uOttawa

L'Université canadienne
Canada's university

FACULTÉ DES ÉTUDES SUPÉRIEURES ET POSTOCTORALES

uOttawa

L'Université canadienne
Canada's university

FACULTY OF GRADUATE AND POSDOCTORAL STUDIES

Weibin Qi

AUTEUR DE LA THÈSE / AUTHOR OF THESIS

M.Sc. (Systems Science)

GRADE / DEGREE

Systems Science

FACULTÉ, ÉCOLE, DÉPARTEMENT / FACULTY, SCHOOL, DEPARTMENT

Image Denoising with Spline Interpolation Based on Singular Value Decomposition and Other Evaluation Methods

TITRE DE LA THÈSE / TITLE OF THESIS

R. Vaillancourt

DIRECTEUR (DIRECTRICE) DE LA THÈSE / THESIS SUPERVISOR

CO-DIRECTEUR (CO-DIRECTRICE) DE LA THÈSE / THESIS CO-SUPERVISOR

EXAMINATEURS (EXAMINATRICES) DE LA THÈSE / THESIS EXAMINERS

B. Dionne

Y. Bourgeault

Gary W. Slater

LE DOYEN DE LA FACULTÉ DES ÉTUDES SUPÉRIEURES ET POSTDOCTORALES /
DEAN OF THE FACULTY OF GRADUATE AND POSTDOCORAL STUDIES

IMAGE DENOISING WITH SPLINE INTERPOLATION BASED ON SINGULAR VALUE DECOMPOSITION AND OTHER EVALUATION METHODS

By
Weibin QI, B.Sc.
January 2005

A Thesis
submitted to the Faculty of Graduate and Postdoctoral Studies
in partial fulfillment of the requirements
for the degree of
Master of Science in Systems Science

Your file Votre référence
ISBN: 0-494-11386-3
Our file Notre référence
ISBN: 0-494-11386-3

Abstract

In this thesis, we construct two new algorithms for image denoising, namely, spline and spline-wavelet, which combine spline interpolation and wavelets together with non-linear filtering based on block singular value decomposition. Those two approaches are compared with other existing methods, which involve BlockSvd filter, wavelet (global thresholding) filter, median filter, average filter, and adaptive filter. The performance of these approaches differs little from each other. Generally speaking, median filter is very suitable for processing images to reduce "salt and pepper" noise. But for zero-mean Gaussian and speckle noises, an adaptive filter and spline-wavelet methods are more stable and slightly superior to other filters in most conditions and for most images. The proposed algorithms were tested under different types of images and a wide range of signal to noise ratios (SNR). The numerical results demonstrate that these methods can be used in different and useful ways for reducing image noise.

Key words: Image denoising, singular value decomposition, spline interpolation, wavelet, block decomposition, median filter, average filter, adaptive filter.

Acknowledgements

I would like to express my deep appreciation to my supervisor, Dr. Rémi Vaillancourt, for his academic and financial support, understanding and guidance during the completion of this thesis.

I would also like to thank Dr. Ryuichi Ashino, Dr. Xinhou Hua, Ms. Ding Yiqun and Ms. Jenny Lee for their valuable discussions and suggestions.

Dedication

To my parents, my elder brother and all my friends, for their love, encouragements and unselfish support.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Motivation

Image restoration is a central problem in image processing. Usually, the goal of image restoration is to relieve human observers from this task by reconstructing a plausible estimate of the original image from distorted or noisy observations [15]. However, in most practical applications, image denoising is applied to produce good estimates of the original image from noisy observations [20]. Nowadays, image denoising is becoming a promising subfield of computer vision, which has been extensively studied. In recent years, denoising image data has been an active area of research with several different approaches being proposed using techniques such as wavelet thresholding, bilateral filtering and non-linear filtering based on singular value decomposition (SVD), etc.

1.2 Object

The subject of this thesis falls more in Computer Science than in Mathematics. Mathematical tools are used to the extent they serve the development of the following points.

1. We shall introduce and implement 2D image denoising approaches for nonlinear noise reduction techniques presented in the literature. Specifically, we shall

focus on several approaches, including wavelet (global thresholding) filtering, non-linear image filtering algorithm based on SVD block processing (BlockSvd), two-dimensional adaptive noise-removal filtering, two-dimensional median filtering, and two-dimensional digital average filtering.

2. Furthermore, we shall propose two new 2D image denoising approaches which combine spline interpolation simulation, SVD block processing with wavelet thresholding filtering, culminating in an improved hybrid approach.

3. We shall implement all the algorithms described in this thesis into a demo program developed in the MATLAB environment, which completely executes and fully compares all the approaches to measure their performance studied under different noise patterns and different images in order to arrive at useful conclusions.

This thesis makes heavy use of numeric and graphic MATLAB and several MATLAB toolboxes.

1.3 Thesis organization

The thesis is organized as follows. Chapter 1 includes the motivation, object and thesis organization, Chapter 2 introduces some basic concepts which involve wavelets and wavelet analysis, the advantage of the wavelet 2D decomposition compared with Fourier analysis and describes different image denoising approaches. There are a couple of different approaches presented in this section. Standard objective measures are defined for evaluating noisy and restored images. Chapter 3 proposes two new improved approaches based on singular value decomposition and combined with spline interpolation and wavelet threshold denoising. Furthermore, a comparative study is done of the several other approaches presented in Chapter 2. Finally, major conclusions are given. The source code of my image denoising demo program is listed in the Appendix as this is common practice in Computer Science.

Chapter 2

Basic Concepts

2.1 Purpose

We shall use the MATLAB environment for our numerical simulation. In fact, MATLAB integrates computation, visualization, and programming in an easy-to-use environment where problems and solutions are expressed in familiar mathematical notation.

MATLAB features a family of application-specific solutions called toolboxes. Very important to most users of MATLAB, toolboxes allow one to learn and apply specialized technology. Toolboxes are comprehensive collections of MATLAB functions (M-files) that extend the MATLAB environment to solve particular classes of problems [9]

We shall use the basic numeric and graphic MATLAB and the image processing, signal processing and wavelet toolboxes.

2.2 The wavelet toolbox

The Wavelet Toolbox is a collection of functions built on the MATLAB Technical Computing Environment. It provides tools for the analysis and synthesis of signals and images, and tools for statistical applications, using wavelets and wavelet packets within the framework of MATLAB [10].

In our project, we only focus on using wavelet tools to develop, analyze and study

image denoising. All the implemented programs in this paper were coded, integrated, and implemented in the MATLAB language.

2.3 Wavelet and Fourier analysis

Wavelets, are mathematical functions, which represent data in terms of frequency components. Thus each component is analyzed with a resolution data containing discontinuities and impulse functions. The basic concept is to base data analysis on scale [8].

In wavelet analysis, the scale that we use to look at data plays a special role. Wavelet algorithms process data at different scales or resolutions. If we look at a signal with a large "window", we would notice gross features. Similarly, if we look at a signal with a small "window", we would notice small features. The result in wavelet analysis is to see both the forest and the trees [5].

This makes wavelets interesting and useful. So, wavelet analysis represents the logical step: a windowing technique with variable-size regions. Wavelet analysis allows the use of long-time intervals where we want more precise low-frequency information, and shorter regions where we want high-frequency information [5] (see right Fig 1).

On the other hand, Fourier analysis breaks down a signal into constituent sinusoids of different frequencies. Another way to think of Fourier analysis is as a mathematical technique for transforming our view of the signal from time-based domain to frequency-based domains.

Fourier analysis has a serious drawback. In transforming to the frequency domain, time information is lost. When looking at the Fourier transform of a signal, it is impossible to tell when a particular event took place [5]. To localize a signal both in time and frequency, one uses windowed Fourier transform (see left Fig 1). In the time-frequency plane, the area, A, of the window must satisfy the following inequality,

$$A \geq \frac{1}{2},$$

according to Heisenberg's uncertainty principle which states that one cannot have a simultaneous sharp localization of both position and momentum of a particle in

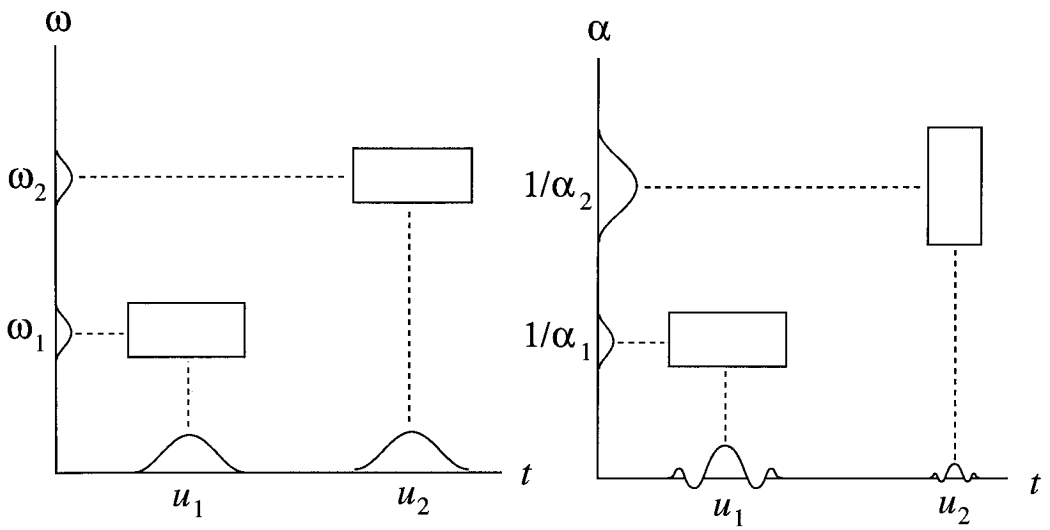


Figure 1: Left: Time-frequency boxes for windowed Fourier transform. Right: Time-scale boxes for wavelet transform.

quantum mechanics.

2.4 Wavelet and windowed Fourier transforms

Wavelet analysis complements Fourier analysis for which the main functions are the fast Fourier transform (FFT) and the inverse fast Fourier transform (IFFT). Fourier analysis uses the basic functions $\sin(\omega t)$, $\cos(\omega t)$, and $\exp(i\omega t)$.

1. In the frequency domain, these functions are perfectly localized. The functions are suited to the analysis and synthesis of signals with a simple spectrum, which is very well localized in frequency; for example, $\sin(\omega t)+0.5\sin(2\omega t)-\cos(3\omega t)$.

2. In the time domain, these functions are not localized. It is difficult for them to analyze or synthesize complex signals presenting fast local variations such as transients or abrupt changes: the Fourier coefficients for a frequency ω will depend on all values in the signal. To limit the difficulties involved, it is possible to "window" the signal using a regular function, which is zero or nearly zero outside a time segment $[-m, m]$.

The continuous windowed Fourier analysis coefficients are the doubly indexed coefficients:

$$G_s(\omega, t) = \int_{\mathbb{R}} e^{i\omega u} s(u) g(t-u)\, du, \tag{1}$$

where the window $g(t)$ is a positive function of compact support and $s(t)$ is an absolutely integrable or square integrable function over $\mathbb{R}$. The window is a square function in the case of Haar wavelets. But often it is a smooth bell function. In the left part of Fig. 1 the two functions of t are $|g(t-u_1)|$ and $|g(t-u_2)|$. The two functions of ω are the absolute values of their Fourier transforms, $|\hat{g}(\omega-\omega_1)|$ and $|\hat{g}(\omega-\omega_2)|$, respectively. The windowed Fourier transform uses a constant Heisenberg rectangle for all times and frequencies.

The analogy between this formula and the formula of the continuous wavelet coefficients is obvious:

$$C(\alpha, t) = \frac{1}{\sqrt{\alpha}} \int_{\mathbb{R}} s(u) \psi\left(\frac{t-u}{\alpha}\right) du,$$

where the function s is as above and the wavelet $\psi(t)$ has zero integral and its Fourier transform satisfies the mild admissibility condition

$$C_\psi = 2\pi \int_{-\infty}^{\infty} |\widehat{\psi}(\omega)|^2 \frac{d\omega}{\omega} < \infty. \tag{2}$$

The wavelet $\psi(t)$ has compact support for finite impulse response filters (FIR). Otherwise, for infinite impulse response filters (IIR) it decays to zero as $t \to \pm\infty$. In the right part of Fig. 1 the two functions of t are $\psi((t-u_1)/\alpha_1)$ and $\psi((t-u_2)/\alpha_2)$. The two functions of $1/\alpha$ are the absolute value of their shifted Fourier transforms $|\hat{\psi}|$, respectively. When the scale decreases, the time support is reduced but the frequency spread increases and covers an interval that is shifted towards high frequencies. In all cases, the height and width of the rectangle change but its area remains constant.

Large values of α correspond to small values of ω. The Fourier coefficient $G_s(\omega, t)$ depends on the values of the signal s on the segment $[t-m, t+m]$ with a constant width. If ψ, like g, is zero outside $[-m, m]$, the $C(\alpha, t)$ coefficients will depend on the values of the signal s on the segment $[t-\alpha m, t+\alpha m]$ of width $2\omega m$, which varies as a function of α. This important difference solves several difficulties, allowing a kind of time-windowed analysis, different at the various scales.

Wavelets remain competitive, however, even in contexts considered favorable for the Fourier technique. I. Daubechies gives an example of windowed-Fourier processing and complex Morlet wavelet processing, with $\alpha = 4$, of a signal composed mainly of the sum of two sines. This wavelet analysis gives good results [6].

The continuous Meyer wavelet is an orthogonal wavelet [14, pp. 71–81], [1, pp. 116–120]. In the Fourier domain, the scaling function and the wavelet, $\widehat{\phi}(\omega)$ and $\widehat{\psi}(\omega)$ have compact support and are given by closed form mathematical expression.

$$\widehat{\phi}(\omega) = \begin{cases} \frac{1}{\sqrt{2\pi}}, & \text{if} \quad \omega \leq \frac{2\pi}{3}, \\ \frac{1}{\sqrt{2\pi}} \cos\left(\frac{\pi}{2} v\left(\left(\frac{3}{2\pi}|\omega|\right)\right)\right), & \text{if} \quad \frac{2\pi}{3} \leq \omega \leq \frac{4\pi}{3}, \\ 0, & |\omega| > \frac{4\pi}{3} \end{cases}$$

and

$$\widehat{\psi}(\omega) = \begin{cases} \frac{1}{\sqrt{2\pi}} e^{i\omega/2} \sin\left(\frac{\pi}{2} v\left(\left(\frac{3}{2\pi}|\omega|\right)\right)\right), & \text{if} \quad \frac{2\pi}{3} \leq |\omega| \leq \frac{4\pi}{3}, \\ \frac{1}{\sqrt{2\pi}} e^{i\omega/2} \cos\left(\frac{\pi}{2} v\left(\left(\frac{3}{2\pi}|\omega|\right)\right)\right), & \text{if} \quad \frac{4\pi}{3} \leq |\omega| \leq \frac{8\pi}{3}, \\ 0, & \text{if} \quad |\omega| \notin \left[\frac{2\pi}{3}, \frac{8\pi}{3}\right], \end{cases}$$

where the auxiliary function $v(\alpha)$ is

$$v(\alpha) = \alpha^4 \left(35 - 84\alpha + 70\alpha^2 - 20\alpha^4\right), \qquad \alpha \in [0, 1],$$

which has value 0 at $\alpha = 0$ and increases smoothly to the value 1 at $\alpha = 1$. This function is plotted in Fig. 2. By changing the auxiliary function one gets a family of different wavelets.

In the time domain, $\phi(x)$ and $\psi(x)$ are plotted in Fig. 3. These functions do not have compact support since they are the inverse Fourier transforms of functions with compact support. But they decrease to 0 when $x \to \pm\infty$ faster than any inverse polynomial x^{-n} for arbitrary large $n > 0$.

In this thesis, frequent use is made of the discrete version of the Meyer continuous scaling function and wavelet. Table 1 to be read from left-to-right and top-to-bottom lists 31 equally sampled values for $t = -30 : 0$ of the Meyer continuous scaling function. Since the scaling function is and even function, the sampled values from $t = 0 : 31$ can be read from right-to-left and bottom-to-top. These values are use

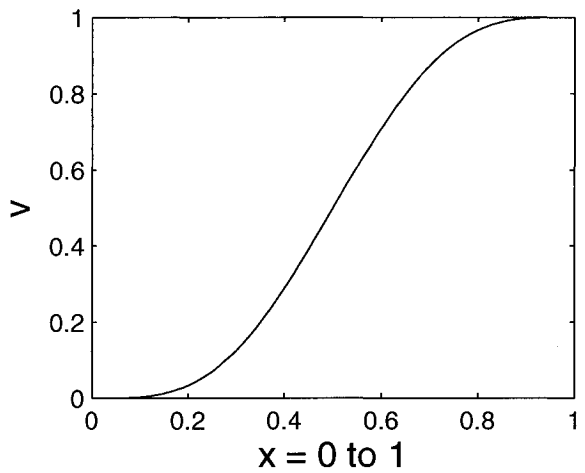


Figure 2: Meyer auxiliary function.

