The Electronic Journal of Mathematics and Technology, Volume 7, Number 4, ISSN 1933-2823

The SVD and some applications in digital image processing

M. Dolores Martínez
e-mail: dolores.martinez@uv.es
Universitat de València. Spain

Javier Pastor
e-mail: pastorv@uv.es
Universitat de València. Spain

Abstract

Our purpose is to bring in the singular value descomposition of a matrix (SVD) to a broad audience, showing that it provides a wide amount of interesting information about the matrix. Moreover, we show that the SVD is an efficient computational platform in applications in digital image processing such as image watermarking, image compression or image restoration.

1 Introduction

In order to realize of the importance, both theoretical and practical, of the singular value decomposition of a matrix (SVD), it is enough to check, by means of a web browser as for example Google, that the SVD plays a central role in a number of papers in recent years. In the same direction, it is interesting to point out that Gilbert Strang designates the existence of the SVD of a matrix and their consequences as 'The fundamental theorem of linear algebra' ([8]).

There are several reasons which explain why the SVD has become a so useful decomposition in numerical linear algebra:

1. It exists for any $m \times n$ complex matrix (although we focus our attention in real matrices since an image can be represented by real matrices).

2. The SVD is achieved by unitary matrices.

3. Singular values of a matrix depend nicely from the matrix and they allow us to determine efficiently the numerical rank of the matrix.

4. There are powerful algorithms to compute the SVD.

5. Important insight about the matrix can be gained using the SVD as well as optimal low-rank approximations to the matrix.

Some applications, which employ the SVD, include determining the rank, range and null space of a matrix, matrix approximation, computing the pseudoinverse, least squares fitting of data, linear inverse problems, and signal processing.

The theory of diagonalizing a real symmetric matrix is part of a first linear algebra course, but it is not the case of the SVD. Nevertheless, as we shall see, the SVD is closely related to the eigenvalue decomposition of a symmetric matrix. The existence of the SVD of an $m \times n$ real matrix A ($A \in \mathbb{R}^{m\times n}$) means that there are orthonormal basis for $\mathbb{R}^n$ and $\mathbb{R}^m$, specifically suited to each matrix, such that A has a diagonal representation relative to these basis. In opposition to this result, it is a well-known fact that, in general, not every $A \in \mathbb{R}^{n\times n}$ has an eigenvalue decomposition.

Our purpose is to introduce the SVD from some of its applications in digital image processing to students who have received a first course in linear algebra and have some basic familiarity with MATLAB. We shall make extensive use of MATLAB notation along the paper. We encourage the reader to get information about the different MATLAB's commands with the command *help*. We give MATLAB simplified functions for the different applications in image processing presented in this paper, which can be downloaded from *http://www.uv.es/pastorv/eJMT/*.

The paper is organized as follows. In Section 2 we give a short introduction to handling of digital images with MATLAB. The least significant bit method for digital information hiding is described in Section 3. The SVD is introduced in Section 4 as a consequence of well-known symmetric eigenvalue problem. A SVD-based scheme digital watermarking is shown in Section 5. In Section 6 we analyze the problem of low rank approximation of a matrix and we introduce an image compression scheme. In the last section, we deal with the problem of restoring a blurred and noised image. The truncated singular value decomposition (TSVD) approach is presented.

We finish this section by introducing basic notation for matrices. The entries of $A \in \mathbb{R}^{m\times n}$ are designated by $A(i,j)$, $1 \le i \le m$, $1 \le j \le n$. A handy way to specify a column or a row is the "colon" notation. Thus, $A(:,j)$ designates the jth column of A, i.e.,

$$A(:,j) := [A(1,j); A(2,j); \ldots; A(m,j)] \ \ (1 \le j \le n),$$

and the ith row of A is designed by

$$A(i,:) := [A(i,1), A(i,2), \ldots, A(i,n)] \ \ (1 \le i \le m).$$

Thus, we can describe A by columns or by rows:

$$A = [A(:,1), A(:,2), \ldots, A(:,n)] = [A(1,:); A(2,:); \ldots; A(m,:)]\,.$$

We shall identify $\mathbb{R}^n$ with $\mathbb{R}^{n\times 1}$. Moreover, $A \in \mathbb{R}^{m\times n}$ can be identify with the vector $A(:) \equiv [A(:,1); A(:,2); \ldots; A(:,n)] \in \mathbb{R}^{mn}$, and reciprocally, any vector $v \in \mathbb{R}^{mn}$ can be interpreted as an element of $\mathbb{R}^{m\times n}$ (MATLAB's command *reshape* has this purpose).

2 Managing images with MATLAB

In this section basic MATLAB commands to manipulate images are introduced through a concrete example. We assume that MATLAB's *Image Processing Toolbox* is available.

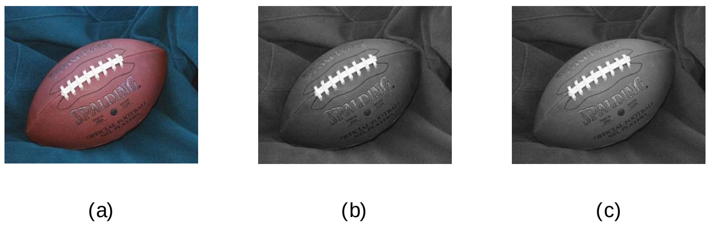


Figure 1: (a) The original color image football; (b) The grayscale image obtained from the green intensity; (c) The grayscale image obtained by means of command rgb2gray applied to the original color image.

As we have installed MATLAB 7.0 in drive C, then in

C:\MATLAB7\toolbox\images\imdemos

we can find some sample images. We have chosen as example the image called *football* which is in *JPEG* format. By means of the command *imshow* we can display the images

>> imshow('C:\MATLAB7\toolbox\images\imdemos\football.jpg')

In this case we can execute just command imshow('football.jpg') since MATLAB knows the path file. To work with our own images, it is more comfortable to change the current directory in MATLAB to avoid writing each time the complete path file.

The command to read the image and to store it in a variable A is

>> $A = imread('football.jpg')$;

By executing command $size(A)$ we get that in A we have a multidimensional array of size $256 \times 320 \times 3$, since A represents a color image in RGB (Red-Green-Blue) format. The array A can viewed as 3 matrices of the same size, 256×320, which represent the intensity in each pixel of the three primary colors. For example, $A(10, 50, 2)$ represents the green intensity of pixel in position $(10, 50)$. Grayscale images are represented by two-dimensional arrays or matrices.

