = 1;
            for c = (1 + a):factorDW:heigthLR*factorDW
                for d = (1 + b):factorDW:widthLR*factorDW
                    approImagesLR(row,col, ((a * factorDW) + b) + 1) = coarseApproximation(c,d);
                    col = col + 1;
                end
                row = row + 1;
                col = 1;
            end
        end
    end
    %%% Compute the erros for every images
    errorImages = imagesLR - approImagesLR;

    for i = 1:nbImages

        %%% -------------------------------------------------------------------
        %%% Compute horizontal, vertical scaling function and also horizontal,
        %%% vertical wavelet function

        %%% Horizontal wavelet function

Gjhs = zeros(size(imagesLR(:,:,1),1), length(kHorizontal));
Gjvs = zeros(size(imagesLR(:,:,1),2), length(kVertical));
Hjhs = zeros(size(imagesLR(:,:,1),1), length(kHorizontal));
Hjvs = zeros(size(imagesLR(:,:,1),2), length(kVertical));

        for j = 1:size(Gjhs,1)
            count = 1;
            for k2 = kHorizontal(1):kHorizontal(length(kHorizontal))
                [phis,psis,xvals] = computeScaleTranslateWavelet(phi,psi,xval,scale,k2);
                ind = find(corrIndexLR(j,1,indexImagesLR(i),1)==xvals);
                if(¬isempty(ind))
                    Gjhs(j,count) = phis(ind);
                end
                count = count + 1;
            end
        end

        %%% Vertical scaling function

        for j = 1:size(Hjvs,1)
            count = 1;
            for k2 = kVertical(1):kVertical(length(kVertical))
                [phis,psis,xvals] = computeScaleTranslateWavelet(phi,psi,xval,scale,k2);
                ind = find(corrIndexLR(1,j,indexImagesLR(i),2)==xvals);
                if(¬isempty(ind))
                    Hjvs(j,count) = psis(ind);
                end
                count = count + 1;
            end
        end

        %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
        %%% Detect horizontal details

        if (i == 1)
            leftsum = kron((Gjhs'*Gjhs),(Hjvs'*Hjvs)) + lambda.*eye(size(kron((Gjhs'*Gjhs),(Hjvs'*Hjvs))));
            tmp = errorImages(:,:,indexImagesLR(i))';
            rightsum = kron(Gjhs',Hjvs')*tmp(:);
        else
            leftsum = leftsum + kron((Gjhs'*Gjhs),(Hjvs'*Hjvs)) + ...
                         lambda.*eye(size(kron((Gjhs'*Gjhs),(Hjvs'*Hjvs))));
            tmp = errorImages(:,:,indexImagesLR(i))';
            rightsum = rightsum + kron(Gjhs',Hjvs')*tmp(:);
        end
    end

    %%% Computation of the approximation coefficient
    detailsHCoeff = (leftsum\rightsum);

    %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
    %%% Compute the horizontal image
    %%% ---------------------------------------------------------------------
    %%% Compute the matrix contaning the shift of the scaling function at
    %%% the different points wanted
    %%% Horizontal wavelet function

    for j = 1:size(Gjh,1)
        count = 1;
        for k2 = kHorizontal(1):kHorizontal(length(kHorizontal))
            [phis,psis,xvals] = computeScaleTranslateWavelet(phi,psi,xval,scale,k2);
            ind = find(j==xvals);
            if(¬isempty(ind))
                Gjh(j,count) = phis(ind);
            end
            count = count + 1;
        end
    end

    %%% Vertical scaling function

    for j = 1:size(Hjv,1)
        count = 1;
        for k2 = kVertical(1):kVertical(length(kVertical))
            [phis,psis,xvals] = computeScaleTranslateWavelet(phi,psi,xval,scale,k2);
            ind = find(j==xvals);
            if(¬isempty(ind))
                Hjv(j,count) = psis(ind);
            end
            count = count + 1;
        end
    end


    %%% Compute the new image with horizontal details
    f1 = f0 + (kron(Gjh,Hjv)*detailsHCoeff);
    horizontalApproximation = reshape(f1,widthLR*factorDW,heigthLR*factorDW)';

    %%% Plot the result of the approximation
    subplot(234);
    imshow(horizontalApproximation);
    title('Horizontal approximation');
    imwrite(horizontalApproximation,'horizontal.bmp','bmp');

    f0 = f1;
    coarseApproximation = horizontalApproximation;
    clear leftsum;
    clear rightsum;