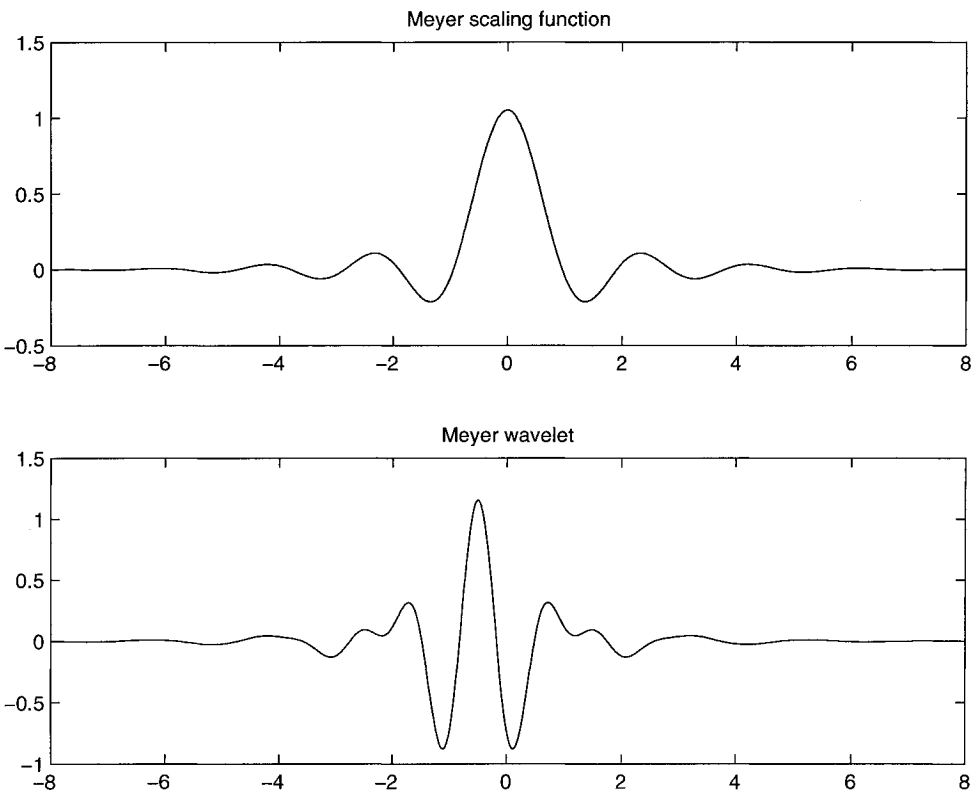


Figure 3: Meyer scaling function and wavelet.

Table 1: Equally spaced interpolation over the 31 points $t = -30 : 0$, in an appropriate scale, of the even Meyer continuous scaling function. The table is to be read from left-to-right and top-to-bottom

−0.00000000000071	0.00000000602417	−0.00000000786264
−0.00000000763592	0.00000004290000	−0.00000007683788
0.00000005798757	0.00000008331843	−0.00000038935709
0.00000079959264	−0.00000105327040	0.00000052096610
0.00000226658981	−0.00001153482109	0.00004634594367
−0.00042507742534	−0.00191249208176	0.00155742686495
0.00427503632826	−0.00451679913188	−0.00782165944267
0.01079753170823	0.01232022893154	−0.02271990328399
−0.01721716407758	0.04507029824632	0.02167642432751
−0.09393574608124	−0.02481064959753	0.31437474052048
0.52650154394838		

for the function `dmey` in the MATLAB wavelet toolbox. The symmetry of the discrete approximation about the central point is immediately apparent. The graph of this approximation is shown in Fig. 4.

As mentioned in [17, p. 46], linear phase wavelets have zero group delay difference. This holds for the symmetric biorthogonal wavelets $B_{9/7}$ and $B_{23/14}$. Symmetric biorthogonal wavelets were introduced in [2] and are also described in [1, Section 8.3]. Since the Meyer wavelets are symmetric, they are linear phase so that they also have zero group delay difference. To see this, the discrete Meyer wavelet has been used to obtain the level-1 decomposition of the square grid shown in Fig 5. In Fig. 6 the horizontal lines in the approximation (top left square) and in the horizontal detail (top right square) have no delay with respect to each other. The same thing holds for the vertical lines in the top and bottom left parts of the figure. In this figure, the four parts are scaled independently to show the four different patterns. Figure 7 illustrates the same horizontal and vertical alignments without group delay in the level-2 decomposition with `dmeyer`.

The absence of group delay difference with the Meyer wavelet may explain its better performance in removing heavier noise as will be shown in this thesis.

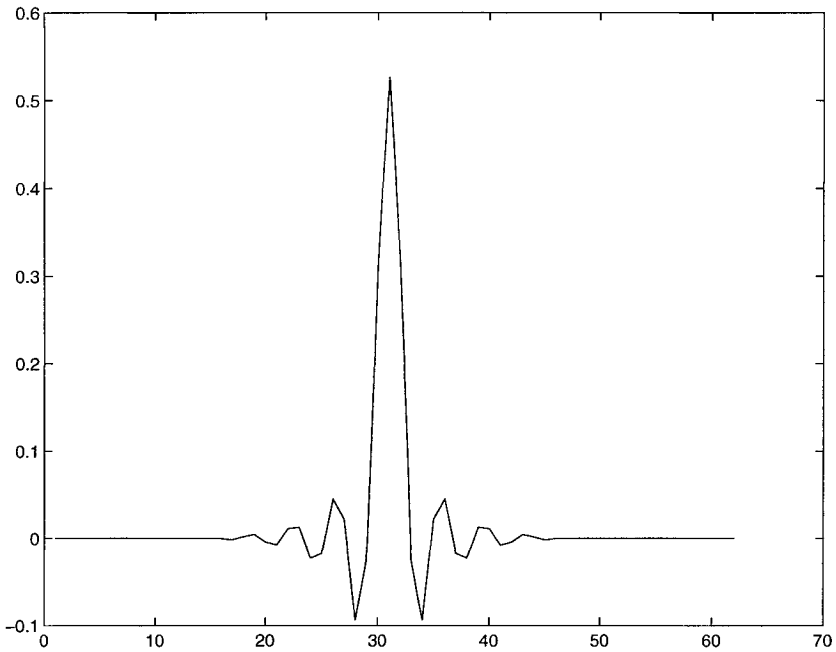


Figure 4: A 61-point equally spaced discrete approximation to Meyer's continuous scaling function.

2.5 Discrete wavelet transform

In many applications, like signal and image processing, one uses discrete wavelet transforms.

A given function $\phi(t)$ with nonzero integral, which satisfies the dilation equation

$$\phi(t) = \sqrt{2} \sum_{k \in \mathbb{Z}} h_k \phi(2t - k),$$

where

$$h_k = \langle \phi,\, \phi_{-1,k} \rangle = \sqrt{2} \int_{\mathbb{R}} \phi(t)\, \overline{\phi(2t - k)}\, dt,$$

is called a scaling function. It is assumed that the set of integer translates $\{\phi(t-k)\}$ is an orthonormal basis for a subspace V_0 of $L^2(\mathbb{R})$. The scaled and shifted functions

$$\phi_{j,k}(t) = 2^{-j/2} \phi(2^{-j} t - k)$$

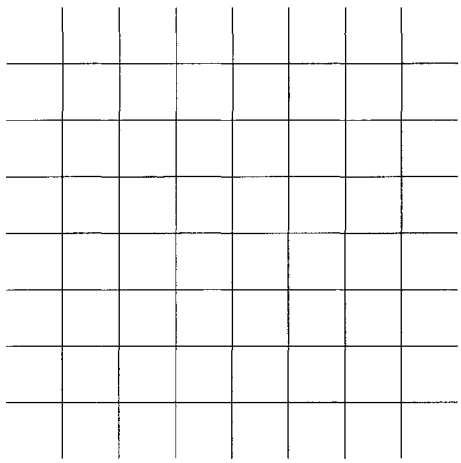


Figure 5: Square grid.

generate nested subspaces V_j:

$$\{0\} \subset \cdots \subset V_{-1} \subset V_0 \subset V_1 \subset \cdots \subset L^2(\mathbb{R}).$$

A function $\psi \in V_1$ with zero integral, which satisfies the wavelet equation

$$\psi(t) = \sqrt{2} \sum_{k \in \mathbb{Z}} g_k \phi(2t - k),$$

where

$$g_k = (-1)^k \overline{h_{1-k}},$$

is called wavelet function. The set of its translates $\{\psi(t-k)\}$ is an orthonormal basis of the subspace W_0 which is the orthogonal complement of V_0 in V_1. Similarly, the sets of scaled and shifted functions $\{2^{-j/2}\phi(2^{-j}t - k)\}$ are orthonormal bases of the subspaces W_j which are the orthogonal complements of V_j in V_{j+1}, for each $j \in \mathbb{Z}$.

A function $f \in L^2(\mathbb{R})$ has the wavelet expansion

$$f(t) = \sum_{k \in \mathbb{Z}} c_k \phi(t - k) + \sum_{j \in \mathbb{Z}_+, k \in \mathbb{Z}} d_{j,k} 2^{-j/2} \psi(2^{-j}t - k),$$

where $\mathbb{Z}_+$ is the set of nonnegative integers.

Mallat's fast wavelet transform consists in convolving the coefficients c_k with the coefficients h_k of the scaling function to decompose a fine scale signal into a coarse

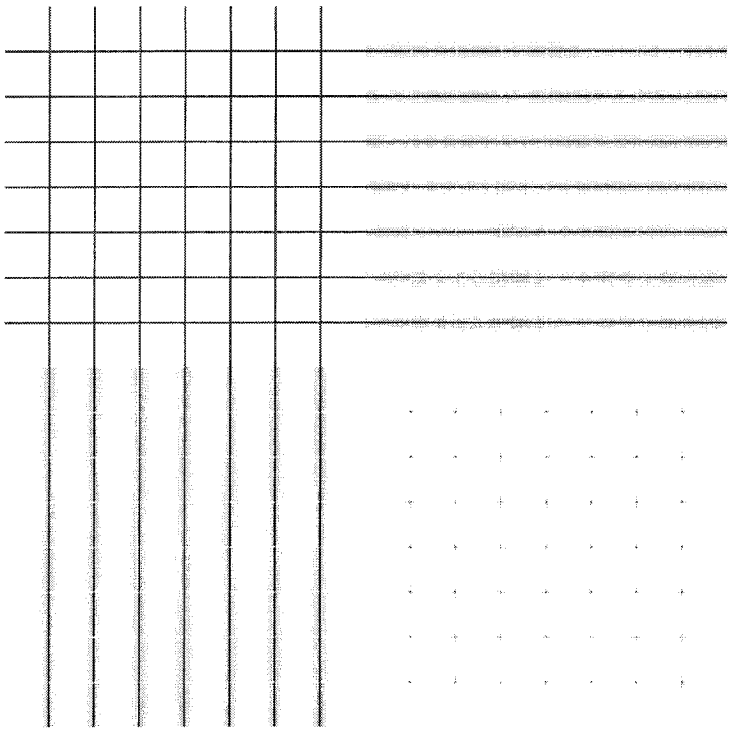


Figure 6: Level-1 decomposition of the square grid with `dmey`.

scale approximation and details at intermediary scales by filtering and downsampling. The inverse fast wavelet transform reconstructs the signal by upsampling and filtering.

The **discrete wavelet transform** is a mapping:

$$\ell^2(\mathbb{Z}) \ni \{c_{j,n}\}_{n\in\mathbb{Z}} \mapsto \left(\{c_{j+1,k}\}_{k\in\mathbb{Z}}, \{d_{j+1,k}\}_{k\in\mathbb{Z}}\right) \in \ell^2(\mathbb{Z})^2,$$

defined by

$$\begin{cases} c_{j+1,k} = \sum_{n\in\mathbb{Z}} \overline{h_{n-2k}}\, c_{j,n} = \sum_{n\in\mathbb{Z}} \overline{h_n}\, c_{j,n+2k}, \\ d_{j+1,k} = \sum_{n\in\mathbb{Z}} \overline{g_{n-2k}}\, c_{j,n} = \sum_{n\in\mathbb{Z}} \overline{g_n}\, c_{j,n+2k}. \end{cases}$$

The DWT can be decomposed into the following two processes.

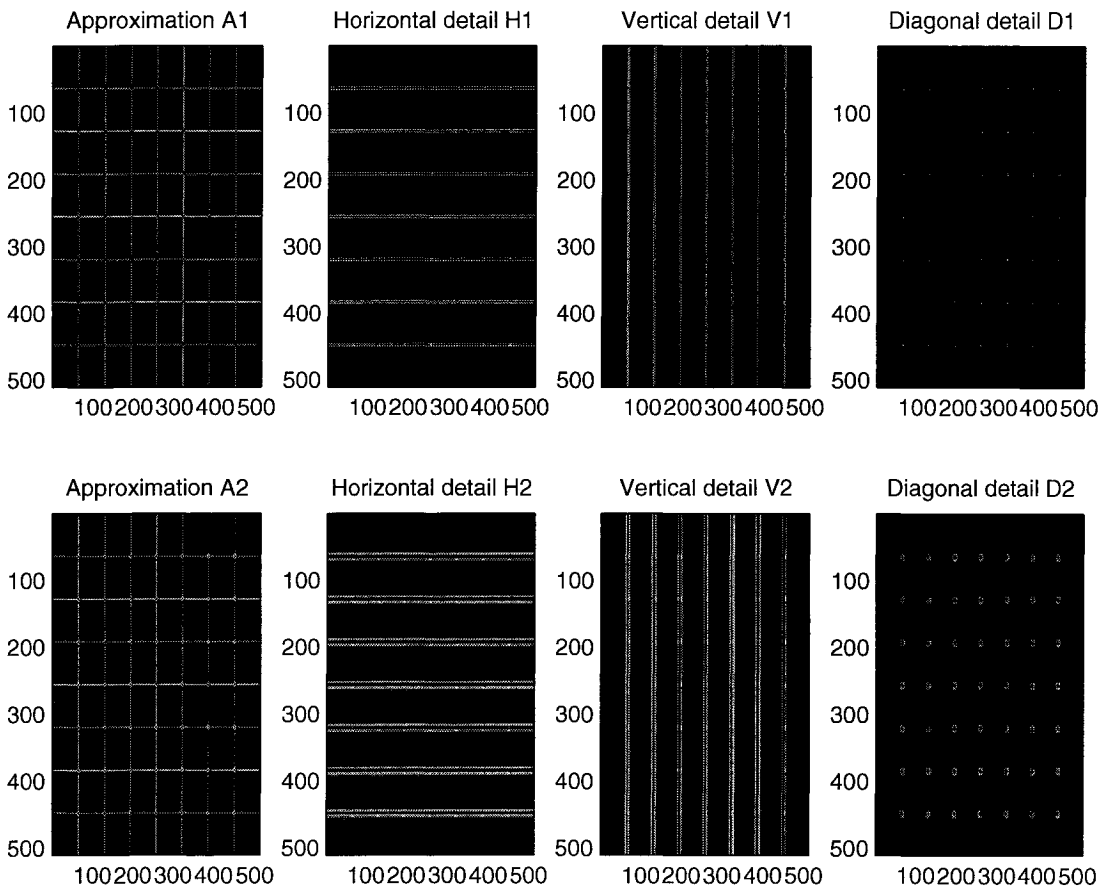


Figure 7: Level-2 decomposition of the square grid with `dmey`.

(i) **Convolution** or **filter**:

$$\widetilde{c}_{j+1,\ell} = \sum_{n\in\mathbb{Z}} \overline{h_{n-\ell}}\, c_{j,n} = (\widetilde{h} * c_j)(\ell),$$
$$\widetilde{d}_{j+1,\ell} = \sum_{n\in\mathbb{Z}} \overline{g_{n-\ell}}\, c_{j,n} = (\widetilde{g} * c_j)(\ell),$$

where $\widetilde{h}_k = \overline{h_{-k}}$, $\widetilde{g}_k = \overline{g_{-k}}$.

(ii) **Down sampling**:

$$c_{j+1,k} = \widetilde{c}_{j+1,2k},$$
$$d_{j+1,k} = \widetilde{d}_{j+1,2k}.$$

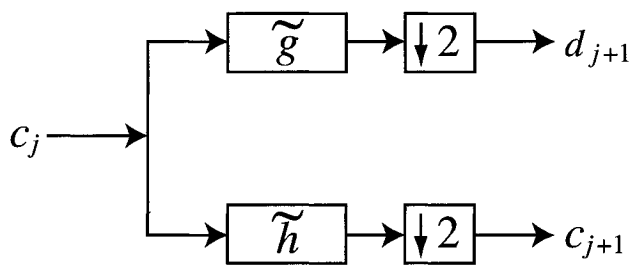


Figure 8: Discrete wavelet transform.

The **inverse discrete wavelet transform** is a mapping:

$$\ell^2(\mathbb{Z})^2 \ni \left(\{c_{j+1,k}\}_{k\in\mathbb{Z}},\, \{d_{j+1,k}\}_{k\in\mathbb{Z}}\right) \mapsto \{c_{j,n}\}_{n\in\mathbb{Z}} \in \ell^2(\mathbb{Z}),$$

defined by

$$c_{j,n} = \sum_{k\in\mathbb{Z}} h_{n-2k}\, c_{j+1,k} + \sum_{k\in\mathbb{Z}} g_{n-2k}\, d_{j+1,k}.$$

The IDWT can be decomposed into the following two processes.