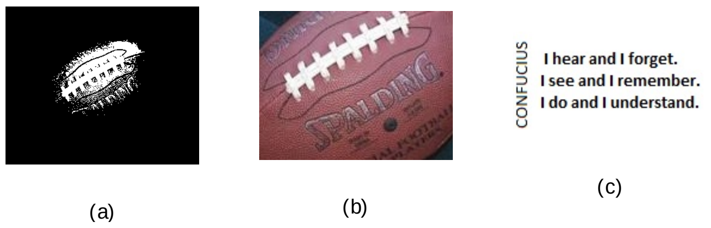
If color is not an important feature, then *rgb2gray* can be used to change a color image into a grayscale image (B=*rgb2gray(A);).* We can also get a grayscale image from the original by $C = A(:, :, 2)$, for instance. Of course B and C represents different images although both have size 256×320. In Figure 1 the three images we have mentioned are shown by using command *subplot.*

The command *imwrite* allows us to write the grayscale picture B in the file footballbn.jpg

>> $imwrite(B,' footballbn.jpg')$

By executing $class(A)$ we realize that the elements in A are of type *uint8*, which means that they are unsigned 8-bit integers, and therefore, they belong to the set $[0, 255] \cap \mathbb{Z}$ (for example, $A(10, 50, 2)$ is 89). It is important to keep in mind that entries in the image arrays belong to a fixed interval and that after performing some arithmetic operation with this kind of values the result could fall outside of the interval, and this could produce an unexpected result. Thus, if we want to operate on image A with mathematical methods, we must convert A to the class double of real numbers in double precision by means of $double(A)$. Of course, to display the image obtained after some algebraic operation or to write it in a file, we must previously convert it to uint8 class by the command with the same name.

A *binary image* is a digital image that has only two possible values for each pixel, usually 0



CONFUCIUS
I hear and I forget.
I see and I remember.
I do and I understand.

Figure 2: (a) A binary version of football image; (b) A portion of original image football.jpg; (c) Image with a Confucius' proverb

(black) or 1 (white). One way to get a binary image from a grayscale image is to use a threshold level, for example 128. By

$>> C = B > 128;$

we convert the grayscale image in B to a binary image by replacing all pixels in the input image with value greater than 128 with the logical value 1 and replaces all other pixels with the logical value 0 (see also command *im2bw*). We show the binary image C in Figure 2.

The Windows program Paint can be used to get our own test images for example with some text in it or a cutting of a small area of an existing image (the same can be done with command *imcrop*). This is also shown in Figure 2.

3 Digital steganography: Hiding information in digital images

Steganography is the art of writing hidden messages in such a way that no one, apart from the sender and intended recipient, suspects the existence of the message. In digital steganography, digital data, as plain text or an image, are embedded inside of a host file, such as an innocent-looking holiday image. The image obtained after hiding some information in a host image is called a stegoimage. In any case, the digital data to be hidden is nothing else than a string of bits, since by ASCII code each written character has a representation as a uint8 number (8 bits). For example, symbol @ is 64 in ASCII and therefore in binary system with 8 bits is 01000000, while letter G is 71 in ASCII which is finally represented by the byte 01000111. The following MATLAB function converts text into a binary sequence.

```
function r=string2bin(q)
% q is a string of plain text; for example 'Mens sana in corpore sano'
% r is a vector representing q as a sequence of zeros and ones
r=uint8(q); % To get the ASCII code of each character in q
r=dec2bin(r,8); % To convert integers between 0 and 255 to binary with 8 bits
r=r';r=r(:); % To get a vector of characters zeros and ones representing q
r=uint8(r==49); % To convert each character to integers 0 and 1
```

Least significant bit (LSB) insertion is a common, simple approach to embedding information in a cover file. Basically, LSB consists of using the least significant bit of each unsigned integer of 8 bits we have in a digital image to hide a bit of the message to be hidden. For example, we hide bit 0 in byte 11000111 by changing it to byte 11000110. The human eye is not able to discern this change in the intensity of a pixel. The MATLAB function *stega* embeds a binary sequence into a grayscale image.

```
function  A=stega(A,r)
% A grayscale host image
% r vector of 0's and 1's to hide into A(:) by means of LSB
[m,n]=size(A); l=length(r);
if l<=m*n % We check that the image is large enough to hide r
    i=1;
    while i<=l
        A(i)=bitset(A(i),1,r(i)); % To change the least significant bit
        i=i+1;
    end
else
    error('Not enough space to embed the message.')
end
```

To extract the hidden message we use the function *undo_stega*.

```
function  q=undo_stega(A)
% A stegoimage with a message hidden by LSB method in A(:)
% q extracted message
c=2*ones(1,8)).^[7:-1:0]; % To change basis
i=1; j=1;
while i+7<=prod(size(A))
    r=bitget(A(i:i+7),1); % To get last bit of each 8 entry set
    q(i)=sum(r.*c); % Conversion from binary with 8 bit to integer
    i=i+8; j=j+1;
end
q=char(q); % Integer to character conversion. The message is at the beginning
```

By combining our functions *string2bin* and *stega* we hide the text ”Mens sana in corpore sano” in the red intensity of a apparently 'innocent' RGB image with LSB method (see Figure 3). The extracted text by our function *undo_stega* is exactly the original one.

Unfortunately, LSB is vulnerable to even a slight image manipulation. For example, if we add 1% Gaussian noise to the stegoimage by means of

$$\begin{aligned} E &= randn(size(A)); \% \ A \text{ is the stegoimage} \\ A &= uint8(double(A) + 0.01 * E * norm(double(A(:)))/norm(E(:))); \end{aligned}$$

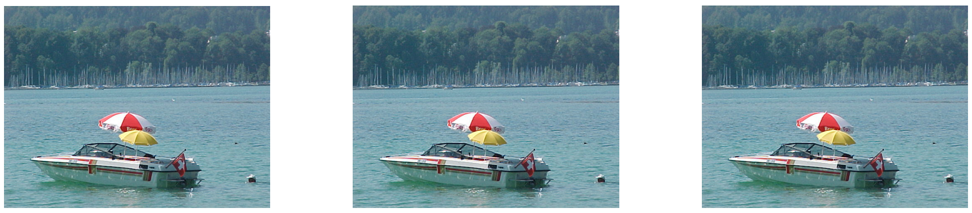


Figure 3: (a) Host image; (b) Stegoimage with a hidden message in the red intensity; (c) Image obtained by adding 1% Gaussian noise to the stegoimage.

where the command *norm* gives the Euclidean norm of a vector, and then we try to recover the hidden message, we get a text with no relationship with the original one. The beginning of the extracted text in this case could be the string 'C } ø Ü ˆZ ,', for instance.