    %%% Create some low resolution images of the approximation image to
    %%% compute the low resolution errors image
    %%% Preallocation
    approImagesLR = zeros(size(imagesLR));
    for a = 0:(factorDW - 1)
        for b = 0:(factorDW - 1)
            col = 1;
            row = 1;
            for c = (1 + a):factorDW:heigthLR*factorDW
                for d = (1 + b):factorDW:widthLR*factorDW
                    approImagesLR(row,col, ((a * factorDW) + b) + 1) = coarseApproximation(c,d);
                    col = col + 1;
                end
                row = row + 1;
                col = 1;
            end
        end
    end
    %%% Compute the erros for every images
    errorImages = imagesLR - approImagesLR;
```

```
    for i = 1:nbImages

        %%% -----------------------------------------------------------------
        %%% Compute horizontal, vertical scaling function and also horizontal,
        %%% vertical wavelet function

        %%% Horizontal scaling function

Gjhs = zeros(size(imagesLR(:,:,1),1), length(kHorizontal));
Gjvs = zeros(size(imagesLR(:,:,1),2), length(kVertical));
Hjhs = zeros(size(imagesLR(:,:,1),1), length(kHorizontal));
Hjvs = zeros(size(imagesLR(:,:,1),2), length(kVertical));

        for j = 1:size(Hjhs,1)
            count = 1;
            for k2 = kHorizontal(1):kHorizontal(length(kHorizontal))
                [phis,psis,xvals] = computeScaleTranslateWavelet(phi,psi,xval,scale,k2);
                ind = find(corrIndexLR(j,1,indexImagesLR(i),1)==xvals);
                if(¬isempty(ind))
                    Hjhs(j,count) = psis(ind);
                end
                count = count + 1;
            end
        end

        %%% Vertical wavelet function

        for j = 1:size(Gjvs,1)
            count = 1;
            for k2 = kVertical(1):kVertical(length(kVertical))
                [phis,psis,xvals] = computeScaleTranslateWavelet(phi,psi,xval,scale,k2);
                ind = find(corrIndexLR(1,j,indexImagesLR(i),2)==xvals);
                if(¬isempty(ind))
                    Gjvs(j,count) = phis(ind);
                end
                count = count + 1;
            end
        end

        %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
        %%% Detect horizontal details

        if (i == 1)
            leftsum = kron((Hjhs'*Hjhs),(Gjvs'*Gjvs)) + lambda.*eye(size(kron((Hjhs'*Hjhs),(Gjvs'*Gjvs))));
            tmp = errorImages(:,:,indexImagesLR(i))';
            rightsum = kron(Hjhs',Gjvs')*tmp(:);
        else
            leftsum = leftsum + kron((Hjhs'*Hjhs),(Gjvs'*Gjvs)) + ...
                                 lambda.*eye(size(kron((Hjhs'*Hjhs),(Gjvs'*Gjvs))));
            tmp = errorImages(:,:,indexImagesLR(i))';
            rightsum = rightsum + kron(Hjhs',Gjvs')*tmp(:);
        end
    end

    %%% Computation of the approximation coefficient
    detailsVCoeff = (leftsum\rightsum);

    %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
    %%% Compute the vertical image
    %%% -----------------------------------------------------------------------
    %%% Compute the matrix contaning the shift of the scaling function at
    %%% the different points wanted
    %%% Vertical scaling function

    for j = 1:size(Hjh,1)
        count = 1;
        for k2 = kHorizontal(1):kHorizontal(length(kHorizontal))
            [phis,psis,xvals] = computeScaleTranslateWavelet(phi,psi,xval,scale,k2);
            ind = find(j==xvals);
            if(¬isempty(ind))
                Hjh(j,count) = psis(ind);
            end
            count = count + 1;
        end
    end

    %%% Horizontal wavelet function
    for j = 1:size(Gjv,1)
        count = 1;
        for k2 = kVertical(1):kVertical(length(kVertical))
            [phis,psis,xvals] = computeScaleTranslateWavelet(phi,psi,xval,scale,k2);
            ind = find(j==xvals);
            if(¬isempty(ind))
                Gjv(j,count) = phis(ind);
            end
            count = count + 1;
        end
    end

    %%% Compute the new image with horizontal details
    f1 = f0 + (kron(Hjh,Gjv)*detailsVCoeff);
    verticalApproximation = reshape(f1,widthLR*factorDW,heigthLR*factorDW)';

    %%% Plot the result of the approximation
    subplot(235);
    imshow(verticalApproximation);
    title('Vertical approximation');
    imwrite(verticalApproximation,'vertical.bmp','bmp');

    f0 = f1;
    coarseApproximation = verticalApproximation;

    clear leftsum;
    clear rightsum;