(i) **Up sampling**:

$$\widetilde{c}_{j+1,\ell} = \begin{cases} 0, & \ell \in 2\mathbb{Z}+1, \\ c_{j+1,\ell/2}, & \ell \in 2\mathbb{Z}. \end{cases}$$

(ii) **Convolution** or **filter**:

$$\sum_{k\in\mathbb{Z}} h_{n-2k}\, c_{j+1,k} = \sum_{\ell\in\mathbb{Z}} h_{n-\ell}\, \widetilde{c}_{j+1,\ell} = (h * \widetilde{c}_{j+1})(n).$$

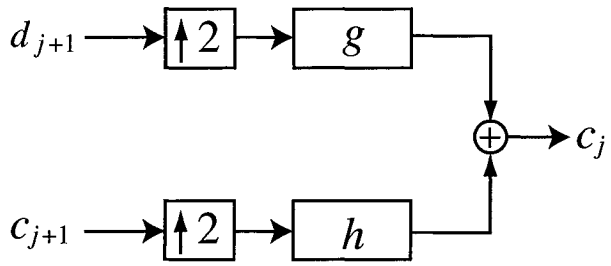


Figure 9: Inverse discrete wavelet transform.

2.6 Image decomposition by 2D wavelet

Images are treated as two-dimensional signals, they change horizontally and vertically. Thus 2D wavelet analysis can be used for images. 2D wavelet analysis uses the same *mother wavelets* but requires an extra step at every level of decomposition. The 1D analysis filters out the high frequency information from the low frequency information at every level of decomposition; so only two subsignals are produced at each level. Images are considered to be matrices with N rows and M columns. At every level of decomposition the horizontal data is filtered, and then the approximation and details produced from this are filtered on columns [7].

Separable 2D scaling functions and wavelets are often made from the tensor product of 1D scaling function $\phi(x)$ and wavelet $\psi(x)$, respectively. Thus we get one scaling function and three wavelet functions:

$$\mathbf{\Phi}(x,y) = \phi(y)\phi(x), \qquad \mathbf{\Psi}_H(x,y) = \psi(y)\phi(x),$$

$$\mathbf{\Psi}_V(x,y) = \phi(y)\psi(x), \qquad \mathbf{\Psi}_D(x,y) = \psi(y)\psi(x).$$

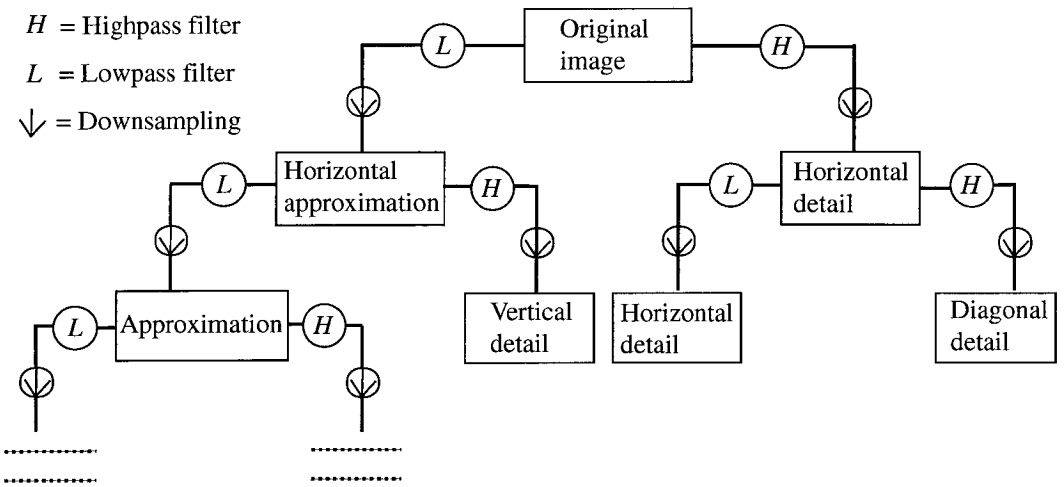


Figure 10: Image decomposition by 2D wavelet analysis.

The scaling functions picks up the approximation and the three wavelets pick up the horizontal, vertical and diagonal details. The tree in Fig. 10 illustrates the 2D image decomposition. In the language of signal processing [1], [18], an analysis lowpass filter lets the low frequency part of a signal go through and eliminates the high frequencies. Similarly, an analysis highpass filter lets the high frequency part go through and eliminates the low frequency part. Thus, instead of a length N signal, one gets two signals each of length N. This redundancy is taken care of in part by downsampling each filtered signal, that is, by eliminating every second term in the signal representation. Aliasing caused by this process is removed by properly choosing the synthesis filters.

The finger print image `Fp1` shown in Fig. 11 is decomposed to level 1 by means of the discrete Meyer wavelet. This decomposition is shown in Fig. 12

2.7 Image denoising

2.7.1 Matlab commands to add noise to images

A two-dimensional model with additive noise is of the form

$$J(i,j) = f(i,j) + e(i,j),$$

Figure 11: The finger print image `Fp1`.

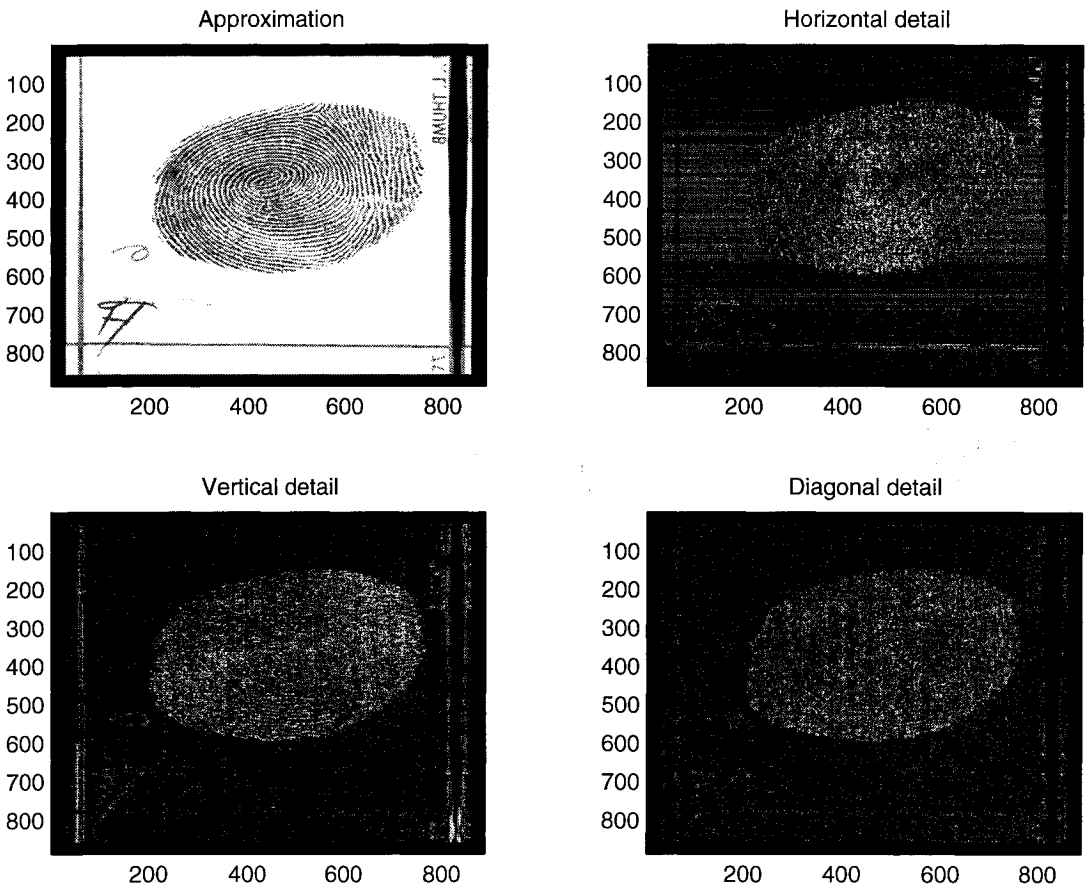


Figure 12: Level 1 decomposition of the image of a fingerprint by means of the discrete Meyer wavelet.

Table 2: MATLAB commands to add noise to images.

Option	MATLAB command
Gaussian	J = imnoise(I, 'gaussian', mean, variance)
'Salt and Pepper'	J = imnoise(I,'Salt and Pepper', density)
Speckle	J = imnoise(I,' Speckle ', variance)

where J is the noised image, f is the original image and e is noise. In this thesis, the noise, e, will be restricted to the following three types: zero-mean, or ordinary, Gaussian, salt and pepper, and speckle.

These additive noises can be produced by the following MATLAB commands.

- `J = imnoise(I, 'gaussian', mean, variance)` adds Gaussian white noise of mean m and variance v to image I. The default is zero mean with 0.01 variance.
- `J = imnoise(I,'salt & pepper',density)` adds salt and pepper noise to image I, where d is the noise density. This affects approximately `dxprod(size(`I`))` pixels. The default is 0.05 noise density.
- `J = imnoise(I,'Speckle',v)` adds multiplicative noise to image I, using the equation $J = I + nI$, where n is a uniformly distributed random noise with mean 0 and variance v. The default for v is 0.04.

Here is an example of additive Gaussian noise. A kind of noise which occurs in recorded images to a certain extent is a detector noise which is due to the discrete nature of radiation, that is, the fact that each imaging system is recording an image by counting photons. Under assumptions (which are valid for many applications) this noise can be modeled with an independent, additive model, where the noise $n(i, j)$ has a zero-mean Gaussian distribution described by its standard deviation, or variance. This means that each pixel in the noisy image is the sum of the true pixel value and a random Gaussian distributed noise value.

Here is an instance of salt and pepper noise. A common form of noise is data drop-out noise (commonly referred to as intensity spikes, speckle or salt and pepper noise).

Here, the noise is caused by errors in data transmission. The corrupted pixels are either set to the maximum value (which looks like snow in the image) or have single bits flipped over. In some cases, single pixels are set alternatively to zero or to the maximum value, giving the image a "salt and pepper" like appearance. Unaffected pixels remain unchanged. The noise is usually quantified by the percentage of pixels which are corrupted.

2.7.2 Removing noise

The two-dimensional denoising procedure has three main steps and uses two-dimensional wavelet tools [11].

- Decompose image I.
- Choose a wavelet and a level N. Compute the wavelet decomposition of I at level N.
- Threshold detail coefficients. For each level from 1 to N, select a threshold and apply soft thresholding to the detail coefficients.
- Reconstruct I.

The wavelet reconstruction uses the original approximation coefficients of level N and the modified detail coefficients of levels 1 to N.

Two points must be addressed: how to choose the threshold and how to perform the thresholding [12].

2.7.3 Soft and hard thresholdings

Hard thresholding can be described as the usual process of setting to zero the elements whose absolute values are lower than the threshold.

Soft thresholding is an extension of hard thresholding, first setting to zero the elements whose absolute values are lower than the threshold, and then shift the nonzero coefficients towards 0 to restore the continuity of the thresholded signal [12].

Soft thresholding is used for all the algorithms for the following reasons: Soft thresholding has been shown to achieve near minimum error over a large number of Besov spaces [4]. Moreover, it is also found to yield visually more pleasing images. Hard thresholding is found to introduce artifacts in the recovered images [16].

2.7.4 Global and local thresholdings

In this thesis we intend to implement soft thresholding as well as a global option to explore the effects of different values of the parameters on the performance.

2.8 Wavelet denoising implementation

In this thesis we use the MATLAB functions `ddencmp( )` to obtain the default values for denoising or compression and `wdencmp( )` to perform a denoising or compression process of a signal or an image using wavelets. As an example, one can use the following M-file to illustrate my denoising function of a real image.

```
function Y=wavelet_denoise(X,Bw)
    % X: noised image; XX middle variable
    % Y: denoise image
    % Bw: import arguments, including wavelet type option and
decomposition level option.
    % Use ddencmp and wdencmp for image matrix denoising.
    XX=X;
    Type=Bw.type; level=Bw.level;
% Find default values. In this case fixed form threshold
% is used with estimation of noise a level, thresholding
% mode is soft and the approximation coefficients are
% kept.

[thr,sorh,keepapp] = ddencmp('den','wv', XX);
    % de-noise image using global thresholding option.
```

```
    Y = wdencmp('gbl', XX, Type ,Level ,thr, sorh, keepapp);
    return;
```

The call of the code is as follows:

```
Bw.Type='db10';
    Bw.Level=2;
    X=nwomen;
    filtered=wavelet_denoise(X,Bw);
```

From the top to bottom, Fig 13 shows the original, noised and denoised 512×512 Goldhill and Boats images, respectively. Gaussian white noise has been added to the original images by the `imnoise` function listed in Table 2. Denoising has been done by the `ddencmp` and `wdencmp` denoising functions with discrete Meyer's wavelets and global threshold strategy as referred to in sections 2.7 and 2.8 to illustrate the effects of the wavelet denoising function.

2.9 Different denoising approaches

The wavelet toolbox provides different ways to remove or reduce noise in an image depending upon the kind of noise. The available methods include:

- Adaptive filtering
- Average filtering
- Median filtering

2.9.1 Adaptive filtering

Adaptive filtering applies the function `wiener2` to an image adaptively. The `wiener2` lowpass filter denoises an intensity image that has been degraded by constant power additive noise, using a pixelwise adaptive Wiener method based on statistics estimated from a local neighborhood of each pixel. An adaptive filter is more selective than a

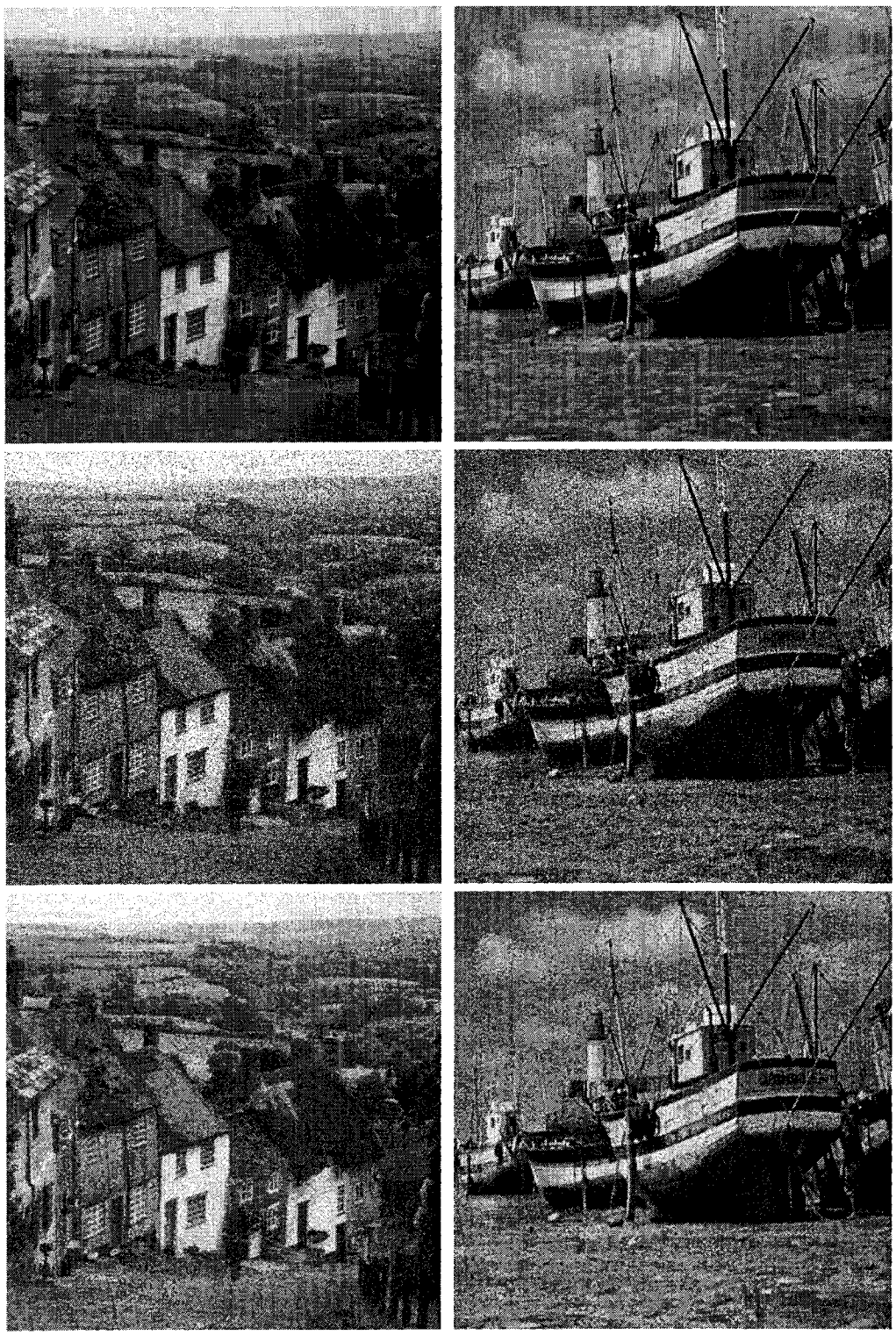


Figure 13: From top to bottom: original, zero-mean Gaussian noised and denoised Goldhill and Boats by means of the `ddencmp` and `wdencmp` denoising functions.

comparable linear filter because it preserves edges and other high frequency parts of an image. The `wiener2` function handles all preliminary computations and implements the filter for an input image. wiener2; however, it requires more computation time than linear filtering.