In Section 5 we shall introduce a SVD-steganography scheme which is shown to be more sturdy to general image processing.

4 The singular value decomposition

The SVD first appeared in a paper by Eugenio Beltrami (1873) in the context of bilinear forms. His proof of the existence of the decomposition shows that it is closely related with the eigenvalue decomposition of the symmetric semidefinite positive A^TA (or AA^T), where A^T is the transpose of A. The interested reader in the historical development of the SVD can find much more details in the nice paper [7].

Recall that if $A \in \mathbb{R}^{n\times n}$ is symmetric, there is a $P \in \mathbb{R}^{n\times n}$ orthogonal matrix ($P^TP = I_n$, where I_n is the identity matrix), such that $D := P^TAP$ is diagonal. In others words, $\{P(:,i)\}_{i=1}^n$ is an orthonormal basis of eigenvectors of A for $\mathbb{R}^n$, and the diagonal entries of D are the eigenvalues of A. It is easy to show that not every real square matrix admits a so nice kind of basis from which the linear transformation represented by A has a diagonal matrix representation, the simplest possible form. Surprisingly, for every linear transformation from $\mathbb{R}^n$ to $\mathbb{R}^m$ there are suitable orthonormal basis in the initial and the final spaces, usually different even when $n = m$, from which the linear transformation has a diagonal matrix representation. This is nothing else than a SVD of a $m \times n$ real matrix.

Definition 1 *The singular value decomposition of $A \in \mathbb{R}^{m\times n}$, for short SVD, is a factorization of the form*

$$A = U\Sigma V^T,$$

where $U \in \mathbb{R}^{m\times m}$ and $V \in \mathbb{R}^{n\times n}$ are orthogonal, and $\Sigma \in \mathbb{R}^{m\times n}$ is diagonal such that, if $diag(\Sigma) = [\mu_1; \mu_2; \ldots; \mu_p]$, $p := \min\{m, n\}$, then

$$\mu_1 \geq \mu_2 \geq \cdots \geq \mu_p \geq 0.$$

The diagonal entries of Σ are known as the singular values of A. The columns of U, $u_i := U(:,i)$ $(1 \leq i \leq m)$, and the columns of V, $v_i := V(:,i)$ $(1 \leq i \leq n)$, are called left-singular vectors and right-singular vectors of A, respectively.

Let us suppose that a SVD of A exists. Then, it is not hard to get the SVD expansion

$$A = \sum_{j=1}^{s} \mu_j u_j v_j^T, \tag{1}$$

where $s \in \{0, 1, \ldots, p\}$ is the number of non-zero singular values. This expansion will lead us to a SVD-based image compression scheme in Section 6.

Moreover, from (1), we easily obtain

$$Av_k = \mu_k u_k \ \ (1 \leq k \leq s); \ Av_k = 0 \ \ (s+1 \leq k \leq n), \tag{2}$$

and

$$A^T u_k = \mu_k v_k \ \ (1 \leq k \leq s); \ A^T u_k = 0 \ \ (s+1 \leq k \leq m).$$

Thus

$$A^T A v_k = \mu_k^2 v_k \ \ (1 \leq k \leq s); \ A^T A v_k = 0 \ \ (s+1 \leq k \leq n),$$

which implies that the left-singular vector v_k are eigenvectors of $A^T A$, and that singular values are the positive square roots of the p-first, in decreasing order, eigenvalues of $A^T A$. This fact suggests a proof of the existence of the SVD for any matrix based in the eigenvalue decomposition of $A^T A$. The matrix $A^T A \in \mathbb{R}^{n \times n}$ is symmetric and positive-semidefinite. Consequently, there exists an orthonormal basis for $\mathbb{R}^n$ of eigenvectors associated to $A^T A$, being the eigenvalues of $A^T A$ non-negative.

Let $\{\mu_j^2\}_{j=1}^n$, $\mu_j \geq 0$ $(1 \leq j \leq n)$, $\mu_1 \geq \mu_2 \geq \cdots \geq \mu_n \geq 0$, the eigenvalues of $A^T A$. Let $\{v_j\}_{j=1}^n$ a orthonormal basis for $\mathbb{R}^n$ with $A^T A v_j = \mu_j^2 v_j$ $(1 \leq j \leq n)$. Of course, we get the right-singular vectors from (2), i.e., $u_j := Av_j/\mu_j$ $\ (1 \leq j \leq s)$. The key point to complete the proof of the existence of a SVD of A is to show that $\{u_j\}_{j=1}^s$ is an orthonormal set, but a simple calculation shows that $u_i^T u_j = \delta_{ij}$ $\ (1 \leq i, j \leq s)$. In case $s < m$, we complete $\{u_j\}_{j=1}^s$ to an orthonormal basis $\{u_j\}_{j=1}^m$ of $\mathbb{R}^m$, even though, as the SVD expansion shows, the added vectors $\{u_j\}_{j=s+1}^m$ are only necessary to complete the dimension, but they make no contribution to A. Recall that this completion can be done by means of the Gram-Schmidt process.

In short, we have outlined the proof of the following result.

Theorem 2 *Every rectangular real matrix has a SVD.*

It is worth to point out that also complex matrices have a SVD, but in this case U and V are unitary transformations.

Example 3 *Let us obtain a SVD of $A = [0, 1; -1, 0; -1, 1] \in \mathbb{R}^{3 \times 2}$ by the process followed to prove the existence of a SVD of A. We have $A^T A = [2, -1; -1, 2]$ with 3 and 1 as eigenvalues with eigenvectors $[-1; 1]$ and $[1; 1]$, respectively. Thus, singular values are $\mu_1 = \sqrt{3}$ and $\mu_2 = 1$, and we take $v_1 = [-1; 1]/\sqrt{2}$ and $v_2 = [1; 1]/\sqrt{2}$. As $Av_1 = [1; 1; 2]/\sqrt{2}$ and $Av_2 =$*

$[1;-1;0]/\sqrt{2}$, *we define* $u_1 = [1;1;2]/\sqrt{6}$ *and* $u_2 = [1;-1;0]/\sqrt{2}$. *Finally, we complete* $\{u_j\}_{j=1}^2$ *to an orthonormal basis for* $\mathbb{R}^3$. *In this simple case, we can take* $u_3 = [1;1;-1]/\sqrt{3}$ *since it is orthogonal to* u_1 *and* u_2, *and its Euclidean norm is* 1. *A simple calculation shows that* $A = U\Sigma V^T$.