    %%% Create some low resolution images of the approximation image to
    %%% compute the low resolution errors image
    %%% Preallocation
    approImagesLR = zeros(size(imagesLR));
    for a = 0:(factorDW - 1)
        for b = 0:(factorDW - 1)
            col = 1;
            row = 1;
            for c = (1 + a):factorDW:heigthLR*factorDW
                for d = (1 + b):factorDW:widthLR*factorDW
                    approImagesLR(row,col, ((a * factorDW) + b) + 1) = coarseApproximation(c,d);
                    col = col + 1;
                end
                row = row + 1;
                col = 1;
            end
        end
    end
    %%% Compute the erros for every images
    errorImages = imagesLR - approImagesLR;

    for i = 1:nbImages

        %%% -------------------------------------------------------------------
        %%% Compute horizontal, vertical scaling function and also horizontal,
        %%% vertical wavelet function

Gjhs = zeros(size(imagesLR(:,:,1),1), length(kHorizontal));
Gjvs = zeros(size(imagesLR(:,:,1),2), length(kVertical));
Hjhs = zeros(size(imagesLR(:,:,1),1), length(kHorizontal));
Hjvs = zeros(size(imagesLR(:,:,1),2), length(kVertical));

        %%% Vertical scaling function

        for j = 1:size(Hjvs,1)
            count = 1;
            for k2 = kVertical(1):kVertical(length(kVertical))
                [phis,psis,xvals] = computeScaleTranslateWavelet(phi,psi,xval,scale,k2);
                ind = find(corrIndexLR(1,j,indexImagesLR(i),2)==xvals);
                if(¬isempty(ind))
                    Hjvs(j,count) = psis(ind);
                end
                count = count + 1;
            end
        end

        %%% Horizontal wavelet function
        for j = 1:size(Hjhs,1)
            count = 1;
            for k2 = kHorizontal(1):kHorizontal(length(kHorizontal))
                [phis,psis,xvals] = computeScaleTranslateWavelet(phi,psi,xval,scale,k2);
                ind = find(corrIndexLR(j,1,indexImagesLR(i),1)==xvals);
                if(¬isempty(ind))
                    Hjhs(j,count) = psis(ind);
                end
                count = count + 1;
            end
        end

        %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
        %%% Detect horizontal details

        if (i == 1)
            leftsum = kron((Hjhs'*Hjhs),(Hjvs'*Hjvs)) + lambda.*eye(size(kron((Hjhs'*Hjhs),(Hjvs'*Hjvs))));
            tmp = errorImages(:,:,indexImagesLR(i))';
            rightsum = kron(Hjhs',Hjvs')*tmp(:);
        else
            leftsum = leftsum + kron((Hjhs'*Hjhs),(Hjvs'*Hjvs)) + ...
                             lambda.*eye(size(kron((Hjhs'*Hjhs),(Hjvs'*Hjvs))));
            tmp = errorImages(:,:,indexImagesLR(i))';
            rightsum = rightsum + kron(Hjhs',Hjvs')*tmp(:);
        end
    end

    %%% Computation of the approximation coefficient
    detailsDCoeff = (leftsum\rightsum);

    %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
    %%% Compute the diagonal image
    %%% ----------------------------------------------------------------------
    %%% Compute the matrix contaning the shift of the scaling function at
    %%% the different points wanted
    %%% Vertical scaling function

    for j = 1:size(Hjv,1)
        count = 1;
        for k2 = kVertical(1):kVertical(length(kVertical))
            [phis,psis,xvals] = computeScaleTranslateWavelet(phi,psi,xval,scale,k2);
            ind = find(j==xvals);
            if(¬isempty(ind))
                Hjv(j,count) = psis(ind);
            end
            count = count + 1;
        end
    end

    %%% Horizontal wavelet function

    for j = 1:size(Hjh,1)
        count = 1;
        for k2 = kHorizontal(1):kHorizontal(length(kHorizontal))
            [phis,psis,xvals] = computeScaleTranslateWavelet(phi,psi,xval,scale,k2);
            ind = find(j==xvals);
            if(¬isempty(ind))
                Hjh(j,count) = psis(ind);
            end
            count = count + 1;
        end
    end

    %%% Compute the new image with horizontal details
    f1 = f0 + (kron(Hjh,Hjv)*detailsDCoeff);
    diagonalApproximation = reshape(f1,widthLR*factorDW,heigthLR*factorDW)';

    %%% Plot the result of the approximation
    subplot(236);
    imshow(diagonalApproximation);
    title('Diagonal approximation');
    imwrite(diagonalApproximation,'diagonal.bmp','bmp');

    f0 = f1;
    coarseApproximation = diagonalApproximation;

end
```