Analysis of the adaptive filtering algorithm

The function `wiener2` estimates the local mean and variance around each pixel $\alpha(n_1, n_2)$,

$$\mu = \frac{1}{NM} \sum_{n_1,n_2 \in \eta} \alpha(n_1, n_2), \qquad \sigma^2 = \frac{1}{NM} \sum_{n_1,n_2 \in \eta} \alpha^2(n_1, n_2) - \mu^2,$$

where η is the $N \times M$ local neighborhood of each pixel in image A. `wiener2` then creates a pixelwise Wiener filter using these estimates,

$$b(n_1, n_2) = \mu + \frac{\sigma^2 - v^2}{\sigma^2} \left[\alpha(n_1, n_2) - \mu\right],$$

where v is the noise variance. If the noise variance is not given, `wiener2` uses the average of the estimates of all local variances [19].

Implementation

Adaptive filtering can be used in two different ways.

- `PJ = wiener2(I,[m n],noise)` filters image I using pixelwise adaptive Wiener filtering in neighborhoods of size $m \times n$ to estimate the local image mean and standard deviation. If one omits the $[m\ n]$ argument, m and n default to 3. The power of the additive Gaussian white noise is assumed to be `noise`.
- `[J,noise] = wiener2(I,[m n])` estimates the additive noise power before doing the filtering. `wiener2` returns this estimate in noise.

The input image I is a two-dimensional image of class `uint8`, `uint16`, or double. The output image J is of the same size and same class as I.

2.9.2 Average filtering

Average filtering applies the function `filter2` to remove certain types of noise, such as grain noise from a photograph. Average filtering is appropriate for this purpose because each pixel is set to the average of the pixels in its neighborhood, local variations

caused by grain are reduced.

Analysis of the average filtering algorithm

Given a matrix X and a two-dimensional FIR filter h, `filter2` rotates the filter matrix by 180 degrees to create a convolution kernel. It then calls `conv2`, the two-dimensional convolution function, to implement the filtering operation. `filter2` uses `conv2` to compute the full two-dimensional convolution of the finite impulse response filter (FIR) with the input matrix. By default, `filter2` extracts the central part of the convolution that is of the same size as the input matrix and returns this as the result. If the shape parameter specifies an alternate part of the convolution for the result, `filter2` returns the appropriate part [13].

Implementation

Average filtering can be used in two ways.

- `Y = filter2(h,X)` filters the data in X with the two-dimensional finite impulse filter (FIR) filter in matrix h. It computes the result, Y, using a two-dimensional correlation, and returns the central part of the correlation that is of the same size as X.

- `Y = filter2(h,X,shape)` returns the part of Y specified by the shape parameter. `shape` is a string with one of the following values: `'full'` returns the full two-dimensional correlation. In this case, Y is larger than X. `'same'` (default) returns the central part of the correlation. In this case, Y is of the size of X. `'valid'` returns only those parts of the correlation that are computed without zero-padded edges. In this case, Y is smaller than X.

2.9.3 Median filtering

The function `medfilt` implements median filtering which is similar to using an averaging filter, in that each output pixel is set to an "average" of the pixel values in the neighborhood of the corresponding input pixel. However, with median filtering, the value of an output pixel is determined by the median of the neighborhood pixels, rather than the mean. Median filtering is a nonlinear operation often used in image

processing to reduce "salt and pepper" noise and is more effective than convolution when the goal is to simultaneously reduce noise and preserve edges.

Analysis of the median filtering algorithm

If the input image A is of an integer class, all the output values are returned as integers. If the number of pixels in the $m \times n$neighborhood is even, some of the median values might not be integers. In such case, the fractional parts are discarded. Logical inputs are treated similarly.

Implementation

Median filtering can be used in three different ways.

- `B = medfilt2(A,[m n])` performs median filtering of the matrix A in two dimensions. Each output pixel contains the median value in the $m \times n$ neighborhood around the corresponding pixel in the input image.
- `B = medfilt2(A)` performs median filtering of the matrix A using the default 3×3 neighborhood.
- `B = medfilt2(A,'indexed',...)` processes A as an indexed image, padding with 0's if the class of A is `uint8`, or 1's if the class of A is double. The input image A can be of class logical, `uint8`, `uint16`, or double (unless the `indexed` syntax is used, in which case A cannot be of class uint16). The output image B is of the same class as A.

2.9.4 BlockSvd filtering

We present an algorithm for filtering noise based on nonlinear block processing of images using singular value decomposition (SVD). Noise filtering is performed in the singular value and singular vector domains. A priori knowledge of the noise variance is not required because an estimation of the singular value noise variance is performed in the first phase of the procedure. The nonlinear filtering is based on eliminating changes in singular values and singular vectors caused by additive Gaussian white noise or other types of noise. Processing the image in smaller blocks makes the SVD procedure computationally feasible.

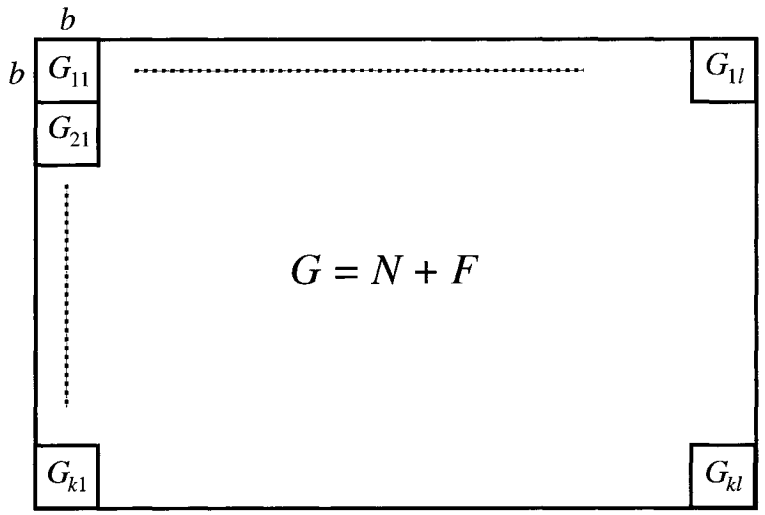


Figure 14: Decomposition of noised image G into $b \times b$ blocks.

Analysis of the BlockSvd Algorithm

The general denoising procedure for BlockSvd involves four steps.

- Decompose the image into small blocks.
- Factor each block by the singular value decomposition method.
- Eliminate changes in singular values and singular vectors in each block.
- Reconstruct the image.

Implementation

Let the original, non-corrupted, image F, be represented as a $K \times L$ matrix. Adding noise to F will produce the noised image G of the same size,

$$G = F + N, \tag{3}$$

where N is a random $K \times L$ noise field. The added noise will degrade the original information contained in F. The proposed algorithm is based on block processing [3].

The noised image is divided into square blocks of size $b \times b$. For simplicity, we suppose that $K = kb$ and $L = lb$ (see Fig. 14). Each block has the SVD representation

$$G_{ij} = U_{ij} S_{ij} V_{ij}^T, \qquad i = 1, 2, \ldots, k, \quad j = 1, \ldots, l, \tag{4}$$

where U_{ij} is the $b \times b$ unitary matrix of left singular vectors, S_{ij} is the $b \times b$ diagonal matrix of singular values, and V_{ij} is the $b \times b$ unitary matrix of right singular vectors [5]. The singular values and the corresponding singular vectors contain complete information about the image block. Equation (4) can be interpreted as an exterior product expansion of the base image,

$$G_{ij} = \sum_{r=1}^{R} U_{ijr} S_{ijr} V_{ijr}^T, \tag{5}$$

where R is the rank of G_{ij}, S_{ijr} are the singular values, U_{ijr} are left singular vectors, and V_{ijr} are right singular vectors. Image degradation, that includes blurring and noising, is reflected in changes in singular values and singular vectors. This is true for the SVD representation of the whole $K \times L$ image, as well as for the SVD representation of the individual $b \times b$ blocks.

Singular value decomposition is performed for each block. Then the average sum of the last $b - t$ singular values is calculated over every block:

$$n_s = \frac{1}{k \times l} \sum_{i=1}^{k} \sum_{j=1}^{l} \sum_{r=b-t+1}^{b} s_{ijr}. \tag{6}$$

This is not a true value of noise variance, but a value that is proportional to it. Previously calculated SVD of image blocks will now be used for filtering.

The first step in the filtering decreases the noised singular values for every block:

$$\hat{s}_{ijr} = s_{ijr} - p_1 n_s w(r), \tag{7}$$

where $\hat{s}_{ijr}$ is a filtered singular value, p_1 is an image dependent parameter and $w(r)$ is a weighting function that determines the percentage of the estimated noise variance to be subtracted from the noised singular values s_{ijr}. The weighting function used in [3] is

$$w(r) = 1 - \left(1 - \frac{r-1}{b/2}\right)^2, \qquad r = 1, 2, \ldots, b, \tag{8}$$

which is an interpolating parabolic function. A plot of this function for $b = 32$ is shown in Fig 15.

In the second step, the singular vectors is carefully filtered to avoid catastrophic changes in images. Thus the filtering operations are limited to a slight filtering of noise in the singular vectors. The singular vectors are first transformed by the DFT and then the part of the Fourier transform that corresponds to the higher frequencies is reduced by a factor p_2. The filtering operation is performed on both left and right singular vectors.

After the filtering operation has been applied to the three matrices of the SVD of each block of the noised image, the filtered block, $\widehat{F}_{ij}$, is calculated:

$$\widehat{F}_{ij} = \widehat{U}_{ij}\widehat{S}_{ij}\widehat{V}ij^T, \qquad i = 1, 2, \ldots, k, \quad j = 1, 2, \ldots, l, \tag{9}$$

where $\widehat{U}_{ij}$ is the $b \times b$ matrix of filtered left singular vectors, $\widehat{S}_{ij}$ is the $b \times b$ diagonal matrix of restored singular values, and $\widehat{V}_{ij}$ is the $b \times b$ matrix of filtered right singular vectors. The complete filtered image F is reconstructed using the filtered blocks [3].

2.10 Objective measures for numerical results

Suitable criteria are needed to evaluate rigorously the performance of a noise removal scheme. In case of images, the search for simple and suitable criteria is hindered by the fact that the results obtained by statistical performance criteria may not agree with the subjective evaluation by the human eye.

There are several objective measures to evaluate the quality of a restored image. Some of these are defined in terms of the Frobenius norm of an $m \times n$ matrix A:

$$\|A\|_{\text{fro}} = \left[\sum_{i=1}^{m}\sum_{j=1}^{n} |A_{ij}|^2\right]^{1/2}. \tag{10}$$

The *mean square error*, (MSE), of A is

$$\text{MSE} = \frac{1}{mn}\|A\|_{\text{fro}}^2. \tag{11}$$

Given an 8-bit-per-pixel original image F, the *peak signal to noise ratio*, (PSNR), in the noised image G is

$$\text{PSNR} = 10\log_{10}\left(\frac{255^2 mn}{\|F - G\|_{\text{fro}}^2}\right). \tag{12}$$

Given an original image F, the *signal to noise ratio*, (SNR), in the noised image G is

$$\text{SNR} = 10\log_{10}\left(\frac{\|F\|_{\text{fro}}^2}{\|F-G\|_{\text{fro}}^2}\right). \tag{13}$$

The *maximum error* between images F and G is

$$\|F-G\|_\infty = \max_{i,j}|F_{ij}-G_{ij}|. \tag{14}$$

In this thesis, we compare the restored images obtained by means of different methods mainly on the two measures: Signal-to-Noise Ratio (SNR) of the noised image and Signal-to-Noise Ratio Improvement (SNRI) of the denoised image approximation. The SNR improvement (SNRI) in db is defined as

$$\begin{aligned}\text{SNRI} &= 10\log_{10}\left[\left(\frac{\|F\|_{\text{fro}}^2}{\|F-\widehat{F}\|_{\text{fro}}^2}\right)\Big/\left(\frac{\|F\|_{\text{fro}}^2}{\|F-G\|_{\text{fro}}^2}\right)\right]\\ &= 10\log_{10}\left(\frac{\|F-G\|_{\text{fro}}^2}{\|F-\widehat{F}\|_{\text{fro}}^2}\right),\end{aligned} \tag{15}$$

where F is the original clean image, G is the noised image, and $\widehat{F}$ is the denoised image. Similarly, the PSNR improvement (PSNRI) in db is

$$\begin{aligned}\text{PSNRI} &= 10\log_{10}\left[\left(\frac{255^2mn}{\|F-\widehat{F}\|_{\text{fro}}^2}\right)\Big/\left(\frac{255^2mn}{\|F-G\|_{\text{fro}}^2}\right)\right]\\ &= 10\log_{10}\left(\frac{\|F-G\|_{\text{fro}}^2}{\|F-\widehat{F}\|_{\text{fro}}^2}\right),\end{aligned} \tag{16}$$

It is seen that the peak signal to noise ratio improvement is the same as the signal to noise ratio improvement.

The percentage improvement rate of image A compared to image B is defined as

$$\text{SNRI rate} = \frac{\text{SNRI of } A}{\text{SNRI of } B}\times 100. \tag{17}$$

It is also seen that the peak signal to noise ratio improvement rate is the same as the signal to noise ratio improvement rate.

In this thesis, we shall denoise different images corrupted by zero-mean Gaussian white noise and other noises, and compare the SNR and SNRI obtained by different approaches for a variety of noise levels. Normally, for low-noise images, SNR should be bigger than 20db.

Chapter 3

A Pair of Improved Algorithms

3.1 Idea derived from BlockSvd

An idea, which is derived from the BlockSvd image denoising, will be presented below. We can say that this is an improved and upgraded method based on the former BlockSvd approach. After testing various types of images, we found two existing problems in the BlockSvd method.

3.2 The weighting function problem

In the BlockSvd method, Devčić and Lončarić [3] suppose that the noise variance is firstly estimated, then the filtering process is performed on the singular values and the singular vectors. Estimating and filtering the singular values of each image block is the most critical process to determine the accuracy and correctness of the method. In the BlockSvd method, a weighting function (see formula (7)) is used to estimate the percentage of the noise difference between a singular value of the noised and the original images. This estimate directly influences the accuracy of the BlockSvd method, but after many experiments, we found that the weighting function (8), which we repeat here for the convenience of the reader,

$$w(r) = 1 - \left(1 - \frac{r-1}{b/2}\right)^2, \qquad r = 1, 2, \ldots, b, \tag{18}$$

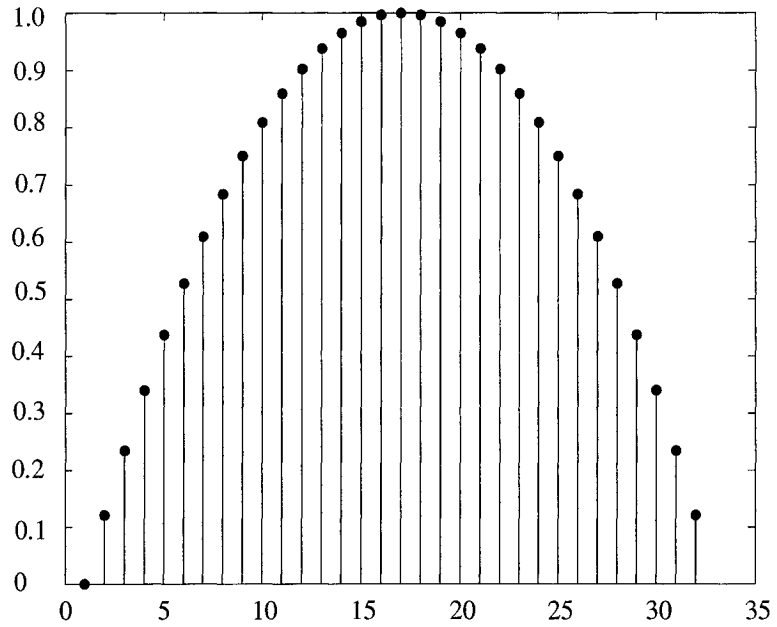


Figure 15: Plot of the weighting function (18).

where $b = 32$ and the blocksize is 32×32, used in [3]. This weighting function (18) which is plotted in Fig. 15, does not sufficiently match the real situations.

To show this point, we are going to compute the actual difference between the singular values of noised and original images. It will be seen from Figs.16, 17 and 18 that, in practice, the curve shown in Fig. 15 is not the best choice for the weighting function $w(r)$ using the estimated weighting function (18).

Table 3 lists the values of the parameters used in the MATLAB commands described in Table 2 to obtain desired levels of SNR in the noised images: Boats, Lena, Fp1, Goldhill and Yogi.

Estimating the difference between the singular values of the original image and the noised image is a critical step in this BlockSvd approach as it directly determines the accuracy of the denoising procedure. We first obtain empirical data for this difference for several images and different noise and plot the curves of the normalized averages differences.

The steps of the analysis procedure are as follows.

Table 3: Parameter values to obtain desired SNR levels in noised images.