From the numerical point of view, it is not suitable to get a SVD of A by forming A^TA, since the errors in the calculation of A^TA in limited precision can lead to a severe loss of information. For example, if we consider the invertible matrix $A = [1, 1; 0, 10^{-8}]$, then when we form the matrix A^TA with MATLAB we get the singular matrix $ones(2)$, which gives 0 as smallest singular value instead of $10^{-8}/\sqrt{2}$. Thus, a maximum absolute error of 10^{-16} in the calculation of entries of A^TA gives an absolute error in the calculation of μ_2 of $10^8\sqrt{2}$ times bigger. The interested reader in computational issues regarding the SVD can find more details in [2, Section 8.6]. In MATLAB we can use the command *svd* to calculate a SVD of a matrix.

From the previous analysis it is clear that the singular values are uniquely determined but in general the SVD is not unique. Nevertheless, in the case of distinct singular values, the SVD is unique up to sign of the columns of U and V.

A SVD of a matrix provides of a good insight about its structure. Among other information, it is not hard to show that it provides us with the main two subspaces related to A

$$range(A) = \langle \{u_i\}_{i=1}^s \rangle, \ \ker(A) = \left\langle \{v_i\}_{i=s+1}^n \right\rangle,$$

their orthogonal complements

$$range(A)^\perp = \left\langle \{u_i\}_{i=s+1}^m \right\rangle, \ \ker(A)^\perp := \langle \{v_i\}_{i=1}^s \rangle,$$

and the rank of the matrix $rank(A) := \dim(range(A)) = s$.

5 A SVD-based digital steganography scheme

In this section we go back to digital steganography, by introducing a block SVD technique to embed binary information into the biggest singular value of each block of a grayscale host image by a quantization process ([9]), which can resist general image processing better than the LSB method. We show here the simplified version proposed in [6].

The host image is split into non-overlapped blocks of size 2×2. For simplicity, let S be one of these blocks. We get a SVD of S: $S = U\Sigma V^T$. Then write $\mu_1 \equiv \mu_1(S) = c * 10 + r$ with c a non-negative integer and $0 \le r < 10$. For an embedded watermark bit valued of 0, if c is even define $c^* := c$ and otherwise $c^* := c - 1$. Thus, in this case the final value of c^* is even. For an embedded watermark bit valued of 1, if c is odd define $c^* = c$ and otherwise $c^* := c + 1$. Therefore, in this case the final value of c^* is odd. Finally, define $\mu_1^* := c^*10 + 5$, $\mu_2^* := \mu_2$, and replace S by $S^* = U\,diag([\mu_1^*; \mu_2^*])V^T$ in the same position in the original image to get the stegoimage. The MATLAB function *stega_SVD* implements this process.

```
function   A=stega_SVD(A,r)
% A grayscale host image
% r vector of '0's and '1's to be hidden into A
```

```
[m,n]=size(A); s=floor(n/2); t=floor(m/2); l=length(r);
if s*t<l % We check that the image is large enough to hide r
    error('Not enough space in host image')
else k=1;
    for i=1:2:2*t-1
        for j=1:2:2*s-1
            [U,S,V]=svd(double(A(i:i+1,j:j+1)));
            c=floor(S(1,1)*0.1);
            if r(k)==0
                if c/2~=floor(c/2)
                    c=c-1;  % Thus, c is even
                end
            else
                if c/2==floor(c/2)
                    c=c+1;  % Thus, c is odd
                end
            end
            S(1,1)=c*10+5;
            A(i:i+1,j:j+1)=uint8(U*S*V');
            k=k+1;
            if k==l+1
                return  % r is already hidden into A
            end
        end
    end
end
```

To recover the hidden binary information we give the following code.

```
function  r=undo_stega_SVD(A)
% r vector containing the binary info hidden in the stegoimage A
[m,n]=size(A); A=double(A); s=floor(n/2); t=floor(m/2); k=1;
for i=1:2:2*t-1
    for j=1:2:2*s-1
        S=svd(double(A(i:i+1,j:j+1))); % vector with the singular values
        c=floor(S(1,1)*0.1);
        if c/2==floor(c/2) % c is even
            r(k)=0;
        else
            r(k)=1;
        end
        k=k+1;
    end
end
```

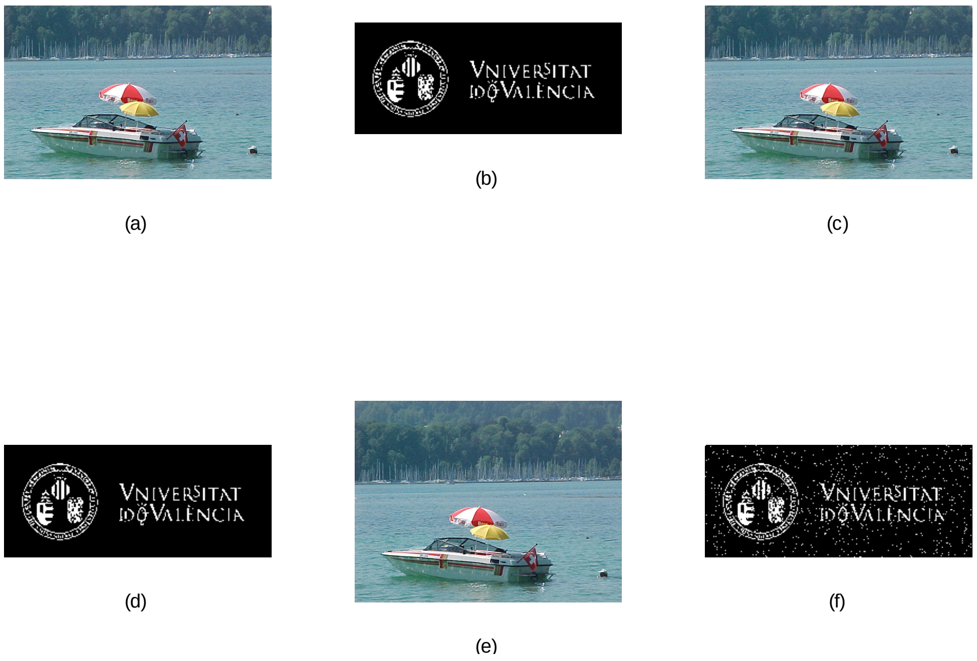


Figure 4: (a) Host image yacht; (b) Binary watermark; (c) Watermarked image; (d) Extracted watermark (correlation 1); (e) Watermarked image attacked with 2% additive Gaussian noise; (f) Extracted watermark from image (e) (correlation 0.8345).

As the reader can check, this technique based in SVD is able to resist image processing, as for example Gaussian noise addition, better than LSB insertion method.

In Figure 4 we show the results of applying this scheme to watermark the red intensity of yacht color image with a binary version of logo of University of Valencia (https://www.uv.es) obtained by applying command *im2bw*. Of course, we could hide more than one copy of the watermark in the three intensities of the color host image and thus we could get more extracted watermarks. In this case we would keep the most correlated with the original watermark of all them. Of course, to extract the watermark we need to know its size. The extracted watermark under a previous attack of the watermarked image with the addition of 1% Gaussian noise is exactly the original one. Nevertheless, with 2% Gaussian noise the result is not so good. We compare the extracted watermark and the original one by calculating the correlation coefficient between both vectors.

6 Reduced rank approximations of a matrix and image compression

The SVD expansion (1) suggests a way to approximate a matrix and save storage. To form A by using (1) we have to store $\{\mu_i, u_i, v_i\}_{i=1}^{s}$, i.e., $s(m+n+1)$ real numbers instead of the original mn entries of A. Clearly, this is not always an advantage. However, one could guess that by summing in (1) to a positive index $k < s$, we possibly could obtain a good approximation of A, resulting in saving of storage space.

In order to compare matrices we need a notion of distance between matrices. The notion of matrix norm is completely analogue to the notion of vector norm and provides a measure of distance on the space of matrices (for more details see [2] or [4]). Probably, the most used matrix norms in numerical linear algebra are the *Frobenius norm* which is defined as

$$\|A\|_F := \|A(:)\|_2 = \left(\sum_{j=1}^{n} \|A(:,j)\|_2^2\right)^{1/2} = trace(A^TA)^{1/2} \quad (A \in \mathbb{R}^{m\times n}),$$

where $\|x\|_2 := (x^Tx)^{1/2}$, $x \in \mathbb{R}^k$, and the *spectral norm*,

$$\|A\|_2 := \max_{\|x\|_2=1} \|Ax\|_2 \quad (A \in \mathbb{R}^{m\times n}).$$

The Frobenius norm has the advantage of being easily computed from entries of the matrix, while the spectral norm is based in the interpretation of A as a linear transformation and we will show that it coincides with the first singular value of A (the spectral norm belongs to the family of subordinated matrix norm or operator norm). Nevertheless, the spectral norm is also easy to compute for some special kind of matrices. For example, if $A \in \mathbb{R}^{m\times n}$ is diagonal ($A(i,j) = 0$ when $i \neq j$), then

$$\|A\|_2 = \max_{1\leq i\leq \min(m,n)} |A(i,i)|\,.$$

Of course, both matrix norms are equivalent. In fact, it is not difficult to show that

$$\|A\|_2 \leq \|A\|_F \leq \sqrt{rank(A)}\,\|A\|_2 \quad (A \in \mathbb{R}^{m\times n}).$$

As it is easily verified, both matrix norms satisfy very important properties. Both norms are *submultiplicative*

$$\|AB\|_t \leq \|A\|_t \|B\|_t \quad (A \in \mathbb{R}^{m\times n},\ B \in \mathbb{R}^{n\times s},\ t = 2, F),$$

and both are *invariant with respect to orthogonal transformations:* If $A \in \mathbb{R}^{m\times n}$ and $P \in \mathbb{R}^{m\times m}$ and $Q \in \mathbb{R}^{n\times n}$ are orthogonal, then

$$\|PAQ\|_t = \|A\|_t \quad (t = 2, F).$$

From this property we have

$$\|A\|_2 = \mu_1 = \left\|A^T\right\|_2,\ \|A\|_F^2 = \sum_{i=1}^{p} \mu_i^2,$$

showing that the relation of singular values with the spectral norm is more direct. Moreover, if $m = n$, then, A is invertible if, and only if, $\mu_n > 0$, and in this case $\|A^{-1}\|_2 = \mu_n^{-1}$. The important number in perturbation theory of simultaneous linear equations, called *spectral condition number* of a invertible matrix A, $k_2(A) := \|A\|_2 \|A^{-1}\|_2$, coincides with μ_1/μ_n.

As it is easily checked, $\left\| u_j v_j^T \right\|_2 = \|u_j\|_2 \|v_j\|_2 = 1$, $1 \le j \le s$. Thus, we could say that the first addends of the SVD expression of A have more contribution in forming A, due to the selected order of the singular values.

In order to shorten the paper, we omit the proof of the following low rank approximation theorem ([2, Theorem 2.5.3]). A similar result for the Frobenius norm can be found in [7].

Theorem 4 *Let $A \in \mathbb{R}^{m\times n}$, with $rank(A) = s \ge 1$. Let $A = U\Sigma V^T$ be a SVD of A. Consider*

$$A_k := \sum_{j=1}^{k} \mu_j u_j v_j^T \quad (1 \le k \le s-1). \tag{3}$$

Then

$$\mu_{k+1} = \|A - A_k\|_2 = \min\left\{ \|A - B\|_2 : B \in \mathbb{R}^{m\times n},\ rank(B) \le k \right\} \quad (1 \le k \le s-1).$$

As a straightforward consequence we have the condition number is a measure of the distance of the matrix to the set of singular matrices.

Corollary 5 *Assume that $A \in \mathbb{R}^{n\times n}$ is invertible. Then*

$$\frac{1}{k_2(A)} = \min\left\{ \frac{\|A - B\|_2}{\|A\|_2} : B \in \mathbb{R}^{n\times n} \text{ is singular} \right\}.$$

With no doubt, one of the main consequences of the approximation theorem is the nice dependence of singular values from the matrix. Thus, the SVD can be used as a numerically reliable estimate of the effective rank of a matrix (see [2] for the notion of numerical rank). The MATLAB command *rank,* to calculate the rank of a matrix, is based on the SVD.