Zero-mean Gaussian noise added to Boats						
Variance	0.003	0.005	0.0075	0.015	0.05	0.13
SNR of noised Boats (db)	20.0	17.7	15.9	12.9	8.2	5.1
Zero-mean Gaussian noise added to Lena						
Variance	0.0025	0.005	0.0075	0.015	0.05	0.13
SNR of noised Lena (db)	20.3	17.3	15.6	12.6	7.9	4.9
Salt and pepper noise added to Fp1						
Density	0.01	0.1	0.2	0.3	0.4	0.5
SNR of noised Fp1 (db)	22.2	12.2	9.2	7.4	6.2	5.2
Salt and pepper noise added to Lena						
Variance	0.01	0.05	0.1	0.2	0.3	0.4
SNR of noised Lena (db)	19.8	12.7	9.7	6.7	5.0	3.7
Speckle noise added to Goldhill						
Variance	0.01	0.02	0.05	0.1	0.2	0.4
SNR of noised Gold (db)	20.1	17.2	13.4	10.6	7.7	5.0
Speckle noise added to Yogi						
Variance	0.01	0.02	0.05	0.1	0.3	0.5
SNR of noised Yogi (db)	20.7	17.7	13.8	10.9	6.3	4.4

1. Apply the `imnoise` function (see section 2.7, say, with zero-mean Gaussian noise with variance listed in Table 3) to the original image, F to get the noised image, $G = F + N$, given in (3).

2. Decompose the noised and original images into kl square blocks, each of size $b \times b$, F_{ij} and G_{ij}, $i = 1, 2, \ldots, k$ and $j = 1, 2, \ldots, l$ as in (4). For instance $b = 32$ in my case. Then apply the BlockSvd approach to factor each block into matrices of left and right singular vectors and singular values. Let s_{ijr} and $\tilde{s}_{ijr}$ be the rth singular value of the ijth block for F and G, respectively.

3. According to the BlockSvd approach, most of the noise resides in the singular values. This is why we have to accurately estimate the added noise in the singular values. The difference between the original and the noised images can be obtained by means of the MATLAB program listed in Appendix A by using the noised singular values of each block minus the corresponding original ones as mentioned in subsection 2.9.4.

4. Average the differences of the singular values of the noised image and the original image for each block by the formula

$$\widetilde{w}(r) = \frac{1}{k \times l} \sum_{i=1}^{k} \sum_{j=1}^{l} (\tilde{s}_{ijr} - s_{ijr}). \tag{19}$$

   Then normalize this curve between zero and one. Note that $\tilde{s}_{ijr} \geq s_{ijr}$ since noise increases the singular values [3].

In Figs. 16, 17 and 18, we draw experimentally obtained bar graphs and the normalized continuous target spline curve shown in Fig 21 for comparison purposes.

In the first set of curves shown in Fig. 16, zero-mean Gaussian white noised is added to the Boats and Lena images with variance listed in Table 3 to obtain curves of the normalized and averaged difference between the singular values of the original and the zero-mean Gaussian noised Boats and Lena images, respectively.

In the second set of curves, the previous procedure has been applied to Fp1 and Lena images with salt and pepper noise of density $D = 0 : 0.1 : 0.9$. The normalized

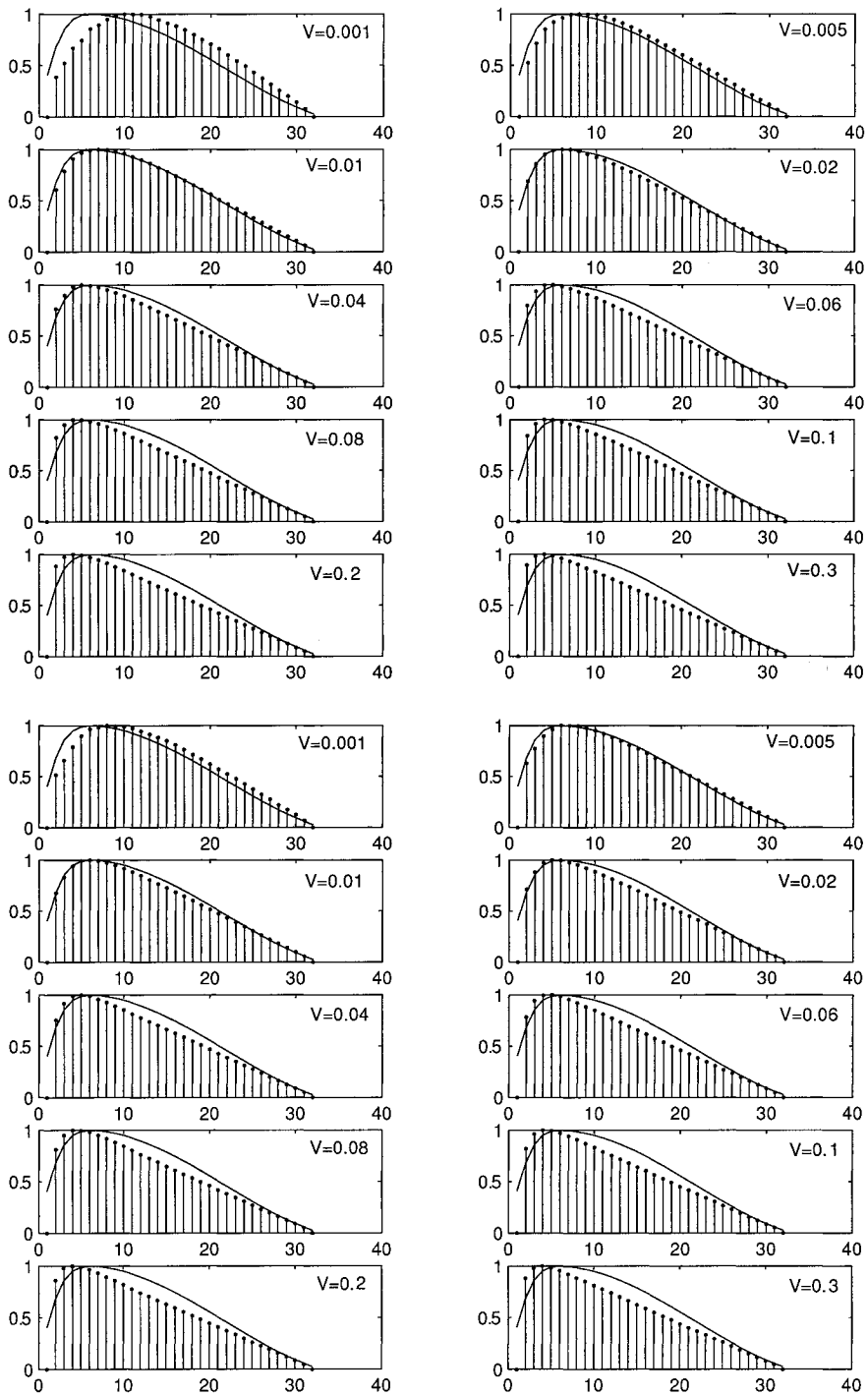


Figure 16: Normalized averaged difference in singular values of mean-zero Gaussian noised and original images with listed variances. Top 10: Boats; bottom 10: Lena. The solid line is the target spline curve shown in Fig. 21.

and averaged difference between the original and noised singular values are shown in Fig. 17.

In the third set of curves, the same procedure has been applied to the Goldhill and Yogi images with speckle noise with variance $V = 0 : 0.1 : 0.9$.The normalized averaged difference between the original and noised singular values are shown in Fig. 18.

Problem definition

By comparing the above graphs it can be seen that the estimated noise curve built with the weighting function (8) does not match very well the shape of the experimental curves shown as bar graphs in Figs. 16, 17, and 18. The solid line in these figures is the target spline curve shown in Fig. 21.

We list several experimental points.

1. Although the location of the top point C changes slightly in each situation, the common shape and scale of the singular value difference curves that we have found experimentally are rather stable except in some special cases. They do not change sensibly with several of images, noise types, block sizes and noise densities. After many experiments, we also found some experimental data for the range of the curves.

2. The value of the first point of the difference is very unstable and does not characterize the behavior of the whole curve. We ignore this unstable point and set the default value of the first point to 0 and then normalize the values of the curve between 0 and 1.

3. A lot of experiments show that changing the block size does not influence the shape of the curves. We compare the exactly same noised image with different block sizes for $b = 16, 32, 64, 128$. The results corroborate this conjecture.

First analysis

The original estimated weight function $w(r)$ given by (8) cannot correctly estimate the actual singular value difference between the noise and the original images. Therefore it directly downgrades the accuracy of the BlockSvd method.

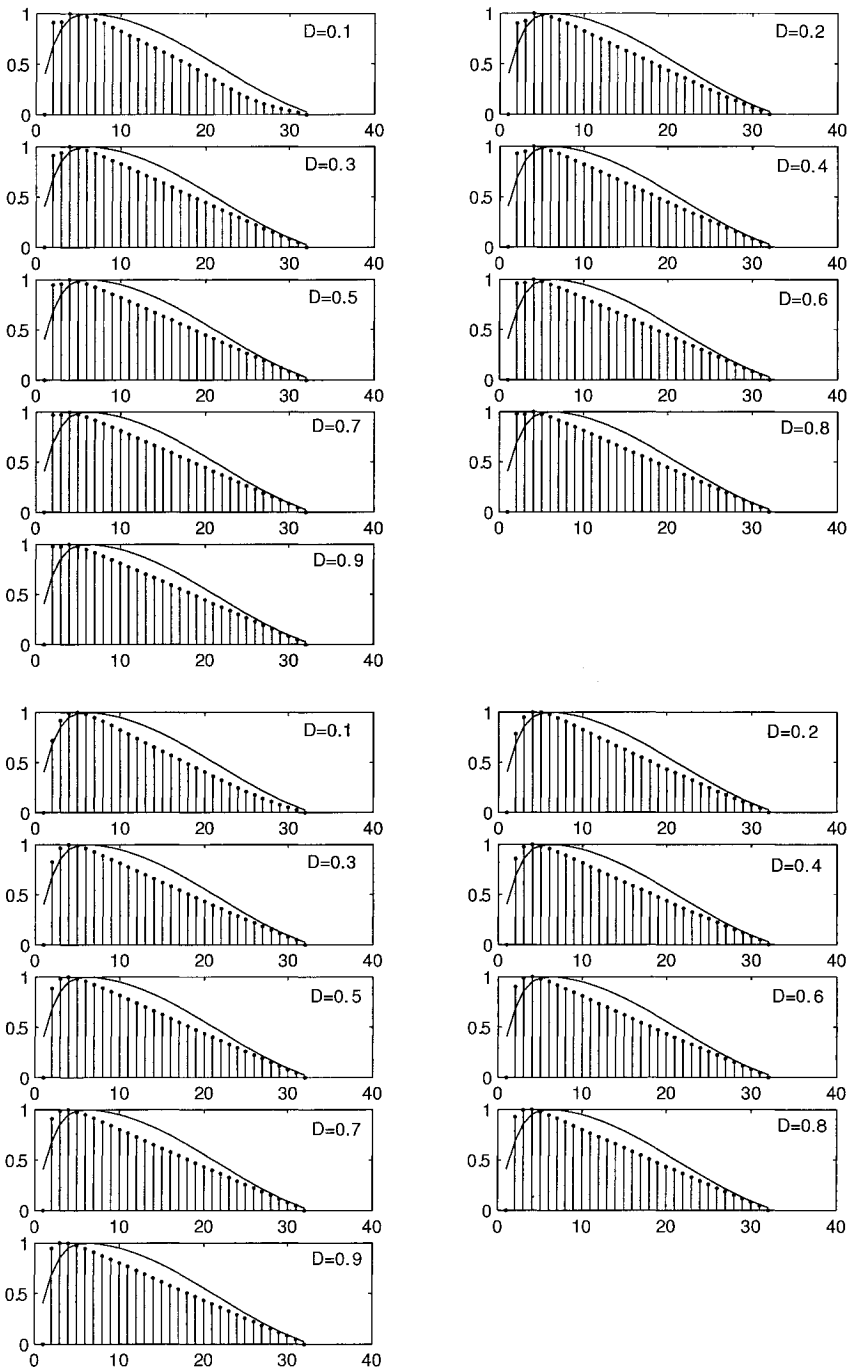


Figure 17: Normalized averaged difference in singular values of salt and pepper noised and original images with density $D = 0 : 0.1 : 0.9$. Top 10: Fp1; bottom 10: Lena. The solid line is the target spline curve shown in Fig. 21.

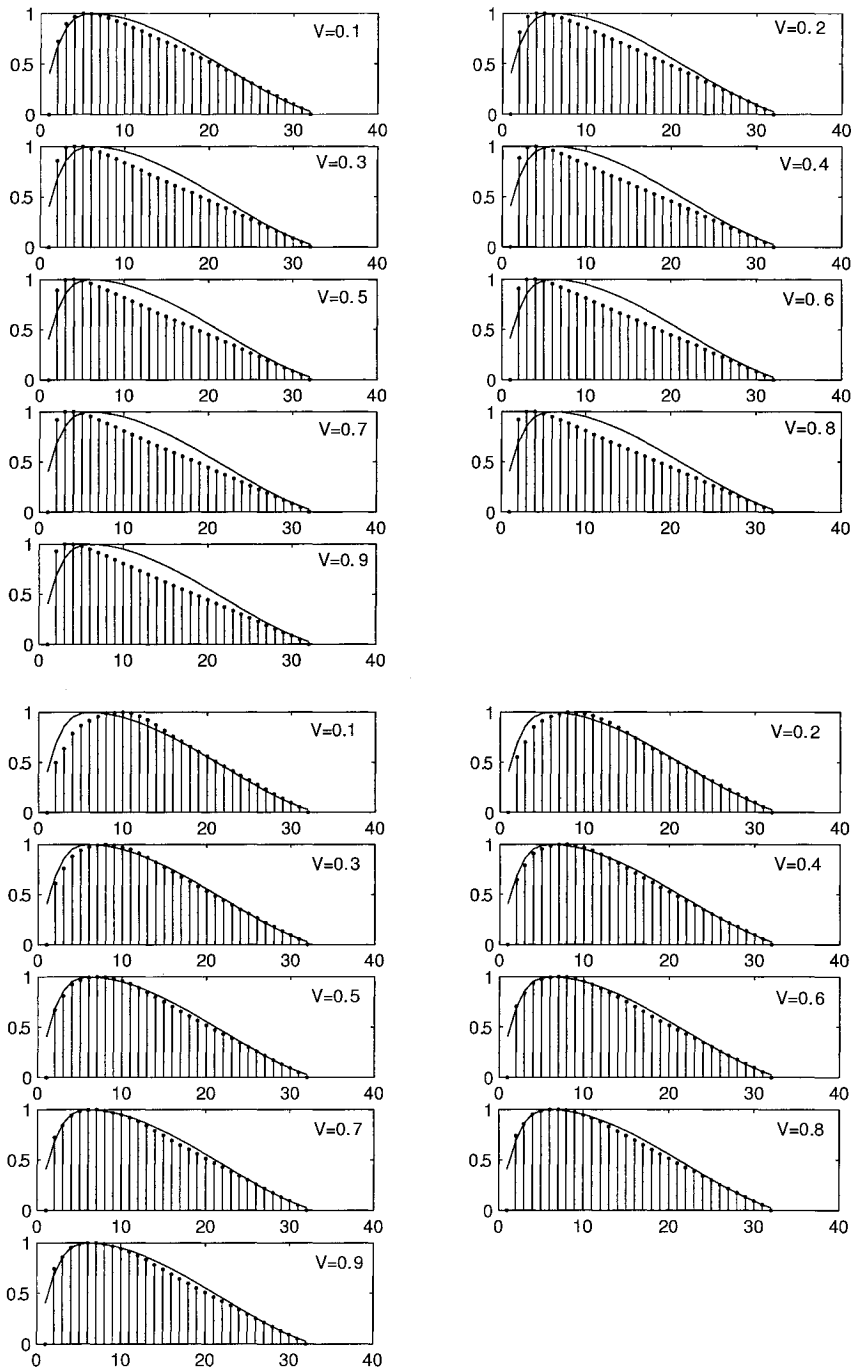


Figure 18: Normalized averaged difference in singular values of speckle noised and original images with with variance $V = 0 : 0.1 : 0.9$. Top 10: Goldhill; bottom 10: Yogi. The solid line is the target spline curve shown in Fig. 21.

3.3 Efficiency problem

Through a comprehensive testing of various images under different noise types, we came across another problem, that is, an adaptive problem. Tables 4–5, 8, and 10 list SNR improvement taken from experimental data in db. A first comparison in denoising performance of the original BlockSvd method, as compared with four other methods, exhibits low efficiency and low adaptability with non Gaussian noise as shown in Tables 4–5.

In this thesis, in order to compare all the tests under the same situation, I shall use the discrete Meyer's wavelet for wavelet denoising, because after many experiments, I found that Meyer's wavelet is slightly more efficient and more stable for image denoising. Other wavelets can be chosen such as Haar or biorthogonal wavelets, but they will not greatly influence the result of the wavelet denoising method.

Table 4: SNR improvement of a given noised image with listed SNR level by five denoising methods.