Corollary 6 *Let A, $B \in \mathbb{R}^{m\times n}$ and let $p = \min\{m, n\}$. Then*

$$|\mu_k(A) - \mu_k(B)| \le \|A - B\|_2 \quad (1 \le k \le p).$$

Let us then to introduce a image compression technique based on the idea outlined at the beginning of the section: A_k is considered as an approximation of A with $k \in \{0, \ldots, s = rank(A)\}$ chosen by the user. The relative error in the spectral norm is

$$er := \frac{\|A - A_k\|_2}{\|A\|_2} = \frac{\mu_{k+1}}{\mu_1},$$

and the ratio of compression is

$$rc := \frac{k(m+n+1)}{mn},$$

since only the k first singular values and the k first left and right singular vectors are needed to form A_k from (3). The following MATLAB function gives A_k, er and rc, from arguments U, S, V, which represent a SVD of the image, and k.

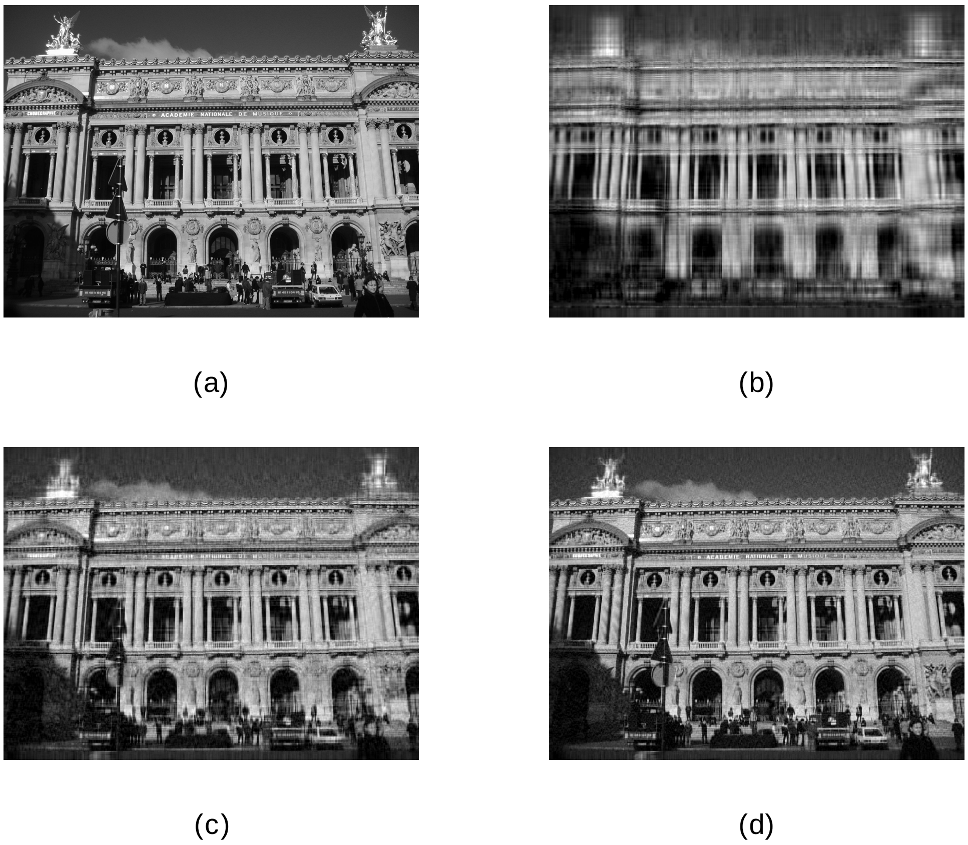


Figure 5: (a) Original Image; (b) k=10, er=0.0661, rc=0.0103; (c) k=50, er=0.0242, rc =0.0514; (d) k=100, er=0.0145, rc=0.1027.

```
function [B,er,rc]=lr_aprox(U,S,V,k)
[m,n]=size(S);
er=S(k+1,k+1)/S(1,1);
rc=k*(m+n+1)/(m*n);
B=uint8(U(:,1:k)*S(1:k,1:k)*V(:,1:k)');
```

As an example we have chosen a 1704×2272 grayscale image and the results are shown in Figure 5. With an approximation of rank $k = 100$ one can recognize the main features of the original image. For example, the place where the picture was taken is easily identified: The Academie Nationale de Musique (Paris). In this case, only around 10% of the original storage space is needed. This scheme can be applied to color images by compressing each intensity of primary color separately and then inserting them in a new multidimensional array.

7 A simple image deblurring problem

Different reasons can cause a recorded image to not represent an ideal scene that is sharp and useful, including an out-of-focus camera lens, atmospheric turbulences, movement of the camera or the object while the shutter is open, etc. Thus, it is unavoidable that scene information spreads to neighboring pixels and hence, the result is that we record a blurred image. If we know the blurring process, then we can try to recover some details about the original image hidden in the blurred image. This matter is usually referred to as the image deblurring problem.

The general issues of these kind of problems can be found in [5] or in the excellent book [3], where the interested reader can expand the basic model we present in this section.

To start, we consider vertical motion blur under zero boundary conditions. Each pixel intensity spreads over uniformly to r vertical adjacent pixels. To describe this process, we look for a matrix A_v such that, if $z = A_v x$, then $z(i)$ is the arithmetic mean of r components of vector x adjacent to component $x(i)$ by both sides. In our model we assume the simplest boundary condition, in which one supposes that the exact image is black (zero pixels) outside the boundary. Here it is the code to get $A_v \in \mathbb{R}^{n\times n}$ from an odd r.

```
 function A=blur_vertical(n,r)
% r is chosen odd by simplicity; n is the size of the matrix A
% zero boundary condition
A=ones(n)/r; s=(r-1)/2; A=tril(triu(A,-s),s);
```

If $X \in \mathbb{R}^{m\times n}$ represents the ideal grayscale scene that we are trying to capture and $B \in \mathbb{R}^{m\times n}$ represents the recorded image under vertical motion blur, they are related by the equation $B = A_v X$, where $A_v \in \mathbb{R}^{m\times m}$ has been described before. We can also modelize horizontal motion blur by multiplying X to the right by $A_h \in \mathbb{R}^{n\times n}$, where A_h follows the same pattern of A_v but likely associated to a different r. Thus, the final relation to describe both motion blur in the recorded image is

$$B = A_v X A_h, \tag{4}$$

where A_v and A_h modelize vertical and horizontal motion blur, respectively. To rewrite this product of matrices as a matrix by vector product we use the *Kronecker product* which for $S \in \mathbb{R}^{n\times n}$ and $T \in \mathbb{R}^{m\times m}$ is the matrix $S \otimes T \in \mathbb{R}^{mn\times mn}$ given by

$$S \otimes T := \begin{bmatrix} S(1,1)T & \dots & S(1,m)T \\ \vdots & \vdots & \vdots \\ S(m,1)T & \dots & S(m,m)T \end{bmatrix}.$$

It is not hard to show that

$$(S \otimes T)X(:) = (TXS^T)(:) \quad (X \in \mathbb{R}^{m\times n}).$$

Therefore, (4) can be written as

$$Ax_{exact} = b_{exact},$$

where $b_{exact} := B(:)$, $A := A_h^T \otimes A_v \in \mathbb{R}^{mn\times mn}$ is the blurring matrix and $x_{exact} := X(:)$.

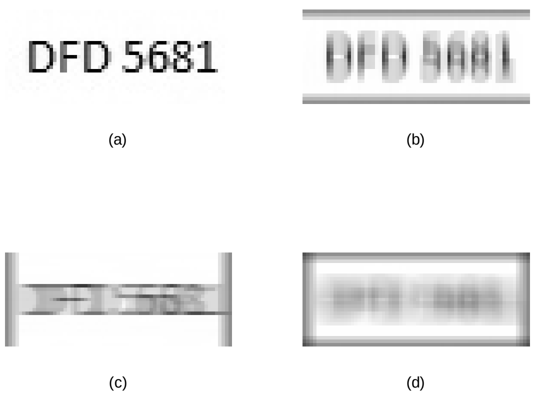


Figure 6: (a) Original Image; (b) Vertical motion blur with $r = 7$; (c) Horizontal motion blur with $r = 9$; (d) Simultaneous vertical ($r = 7$) and horizontal ($r = 9$) motion blur.

Here we include a MATLAB function to get the Kronecker product of two matrices.

```
function A=kronecker(S,T)
[m,s]=size(S); [n,s]=size(T); C=[ ]; A=[ ];
for j=1:m
    for i=1:m
        C=[C;S(i,j)*T];
    end
    A=[A,C]; C=[ ];
end
```

In Figure 6 (d) there is an example of motion blur applied to a grayscale image of size 27×66. Observe the presence of a black boundary consequence of zero boundary condition in our model. The blurring matrix A is square of size $N = 1782$ and, although it is invertible, it is a badly conditioned for $\mu_1/\mu_N = 3.0823e + 018$, which alerts us that a system of linear equations with A as the coefficient matrix is very ill-conditioned; that is, the solution of the system is severely sensitive to small changes in the input data.

Hence, in the abstract scheme we have a large blurring matrix $A \in \mathbb{R}^{N\times N}$, $b_{exact} \in \mathbb{R}^N$ as the blurred image, and $x_{exact} \in \mathbb{R}^N$ as the original image, not available, related by the linear model

$$Ax_{exact} = b_{exact}.$$

In Figure 7 we show the result of solving this system in our example by standard techniques.

In addition to blurring, the recorded images are usually contaminated with noise due to the use of a mechanical device to capture images and to digitalization. Let $E \in \mathbb{R}^{m\times n}$ be the noise image and define $e := E(:)$. Then, the noisy blurred image is represented by vector $b = b_{exact} + e$. Accordingly, in practice we have to solve the linear very ill-posed perturbed problem

$$Ax = b \tag{5}$$

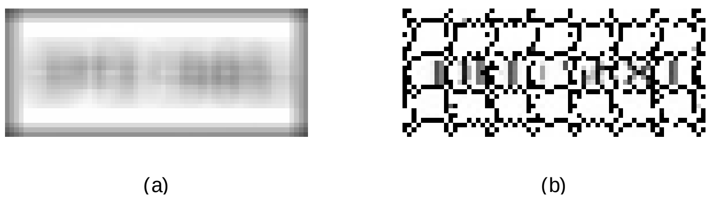


Figure 7: (a) Blurred image; (b) Theoretical solution $A^{-1}b_{exact}$

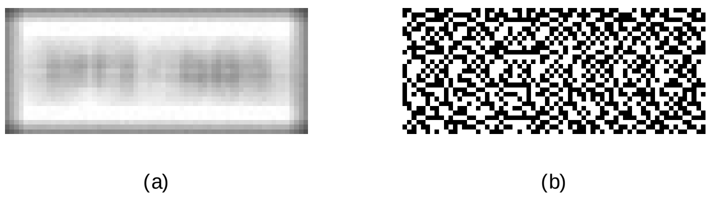


Figure 8: (a) Available blurred and noise-contaminated image (1% Gaussian noise); (b) Naive reconstruction.

in order to recover the main features of the ideal scene.

As A is invertible, the theoretical solution of the perturbed system of equations can be easily obtained from a SVD of A. Indeed, if $A = U\Sigma V^T$ is a SVD, then the solution of (5) is

$$x_{naive} := A^{-1}b = V\Sigma^{-1}U^T b = \sum_{i=1}^{N} \frac{u_i^T b}{\mu_i} v_i = x_{exact} + \sum_{i=1}^{N} \frac{u_i^T e}{\mu_i} v_i,$$

being the last term the inverted noise contribution to the solution.

It is interesting to note that for rank deficient matrix A, the solution of (5) in the sense of least squares, i.e., which minimizes the function $x \in \mathbb{R}^N \to \|Ax - b\|_2^2$, of minimum Euclidean norm, is given by the SVD expansion $\sum_{i=1}^{rank(A)} \mu_i^{-1} u_i^T b \; v_i$ ([2, Theorem 5.5.3]).

In Figure 8 we have the naive solution associated to the image obtained by adding 1% Gaussian noise to our motion blurred image. It is beyond the scope of this paper to go further into the underlying theory which explains why, even without noise, we get a so bad result. Roughly speaking, the naive solution is completely dominated by the error term due to the SVD components corresponding to the smallest singular values, making it useless. Since the singular values are ordered decreasingly, we can remove the smallest ones by truncation. The resulting method is called *truncated singular value decomposition* (TSVD), where, for $1 \leq k \leq N$, the associated approximate solutions of (5) are given by

$$x_k := \sum_{i=1}^{k} \frac{u_i^T b}{\mu_i} v_i.$$

The following MATLAB code gives truncated solutions.