Zero-mean Gaussian noise added to Boats						
SNR of noised Boats (db)	20.0	17.7	15.9	12.9	8.2	5.1
Median improvement (db)	3.14	4.36	5.13	6.07	6.61	5.99
Averaging improvement (db)	2.60	4.26	5.40	6.93	8.26	8.43
Adaptive improvement (db)	5.02	5.82	6.23	6.69	6.99	7.28
BlockSvd improvement (db)	2.59	3.40	3.85	4.39	4.82	4.89
Wavelet (dmey) improvement (db)	3.02	3.92	4.45	6.47	9.05	9.72
Zero-mean Gaussian noise added to Lena						
SNR of noised Lena (db)	20.3	17.3	15.6	12.6	7.9	4.9
Median improvement (db)	4.94	5.96	6.40	6.85	6.84	6.07
Averaging improvement (db)	4.30	6.17	7.01	8.05	8.81	8.67
Adaptive improvement (db)	6.16	6.78	6.95	7.13	7.25	7.38
BlockSvd improvement (db)	2.65	3.70	4.14	4.62	4.93	4.96
Wavelet (dmey) improvement (db)	4.49	5.92	7.00	8.71	10.4	10.4
Salt and pepper noise added to Fp1						
SNR of noised Fp1 (db)	22.2	12.2	9.2	7.4	6.2	5.2
Median improvement (db)	10.81	17.91	16.18	12.78	9.45	6.75
Averaging improvement (db)	4.97	7.95	7.62	7.14	6.69	6.25
Adaptive improvement (db)	0.42	3.52	5.06	5.51	5.66	5.62
BlockSvd improvement (db)	1.39	3.27	4.16	4.24	4.17	4.04
Wavelet (dmey) improvement (db)	1.92	4.39	8.45	8.21	7.88	7.48
Salt and pepper noise added to Lena						
SNR of noised Lena (db)	19.9	12.7	9.7	6.7	5.1	3.7
Median improvement (db)	9.59	15.93	17.69	16.43	12.76	9.43
AveTab3raging improvement (db)	4.56	7.82	8.31	8.33	8.15	7.84
Adaptive improvement (db)	0.57	2.69	4.10	5.63	6.31	6.58
BlockSvd improvement (db)	1.38	2.72	3.73	4.52	4.65	4.64
Wavelet (dmey) improvement (db)	2.63	3.62	4.56	10.86	10.97	10.77

Table 5: SNR improvement of a given noised image with listed SNR level by five denoising methods. (Continued)

Speckle noise added to Goldhill						
SNR of noised Gold (db)	20.1	17.2	13.4	10.6	7.7	5.0
Median improvement (db)	2.63	3.74	4.62	4.95	5.09	5.05
Averaging improvement (db)	2.74	4.82	6.73	7.56	7.94	7.91
Adaptive improvement (db)	4.38	4.98	5.39	5.57	5.70	5.84
BlockSvd improvement (db)	2.59	3.40	4.03	4.25	4.39	4.47
Wavelet (dmey) improvement (db)	2.14	4.36	6.87	8.24	9.63	10.44
Speckle noise added to Yogi						
SNR of noised Yogi (db)	20.7	17.7	13.8	10.9	6.3	4.4
Median improvement (db)	-3.47	-0.96	1.76	3.17	4.29	4.34
Averaging improvement (db)	-4.09	-1.44	1.74	3.66	5.68	6.21
Adaptive improvement (db)	4.23	4.88	5.14	5.24	5.39	5.59
BlockSvd improvement (db)	-0.73	0.93	2.45	3.21	3.83	4.01
Wavelet (dmey) improvement (db)	-0.97	0.32	2.13	3.29	5.62	6.42

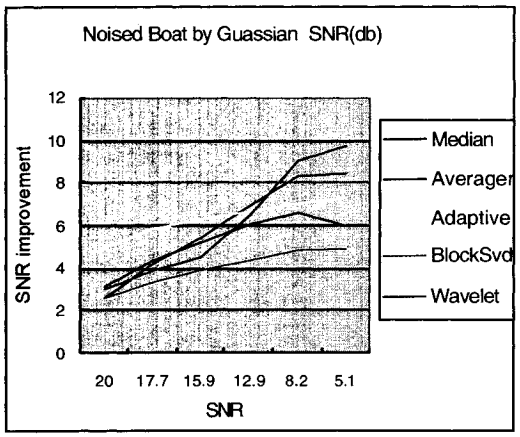
Explanation of the bar graphs in Fig. 19

From the data in Tables 4–5, and the performance curves shown in Fig. 19, we clearly see that the BlockSvd approach (the light blue curve) is not as good as the other approaches listed above for denoising images. We list some experimental issues.

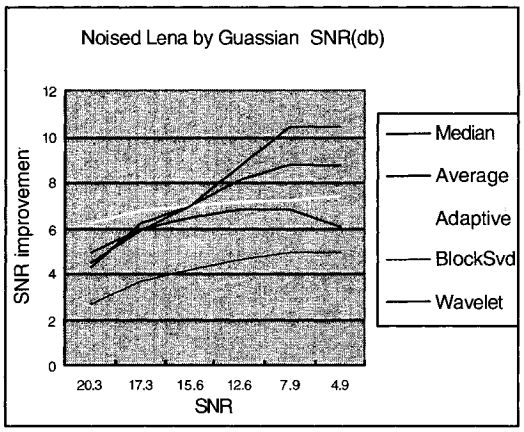
1. The value of SNR is inversely proportional to the noise degree. The lighter the noise we applied, the higher the value of the SNR we can get. On the contrary, SNR improvement is directly proportional to the image quality. The higher the value you get, the better the efficiency.

2. In the zero-mean Gaussian noise case, the gradient of the curves for the BlockSvd is gentle and smooth in different diagrams and SNR improvement reaches its highest point when the heavy noise SNR is between 5db and 8db. This suggests that BlockSvd is better for heavy ordinary Gaussian noise. Compared with other approaches, the performance of the BlockSvd is weak.

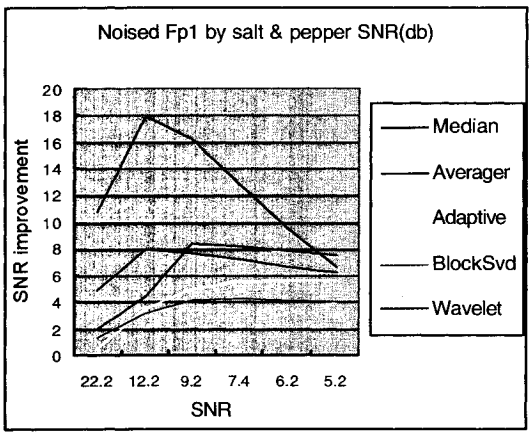
3. Same as for zero-mean Gaussian, the gradient of the BlockSvd is a little sharper



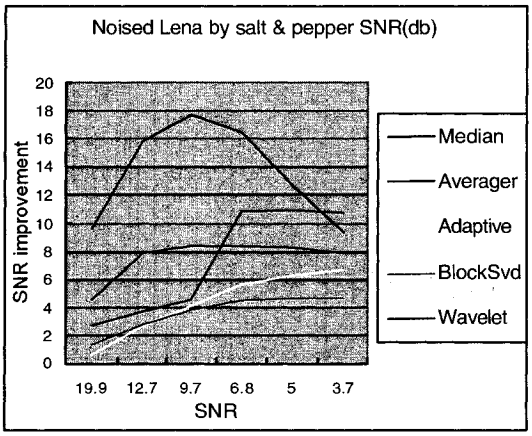
Boat-Gaussian



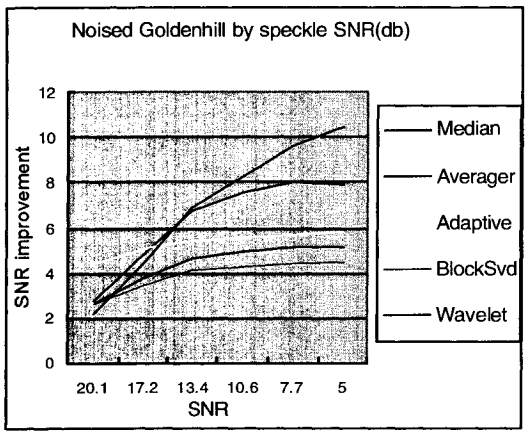
Lena-Gaussian



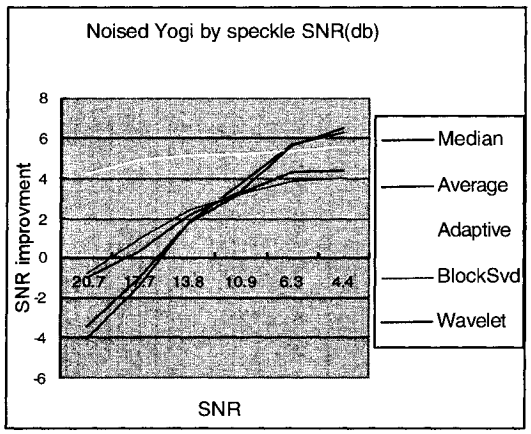
Fp1-salt



Lena-salt



Goldenhill-speckle



Yogi-speckle

Figure 19 Performance comparison

at higher levels for salt and pepper and speckle noises, suggesting that more noise can be cleared in both heavy noise situations. However the light blue curve appears very low in performance.

4. From Fig. 19, we easily discover that the median curve is constantly high, suggesting that median filtering has outstanding performance with salt and pepper noise. Similarly, the adaptive filter has an amazing performance with zero-mean Gaussian noise. The other approaches have a steady performance with different noises.

Second analysis procedure

Generally speaking, BlockSvd algorithm is better for eliminating heavy zero-mean Gaussian noise at SNR around 5db to 8db. It is necessary for us to reanalyze and modify the BlockSvd algorithm in order to further improve its performance and stability.

3.4 Two strategies to improve the original algorithm

According to the above problems, we propose two rules corresponding to each problem in order to improve the efficiency of the BlockSvd algorithm.

3.4.1 First strategy: spline interpolation

The first strategy is to find a new weighting function in order to build an estimated singular value difference curve instead of the previous curve which does a poor matching. As mentioned in Section 3.2, we have already found the general behavior of the practical noise difference curve as shown in Figs. 16, 17 and 18. They are smooth curves similar to the solid-line curve in Fig. 20, but they do not match the original estimated curve.

Now, our goal is to construct a target curve with top point C that fits our problem better than the dashed curve (8) with top point C' as shown in Fig. 20. So we need

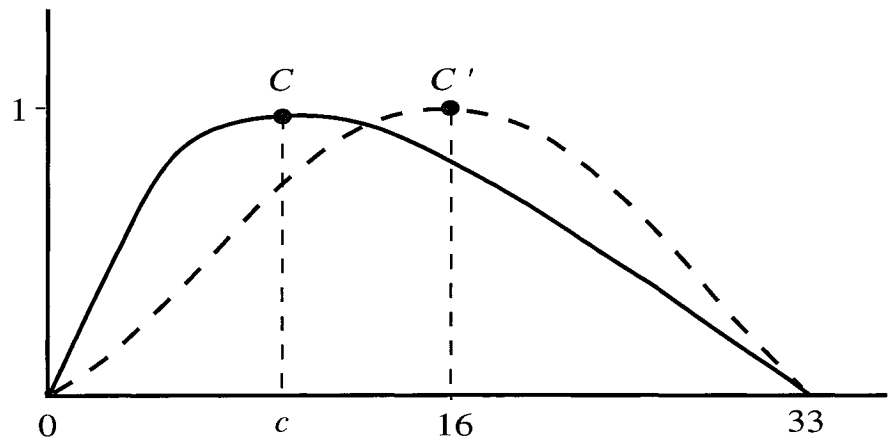


Figure 20: Weight function (8) (dashed line) and target (solid line) singular value difference curve.

to focus attention on numerical interpolation.

Numerical interpolation is used to estimate the value of a function between known data points without knowing the actual function. Interpolation methods can be divided into two main categories [6].

Global interpolation: these methods rely on the construction of a single curve that fits all the data points. This curve is usually given by a high degree polynomial. Although these methods result in smooth curves, they are usually not well suited for engineering applications, as they are prone to severe oscillations and overshoots at intermediate points.

Piecewise interpolation: these methods rely on the construction of a polynomial of low degree between each pair of known data points. If a first degree polynomial is used, it is called *linear interpolation.* For second and third degree polynomials, it is called *quadratic* and *cubic splines*, respectively. The higher the degree of the spline, the smoother the curve. Splines of degree m have continuous derivatives up to order $m-1$ at the data points [6].

Linear interpolation results in straight line segments between each pair of points and all derivatives are discontinuous at the data points. As it never overshoots or oscillates, it is frequently used in chemical engineering despite the fact that the curves are not smooth.

To obtain a smoother curve, cubic splines are frequently recommended. They are generally well behaved and continuous up to the second-order derivative at the data points. Splines are drafting aids used to draw smooth curves through a set of points. Weights are attached at the points to be connected and a flexible strip is shaped around the weights. Thus, splines can fit random sets of data, making numerical analysis possible even when the actual function is not known.

Up to now, our experimental curves give us a key to the solution for the weight function. So let us use the shape of theses curves to design better weight functions.

Consider a collection of given points,

$$\{(x_0, y_0), \quad (x_1, y_1), \quad \ldots, \quad (x_{i-1}, y_{i-1}), \quad (x_i, y_i), \quad (x_{i+1}, y_{i+1}), \quad \ldots, \quad (x_n, y_n)\}.$$

To interpolate between these data points using traditional cubic splines, a third-degree polynomial is constructed between each pair of points. The polynomial to the left of point $P_i = (x_i, y_i)$ is denoted by p_i with value $f_i(x_i)$ at P_i. Similarly, the polynomial to the right of P_i is denoted by p_{i+1} with value $f_{i+1}(x_i)$ at point P_i.

Traditionally, the construction of cubic splines, p_i, $i = 1, 2, \ldots, n$, is based on the following criteria.

- Curves are piecewise third-degree polynomials,

$$p_i(x) = a_i + b_i x + c_i x^2 + d_i x^3. \tag{20}$$

- Curves pass through all given points,

$$p_i(x_i) = p_{i+1}(x_i) = y_i. \tag{21}$$

- The slope, or first-order derivative, is the same for both polynomials on each side of a point.

$$p_i'(x_i) = p_{i+1}'(x_i). \tag{22}$$

- The second-order derivative is the same for both functions on each side of a point,

$$p_i''(x_i) = p_{i+1}''(x_i). \tag{23}$$

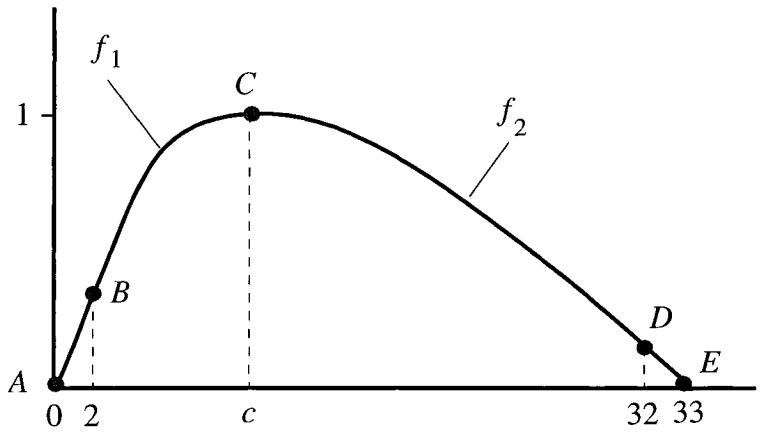


Figure 21: Normalized target spline curve.

This results is in a linear system of $n-1$ equations in $n+1$ unknowns. The remaining two equations are determined by the boundary conditions at the starting and end points, $(x_0, p_1(x_0))$ and $(x_n, p_n(x_n))$, respectively. Natural splines have zero curvature, or zero second derivative at the end points (x_0, y_0) and (x_n, y_n). Clamped splines have zero slope, or zero first derivative at the same endpoints.

Now we go back to consider how to build this new estimating curve by using spline interpolation [6].

It is quite easy to apply spline theory to our system. First, we need to pick five points to construct a system of equations from the above virtual curve. One thing for sure is that the more points one uses for the system of equations, the more accurate and smoother the curve will be. But at the same, time, one trades off the simplicity of the system of equations. In order to reduce the complexity, we separate the above curve into two parts, curve f_1 and curve f_2, and pick the five points as follows and as shown in Fig. 21.

- Point $A = (0, 0)$ is a left extension point for f_1.
- Point $B = (2, y_2)$ is the second point.
- Point $C = (c, 1)$ is the top point connecting f_1 to f_2.
- Point $D = (32, y_{32})$ is the end point.

- Point $E = (33, 0)$ is a right extension point after the last point of f_2.
- We pick the second point instead of the first one, because the value of the first one is very unstable, so we ignore this abnormal point.
- The value of the top point is always 1, since we have already normalized each difference before building this curve.