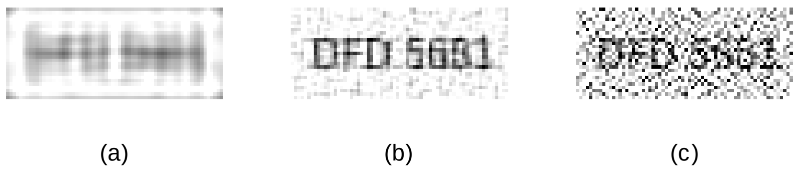


Figure 9: Several TSVD solutions: (a) k=100; (b) k=600; (c) k=1300.

```
function  x=tsvd(U,S,V,b,k)
% A=U*S*V' is the blurring matrix
x=V(:,1:k)*diag(diag(S(1:k,1:k)).^(-1))*U(:,1:k)'*b;
```

In Figure 9 we can see three TSVD solutions x_k of our image deblurring and denoising test problem. Observe that as k goes from small to large values, the corresponding solutions range from oversmoothed to undersmoothed, being (b) the reconstruction visually more pleasant.

Of course, the regularization parameter choice is the crucial and more delicate point in regularization methods. When a good estimate of the norm of the noise, $\|e\|_2 \approx \delta$, is available, the *discrepancy principle* suggests that the selected parameter k in TSVD should satisfy

$$\|Ax_k - b\|_2 = (\sum_{i=k+1}^{N} |u_i^T b|^2)^{1/2} \leq c\delta,$$

where $c > 1$ is a user-supplied constant ([3]). Thus, the parameter is sought that provides a residual of the same magnitude as the noise in the data. For TSVD the residual norm is monotonically nonincreasing. Thus, we can systematically increase k from 1 to N until the condition is achieved. The code *tsvd_discrep* implements TSVD with truncation parameter selected by discrepancy principle.

```
function [x,k]=tsvd_discrep(U,S,V,b,c,delta)
[m,n]=size(S); r=norm(U'*b); k=0;
while r>c*delta & k<=m
    k=k+1;
    r=sqrt(r^2-(U(:,k)'*b)^2);
end
x=tsvd(U,S,V,b,k);
```

The *Generalized Cross Validation* (GCV) is another parameter choice method. The advantage is that we do not need a priory estimate of noise (see [1] for a detailed analysis of this principle). In GCV for TSVD the truncation parameter is found as the minimum of the discrete function

$$G(k) := \frac{\sum_{i=k+1}^{N}(u_i^T b)^2}{(N-k)^2} \; (1 \leq k < N).$$

Here we include a MATLAB function for this parameter choice criterion.

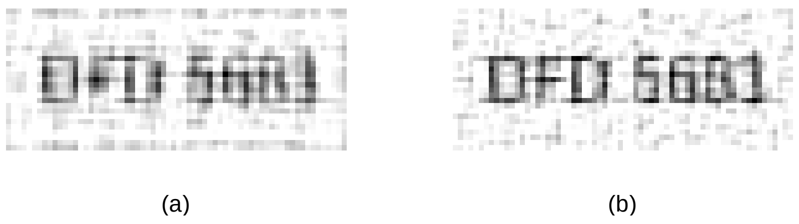


Figure 10: (a) TSVD with discrepancy principle with c=1.01 and delta=0.01*norm(b) (k=450); (b) TSVD with GCV (k=576);

```
function  [x,k]=tsvd_gcv(U,S,V,b)
[m,n]=size(S);
r=norm(U'*b)^2;
for i=1:m-1
    r=r-(U(:,i)'*b)^2;
    c(i)=r/(m-i)^2;
end
[no_use,k]=min(c);
x=tsvd(U,S,V,b,k);
```

The results of applying TSVD to the our noisy blurred test image with both parameter choices are shown in Figure 10.

8 Concluding remarks

These notes are intended to introduce the SVD of a matrix from a practical point of view, showing some of its applications in the field of digital image processing. They are accessible to undergraduate students who have followed the usual topics of a first linear algebra course and have a basic knowledge of MATLAB.

It is worth to highlight that the SVD exists for every complex matrix and provides the most prominent features of the matrix. Moreover, the main properties of the SVD can be exploited in image processing.

In order to make it easier for the interested reader to be able to experiment with the algorithms related with the proposed applications, we have included MATLAB simplified codes which are also available online.

Along the paper it has often been necessary to omit interesting details, but we encourage the reader to consult the references for a more thorough treatment of the SVD and its applications.

Acknowledgements. The authors are very grateful to the referee for valuable comments and corrections.

The second author is indebted to José A. Vallejo for his kind invitation to the Fourth Summer School UVEG-UASLP held at Universidad Autónoma of San Luis Potosí (México). His research was supported by Universitat de Valencia, grants UV-INV EPDI12-67375 and UV-RI_IP11-76088.

References

[1] G. H. Golub, M. Heath and G. Wahba, Generalized cross-validation as a method for choosing a good ridge parameter, *Technometrics*, Vol. 21, n^o 2, 1979.

[2] G. H. Golub, C. F. Van Loan, *Matrix Computations*, Johns Hopkins University Press, Baltimore, MD, Third Edition, 1996.

[3] P. C. Hansen, J. Nagy, and D. P. O'Leary, *Deblurring images. Matrices, Spectra and Filtering*, SIAM, 2006.

[4] R. A. Horn and C. R. Jhonson, *Matrix Analysis*, Cambridge University Press, 1985.

[5] R. C. Puetter, T. R. Gosnell and A. Yahil, Digital Image reconstrution: Deblurring and denoising, *Annu. Rev. Astron. Astrophys.,* 2005, 43:139–194.

[6] A. Rojas, A. Cano, Trabajando con imágenes digitales en clase de Matemáticas, *Gaceta de la Real Sociedad Matematica Española*, Vol. 13, N^o 2, 2010 , págs. 317-336.

[7] G. W. Stewart, On the Early History of the Singular Value Decomposition, *SIAM Review*, Vol. 35, No. 4 (Dec.,1993), 551-566.

[8] G. Strang, The fundamental theorem of Linear Algebra, *American Mathematical Monthly*, 100(9), pp. 848–855, 1993.

[9] R. Sun, H. Sun and T. Yao, A SVD and quantization based semi-fragile watermarking technique for image authentication. *Proc. Internat. Conf. Signal Process.* 2, 2002, 1952–1955.