We build the following third-degree polynomial for the curve $y = f_1(x)$,

$$p_1(x) = a_0 + a_1 x + a_2 x^2 + a_3 x^3. \tag{24}$$

Similarly, we get the curve $y = f_2(x)$ for the third-degree polynomial,

$$p_2(x) = b_0 + b_1 x + b_2 x^2 + b_3 x^3. \tag{25}$$

Thus, to implement the spline theory, we construct the proposed constrained spline equations as follows, according to equations (21), (22), and (23), and (24):

$$p_1(c) = p_2(c), \tag{26}$$
$$p_1'(c) = 0, \tag{27}$$
$$p_1(0) = 0, \tag{28}$$
$$p_1''(c) = p_2''(c), \tag{29}$$
$$p_1'(2) = k_2, \tag{30}$$
$$p_2'(c) = 0, \tag{31}$$
$$p_2(33) = 0, \tag{32}$$
$$p_2'(32) = k_{32}. \tag{33}$$

We note that k_2 is the slope at the second point,

$$k_2 = \frac{y_3 - y_2}{x_3 - x_2},$$

and $x_{31} = 31$, $x_{32} = 32$. Moreover, k_{32} is the slope at the x_{32},

$$k_{32} = \frac{y_{31} - y_{32}}{x_{31} - x_{32}}.$$

We obtain an unnormalized target spline curve with top point $C = (c, 1.4)$. This amounts to take the parameter $p_1 = 5.9$ in (7).

Now we have the system of spline equations (26)–(33). So we can convert them into the corresponding matrix equation $M\mathbf{u} = \mathbf{r}$ with

$$\mathbf{u} = \begin{bmatrix} a_3 & a_2 & a_1 & a_0 & b_3 & b_2 & b_1 & b_0 \end{bmatrix}^T \tag{34}$$

and

$$M = \begin{bmatrix} c^3 & c^2 & c & 1 & -c^3 & -c^2 & -c & -1 \\ 3c^2 & 2c & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 6c & 2 & 0 & 0 & -6c & -2 & 0 & 0 \\ 3 \times 4 & 2 \times 2 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 3c^2 & 2c & 1 & 0 \\ 0 & 0 & 0 & 0 & 33^3 & 33^2 & 33 & 1 \\ 0 & 0 & 0 & 0 & 3 \times 32^2 & 2 \times 32 & 1 & 0 \end{bmatrix} \tag{35}$$

For the fifth row, we used equation (30), so the second point is $B = (2, y_2)$. For the eighth row, we used equation (33), so the fifth point is $E = (32, y_{32})$. Thus,

$$\mathbf{r} = \begin{bmatrix} 0 & 0 & 0 & 0 & k_2 & 0 & 0 & k_{32} \end{bmatrix}^T. \tag{36}$$

The solution $\mathbf{u}$ is easily obtained by the MATLAB left inverse operator \,

$$\mathbf{u} = M \setminus \mathbf{r}. \tag{37}$$

Eventually, we obtain the coefficient vector $\mathbf{u}$ for the spline curves. The coefficients of $p_1(x)$ are a_0, a_1, a_2, and a_3, and the coefficients of $p_2(x)$ are b_0, b_1, b_2, and b_3.

We remark that, in the above calculation, k_2, k_{32}, and the top point C are the only parameters used for determining the accuracy of the new weighting function.

After many tests we collected lots of experimental data range for their range. However, in each case, one can try to adjust each value to obtain the best denoising result. We also set default values as in the following table.

Up to now we have solved the first problem logically, but does it really work? Is the new weighting function from spline curves really better than the original one?

Table 6: Parameter range.

Parameter	Ideal range	Default
Abscissa c of top point C	$[5, 9]$	6
Slope k_2 at $(2, y_2)$	$[0.28, 0.42]$	0.31
Slope k_{32} at $(32, y_{32})$	$[-0.01, -0.06]$	-0.04

Table 7 lists the SNR gain in db of the BlockSvd and the Spline method applied to Boats and Lena with zero-mean Gaussian noise, Fp1 and Lena with salt and pepper noise, and Goldhill and Yogi with speckle noise, at six given SNR's in db, respectively. The table also lists the percentage improvement rates of the Spline method over the BlockSvd method.

- Definition: The improvement rate is equal to 100 times the ratio of the SNR improvement (15) with the spline method over the SNR improvement with the BlockSvd method.

- In my testing, I used the default value for fixing k_2, k_{32}, and the top point C. In different situations, one can adjust these parameters according the above parameter range table for possibly obtaining a better gain than the ones listed here. We put all the above data into bar graphs in Fig. 22.

Additional testing data are listed in Tables 15, 16 and 17 of Appendix B for the Fp2, Fp3 and Barb images with zero-mean Gaussian, salt and pepper and speckle noises to given SNR levels. The testing has been done with the seven denoising methods considered in this work.

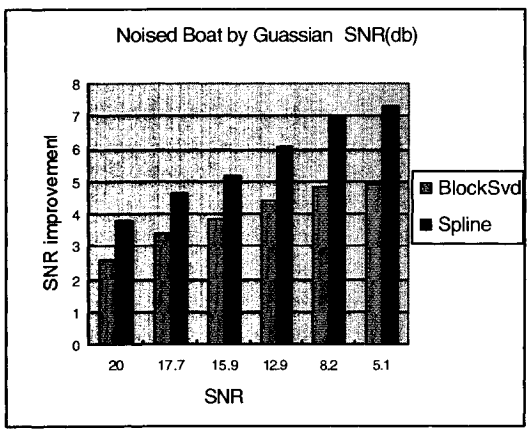
Explanation of the bar graphs in Fig. 22.

From the data in Tables 7 and the performance bar graphs shown in Fig. 22, we have the following points.

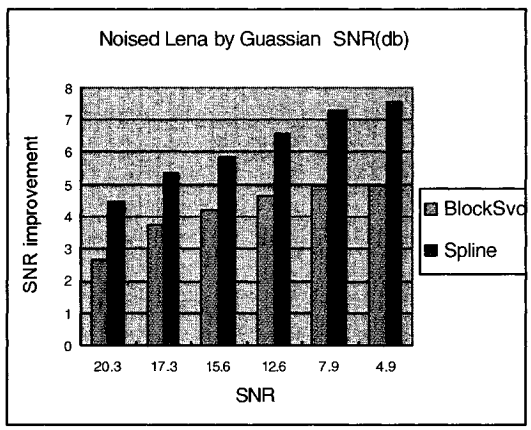
1. For zero-mean Gaussian noise, we find that both the spline and the BlockSvd approaches are good to remove heavy noise. It is best with SNR value around

Table 7: SNR improvement rate of the spline method over the BlockSvd method for a given noised image with listed SNR level.

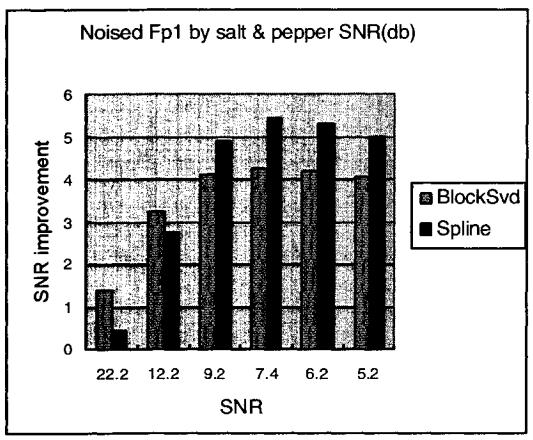
Zero-mean Gaussian noise added to Boats						
SNR of noised Boats (db)	20.0	17.7	15.9	12.9	8.2	5.1
BlockSvd improvement (db)	2.59	3.40	3.85	4.39	4.82	4.89
Spline improvement (db)	3.77	4.59	5.17	6.04	6.98	7.28
Improvement rate (%)	146	135	144	138	145	149
Zero-mean Gaussian noise added to Lena						
SNR of noised Lena (db)	20.3	17.3	15.6	12.6	7.9	4.9
BlockSvd improvement (db)	2.65	3.70	4.14	4.62	4.93	4.96
Spline improvement (db)	4.42	5.33	5.83	6.58	7.26	7.58
Improvement rate (%)	167	144	141	142	147	153
Salt and pepper noise added to Fp1						
SNR of noised Fp1 (db)	22.2	12.2	9.2	7.4	6.2	5.2
BlockSvd improvement (db)	1.39	3.25	4.12	4.25	4.19	4.02
Spline improvement (db)	0.44	2.76	4.89	5.43	5.32	4.99
Improvement rate (%)	32	85	119	128	127	124
Salt and pepper noise added to Lena						
SNR of noised Lena (db)	19.8	12.7	9.7	6.7	5.0	3.7
BlockSvd improvement (db)	1.39	2.74	3.79	4.54	4.63	4.63
Spline improvement (db)	0.95	1.84	4.02	6.33	6.82	6.98
Improvement rate (%)	69	67	107	139	147	150
Speckle noise added to Goldhill						
SNR of noised Gold (db)	20.1	17.2	13.4	10.5	7.7	5.0
BlockSvd improvement (db)	2.61	3.40	4.0	4.23	4.39	4.48
Spline improvement (db)	3.18	3.93	4.77	5.25	5.63	5.78
Improvement rate (%)	122	116	119	124	128	129
Speckle noise added to Yogi						
SNR of noised Yogi (db)	20.7	17.7	13.8	10.9	6.3	4.4
BlockSvd improvement (db)	-0.73	0.93	2.45	3.21	3.83	4.01
Spline improvement (db)	3.16	3.45	3.80	4.24	4.81	5.1
Improvement rate (%)	380	372	155	132	126	127



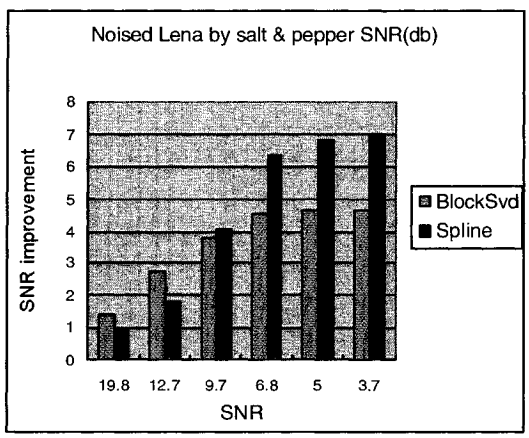
Boat-gaussian



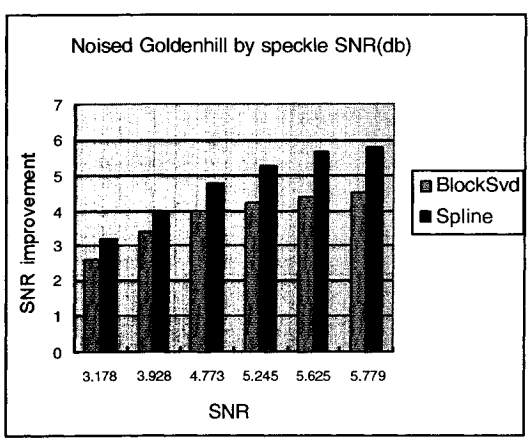
Lena-gaussian



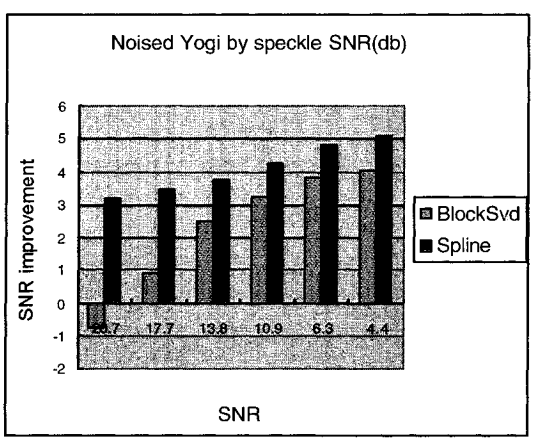
Fp1-salt



Lena-salt



Goldenhill-speckle



Yogi-speckle

Figure 22 Results for the Spline and BlockSvd methods

5db to 8db. The performance of the Spline method (dark color bars) is much better than the original BlockSvd under the same situation.

2. For salt and pepper noise, both the spline and the BlockSvd are good for filtering out more noise in the heavy noise situation than light noise. The improved spline method shows greater ability and stability than the original BlockSvd.

3. For the speckle noise, the result is similar to salt and pepper situation. Spline obtains more efficient results than the original BlockSvd. BlockSvd method could lead to a negative SNR improvement result. It all depends on the quality of the images.

3.4.2 Summary for the spline approach

After analyzing the above graphs and tables, we find that the new spline method is an upgraded BlockSvd method, and it is more stable and suitable for the different noise situations. Generally speaking, the entire performance could be improved, on the average, by 20 to 30%. Also, you can try to adjust the three parameters k_2, k_{32} and C, in order to obtain the best results.

To follow up, we rewrite the entire spline filtering algorithm.

Algorithm (Spline filtering method)

1. Divide a given image matrix, $X \in \mathbb{R}^{m\times n}$, into $b \times b$ submatrices G_{ij}. By default $b = 32$.

2. Decompose each submatrix G_{ij} by the SVD decomposition:

$$G_{ij} = U_{ij} S_{ij} V_{ij}^T,$$

where U_{ij} is the $b \times b$ unitary matrix of left singular vectors, S_{ij} is the $b \times b$ diagonal matrix of singular values, and V_{ij} is the $b \times b$ unitary matrix of right singular vectors.

3. Estimate the singular values difference $w(r)$ (new weighting function) through the spline interpolation method and build the difference curves f_1 and f_2 by

means of the solution,

$$\mathbf{u} = M \setminus \mathbf{r},$$

of system (26)–(33),

$$M\mathbf{u} = \mathbf{r}.$$

4. Eliminate noise in the singular values by the formula

$$\hat{s}_{ijr} = s_{ijr} - p_1 n_s w(r).$$

5. Reconstruct each filtered block by the formula

$$\widehat{F}_{ij} = \widehat{U}_{ij}\widehat{S}_{ij}\widehat{V}_{ij}^T, \qquad i = 1, 2, \ldots, k, \quad j = 1, 2, \ldots, l.$$

Experimental results have proved that after reducing the noise from the singular values, we already have better results than reducing the estimated noise from the right and left singular vectors again. Comparing with the BlockSvd, we do not implement the FFT transformation to reduce noise from the singular vectors since, after several tests, this procedure was found to have adverse effect on the reconstructed image. Thus, the spline algorithm is also faster than BlockSvd.

3.4.3 Second strategy: hybrid approach

The application of the spline interpolation into BlockSvd has greatly improved the performance and efficiency of the denoising procedure. It is a big leap, but we have not ended yet. Meanwhile, we realize that comparing with the other methods such as wavelet, adaptive filtering, media filtering, the performance of the spline method is far from perfect. There is still room left for further improvement. Let us start by reviewing the performance in Table 8 first.

- For zero-mean Gaussian noise: Imp rate = SNR improvement of spline/SNR improvement of the adaptive filter $\times 100$.
- For salt and pepper noise: Imp rate = SNR improvement of spline/SNR improvement of the median filter $\times 100$.

Table 8: SNR improvement rate of the spline method over the adaptive method for a given noised image with listed SNR level.

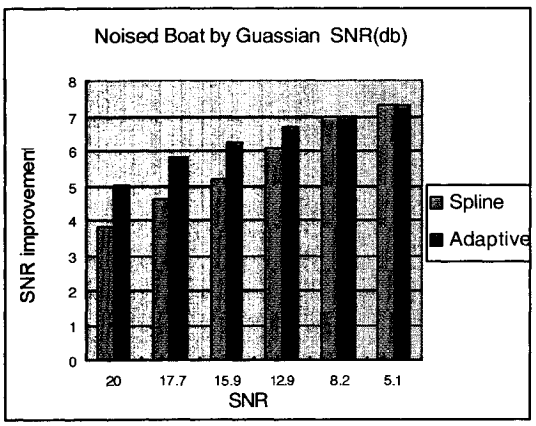
Zero-mean Gaussian noise added to Boat						
SNR of noised Boats (db)	20.0	17.7	15.9	12.9	8.2	5.1
Adaptive improvement (db)	5.02	5.82	6.23	6.69	6.99	7.28
Spline improvement (db)	3.77	4.59	5.17	6.04	6.98	7.28
Improvement rate (%)	75	78	83	90	100	100
Zero-mean Gaussian noise added to Lena						
SNR of noised Lena (db)	20.3	17.3	15.6	12.6	7.9	4.9
Adaptive improvement (db)	6.16	6.78	6.95	7.13	7.25	7.38
Spline improvement (db)	4.42	5.33	5.83	6.58	7.26	7.58
Improvement rate (%)	71	91	84	92	100	101
Salt and pepper noise added to Fp1						
SNR of noised Fp1 (db)	22.2	12.2	9.2	7.4	6.2	5.2
Median improvement (db)	10.81	17.91	16.18	12.78	9.45	6.75
Spline improvement (db)	0.44	2.76	4.89	5.43	5.32	4.99
Improvement rate (%)	4	15	30	43	56	74
Salt and pepper noise added to Lena						
SNR of noised Lena (db)	19.8	12.7	9.7	6.7	5.0	3.7
Median improvement (db)	9.59	15.93	17.69	16.43	12.76	9.43
Spline improvement (db)	0.95	1.84	4.02	6.33	6.82	6.98
Improvement rate (%)	9	12	23	39	53	74
Speckle noise added to Goldhill						
SNR of noised Gold (db)	20.1	17.2	13.4	10.6	7.7	5.0
Wavelet (dmey) improvement (db)	2.14	4.36	6.87	8.24	9.63	10.44
Spline improvement (db)	3.18	3.93	4.77	5.25	5.63	5.78
Improvement rate (%)	147	90	69	64	58	55
Speckle noise added to Yogi						
SNR of noised Yogi (db)	20.7	17.7	13.8	10.9	6.3	4.4
Wavelet (dmey) improvement (db)	-0.97	0.32	2.13	3.29	5.62	6.42
Spline improvement (db)	3.16	3.45	3.80	4.24	4.81	5.1
Improvement rate (%)	400	377	178	128	86	79

- For speckle noise: Imp rate = SNR improvement of spline/SNR improvement of wavelet (dmey) $\times 100$.
- In my testing, I used the default value for k_2, k_{32}, and the top point C. In different situations, one can adjust these parameters according to the parameter range listed in Table 6, and might obtain better result than mine. We put all the above data into bar graphs in Fig. 13.
- In this thesis, in order to compare all the tests under the same situation, I picked the same wave type (Meyer's wavelet) to use for wavelet denoising because after lots of experiments, I found Meyer's wavelet to be more efficient and more stable for image denoising. One can choose other types of wavelets such as Haar or biorthogonal wavelets, but they will not influence much the result of wavelet denoising methods.

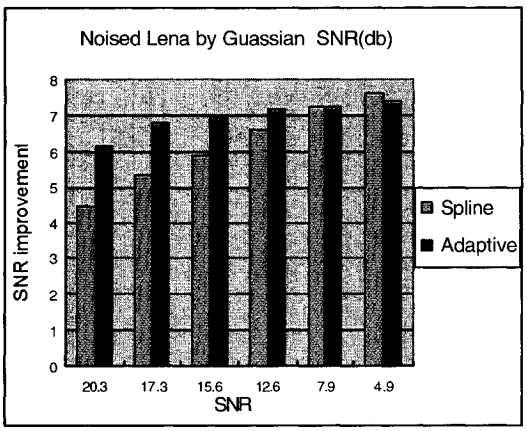
Explanation of the bar graphs in Fig. 23.

From the data in Tables 8 and the correspondent performance curves shown in bar graphs in Fig. 23, we make the following comments.

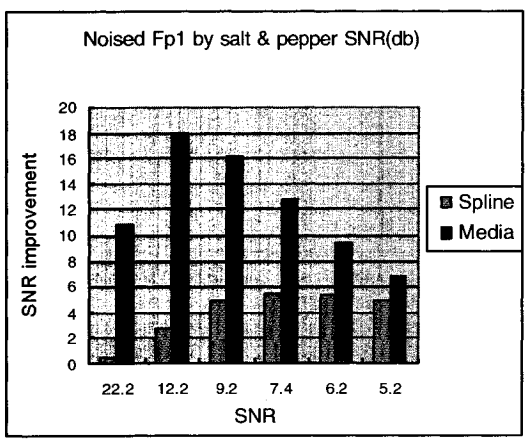
1. For zero-mean Gaussian noise, we picked the best approach, which is the adaptive filter from Fig. 19, to compare with the new spline algorithm. It is clearly seen from Fig. 23 that the performance of the spline method (light blue bars) cannot beat the adaptive filter (dark color bars) in light zero-mean Gaussian noise situations. But they already have very close performance in dealing with both light and heavy noises.
2. For salt and pepper noise, we compare the median filter with the new spline algorithm. Although the spline is an improved approach over the original BlockSvd, its performance is still weak, if compared with the median filter.
3. For speckle noise, the spline has a better performance in light noise situations, whereas wavelet performs better in heavy noise cases. The wavelet method could lead to negative SNR improvement. It all depends on the quality of the images.



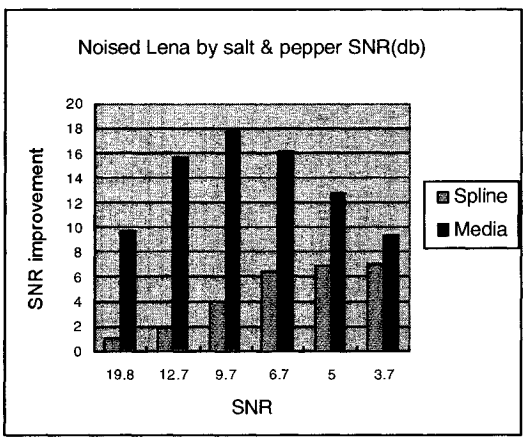
Boat-Gaussian



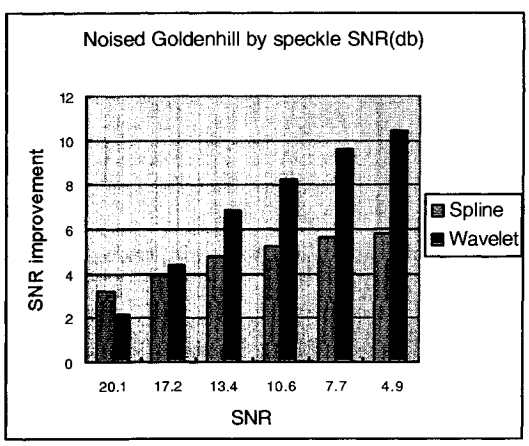
Lena-Gaussian



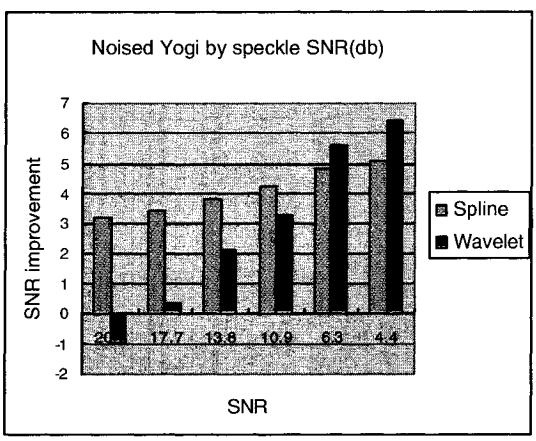
Fp1-salt



Lena-salt



Goldenhill-speckle



Yogi-speckle

Figure 23 Results of the Spline and other methods

The above tables and bar graphs show that, compared with the other denoising methods such as median filtering, adaptive filtering, and wavelets, the performance and stability of the spline method still looks a little weak, especially with non-Gaussian noise. In order to overcome this problem, we propose a hybrid method which combines the spline method based on singular value decomposition with a wavelet global threshold denoising.

We need to further eliminate the noise after applying spline denoising process. According to the previous introduction of the wavelet denoising, the way the wavelet decomposition is completed differs from the way the singular value decomposition is done, and, from the experimental data we found that wavelet shows a little weak performance in the case of zero-mean Gaussian noise compared to other types of noise. On the contrary, the spline approach works in the opposite way. In this sense, this suggests that combining these two algorithms together could work on different sides and eliminate the residual noise. So a hybrid method could maximally make up for the weakness on individual methods. For this reason, we think that the idea of hybrid methods can be widely used in scientific research.

Now we are proposing a hybrid method which combines the spline method based on singular value decomposition with wavelet global threshold denoising. This method consists in the following steps.

Hybrid spline-wavelet algorithm

Procedure 1 (Spline)

- Decompose the image blocks by the singular value decomposition method.
- Estimate the singular values difference w(r) (weighting function) through the spline interpolation method, building the difference curves f_1 and f_2 corresponding to each coefficient from the vector $\mathbf{u}$,

$$\mathbf{u} = M \setminus \mathbf{r},$$

where M is the coefficient matrix of equations (26)–(33) and $\mathbf{r}$ is the vector with parameters, k_2, k_{32} and C (default). In addition, the backslash is the MATLAB left matrix inverse operator.

- Eliminate noise in the singular values for each block.
- Reconstruct the image.

Procedure 2 (Wavelet)

- Use wavelet to decompose the filtered image again.
- Apply the `ddencmp` function to detect default values for all the general procedures related to denoising:

```
[THR,SORH,KEEPAPP] = ddencmp(IN1,'wv',X)
```

where `THR` is the threshold, `SORH` is for soft or hard threshold, `KEEPAPP` allows one to keep approximation coefficients.

- Implement the `wdencmp` function to perform a second time denoising process on the image,

```
Filtered image=wdencmp('glb',X,'wname',N,THR,SORH)
```

where `glb` is the global threshold, `X` is the image, `wname` is the wavelet type, `N` is the denoised level, `SORH` and `THR` are detected by the above `ddencmp` function.

- Reconstruct the image.

In Tables 9 we give the improvement rate of the spline-wavelet over the spline for three different additive noises and different figures.

In Tables 10 and 11, we list the improvement of the seven methods considered in this thesis over the BlockSvd.

- For each noise: Imp rate = SNR improvement of spline-wavelet/SNR improvement of spline.

Table 9: SNR improvement rate of spline-wavelet over spline for a given noised image with listed SNR level.

Zero-mean Gaussian noise added to Boats						
SNR of noised Boats (db)	20.0	17.7	15.9	12.9	8.2	5.1
Spline improvement (db)	3.77	4.59	5.17	6.04	6.98	7.28
Spline-wavelet improvement (db)	3.59	4.85	4.99	6.81	9.61	10.51
Improvement rate (%)	95	105	79	113	138	144
Zero-mean Gaussian noise added to Lena						
SNR of noised Lena (db)	20.3	17.3	15.6	12.6	7.9	4.9
Spline improvement (db)	4.42	5.33	5.83	6.58	7.26	7.58
Spline-wavelet improvement (db)	5.11	6.03	7.12	8.75	10.9	11.3
Improvement rate (%)	116	113	132	133	150	149
Salt and pepper noise added to Fp1						
SNR of noised Fp1 (db)	22.2	12.2	9.2	7.4	6.2	5.2
Spline improvement (db)	0.44	2.76	5.0	5.43	5.32	4.99
Spline-wavelet improvement (db)	2.20	8.21	8.48	7.67	7.06	6.57
Improvement rate (%)	497	297	169	141	133	131
Salt and pepper noise added to Lena						
SNR of noised Lena (db)	19.9	12.7	9.7	6.7	5.1	3.7
Spline improvement (db)	0.95	1.84	4.02	6.33	6.82	6.98
Spline-wavelet improvement (db)	2.87	5.99	9.84	10.39	10.48	10.33
Improvement rate (%)	300	326	245	164	154	148
Speckle noise added to Goldhill						
SNR of noised Gold (db)	20.1	17.2	13.4	10.6	7.7	5.0
Spline improvement (db)	3.18	3.93	4.77	5.25	5.63	5.78
Spline-wavelet improvement (db)	3.51	5.12	6.81	8.22	9.21	9.63
Improvement rate (%)	110	130	142	156	164	166
Speckle noise added to Yogi						
SNR of noised Yogi (db)	20.7	17.7	13.8	10.9	6.3	4.4
Spline improvement (db)	3.16	3.45	3.80	4.24	4.81	5.1
Spline-wavelet improvement (db)	1.93	2.5	3.44	4.35	5.67	6.21
Improvement rate (%)	61	72	91	103	118	122

Table 10: SNR improvement of a given noised image with listed SNR level by the seven denoising methods considered in this thesis.

Zero-mean Gaussian noise added to Boats						
SNR of noised Boats (db)	20.0	17.7	15.9	12.9	8.2	5.1
Median improvement (db)	3.14	4.36	5.13	6.07	6.61	5.99
Averaging improvement (db)	2.60	4.26	5.40	6.93	8.26	8.43
Adaptive improvement (db)	5.02	5.82	6.23	6.69	6.99	7.28
BlockSvd improvement (db)	2.59	3.40	3.85	4.39	4.82	4.89
Wavelet (dmey) improvement (db)	3.02	3.92	4.45	6.47	9.05	9.72
Spline improvement (db)	3.77	4.59	5.17	6.04	6.98	7.28
Spline-wavelet improvement (db)	3.59	4.85	4.99	6.81	9.61	10.51
Zero-mean Gaussian noise added to Lena						
SNR of noised Lena (db)	20.3	17.3	15.6	12.6	7.9	4.9
Median improvement (db)	4.94	5.96	6.40	6.85	6.84	6.07
Averaging improvement (db)	4.30	6.17	7.01	8.05	8.81	8.67
Adaptive improvement (db)	6.16	6.78	6.95	7.13	7.25	7.38
BlockSvd improvement (db)	2.65	3.70	4.14	4.62	4.93	4.96
Wavelet (dmey) improvement (db)	4.49	5.92	7.00	8.71	10.4	10.4
Spline improvement (db)	4.42	5.33	5.83	6.58	7.26	7.58
Spline-wavelet improvement (db)	5.11	6.03	7.12	8.75	10.9	11.3
Salt and pepper noise added to Fp1						
SNR of noised Fp1 (db)	22.2	12.2	9.2	7.4	6.2	5.2
Median improvement (db)	10.81	17.91	16.18	12.78	9.45	6.75
Averaging improvement (db)	4.97	7.95	7.62	7.14	6.69	6.25
Adaptive improvement (db)	0.42	3.52	5.06	5.51	5.66	5.62
BlockSvd improvement (db)	1.39	3.27	4.16	4.24	4.17	4.04
Wavelet (dmey) improvement (db)	1.92	4.39	8.45	8.21	7.88	7.48
Spline improvement (db)	0.44	2.76	5.0	5.43	5.32	4.99
Spline-wavelet improvement (db)	2.20	8.21	8.48	7.67	7.06	6.57
Salt and pepper noise added to Lena						
SNR of noised Lena (db)	19.9	12.7	9.7	6.7	5.1	3.7
Median improvement (db)	9.59	15.93	17.69	16.43	12.76	9.43
Averaging improvement (db)	4.56	7.82	8.31	8.33	8.15	7.84
Adaptive improvement (db)	0.57	2.69	4.10	5.63	6.31	6.58
BlockSvd improvement (db)	1.38	2.72	3.73	4.52	4.65	4.64
Wavelet (dmey) improvement (db)	2.63	3.62	4.56	10.86	10.97	10.77
Spline improvement (db)	0.95	1.84	4.02	6.33	6.82	6.98
Spline-wavelet improvement (db)	2.87	5.99	9.84	10.39	10.48	10.33

Table 11: SNR improvement of a given noised image with listed SNR level by the seven denoising methods considered in this thesis. (Continued)

Speckle noise added to Goldhill						
SNR of noised Gold (db)	20.1	17.2	13.4	10.6	7.7	5.0
Median improvement (db)	2.63	3.74	4.62	4.95	5.09	5.05
Averaging improvement (db)	2.74	4.82	6.73	7.56	7.94	7.91
Adaptive improvement (db)	4.38	4.98	5.39	5.57	5.70	5.84
BlockSvd improvement (db)	2.59	3.40	4.03	4.25	4.39	4.47
Wavelet (dmey) improvement (db)	2.14	4.36	6.87	8.24	9.63	10.44
Spline improvement (db)	3.18	3.93	4.77	5.25	5.63	5.78
Spline-wavelet improvement (db)	3.51	5.12	6.81	8.22	9.21	9.63
Speckle noise added to Yogi						
SNR of noised Yogi (db)	20.7	17.7	13.8	10.9	6.3	4.4
Median improvement (db)	-3.47	-0.96	1.76	3.17	4.29	4.34
Averaging improvement (db)	-4.09	-1.44	1.74	3.66	5.68	6.21
Adaptive improvement (db)	4.23	4.88	5.14	5.24	5.39	5.59
BlockSvd improvement (db)	-0.73	0.93	2.45	3.21	3.83	4.01
Wavelet (dmey) improvement (db)	-0.97	0.32	2.13	3.29	5.62	6.42
Spline improvement (db)	3.16	3.45	3.80	4.24	4.81	5.1
Spline-wavelet improvement (db)	1.93	2.5	3.44	4.35	5.67	6.21

- In my testing, I used the default value for k_2, k_{32}, and the top point C. In different situations, one can adjust these parameters according to the parameter range listed in Table 6 and might obtain better results than mine. Once more, we put all the above data into bar graphs in Fig. 14.

- In my testing, I compared all the tests under the same situation. For this reason, I picked the same wavelet (Meyer's wavelet) to denoise different types of noise. After many experiments, I found Meyer's wavelet to be more efficient and more stable for image denoising. One can pick other types of wavelets such as Haar or biorthogonal wavelets, but they will not influence the wavelet results very much.

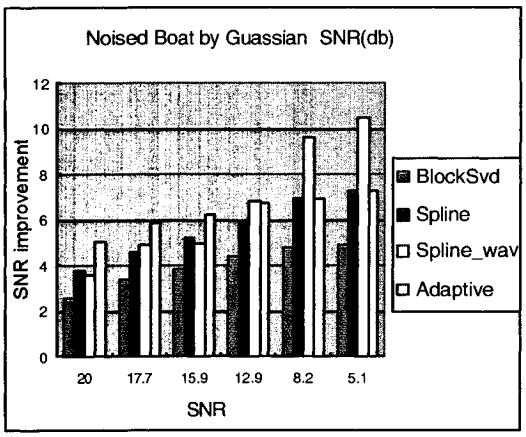
Explanation of the bar graphs in Fig. 24.

From the data in Tables 10 and 11 and the performance bar graphs shown in Fig. 24 we have the following points.

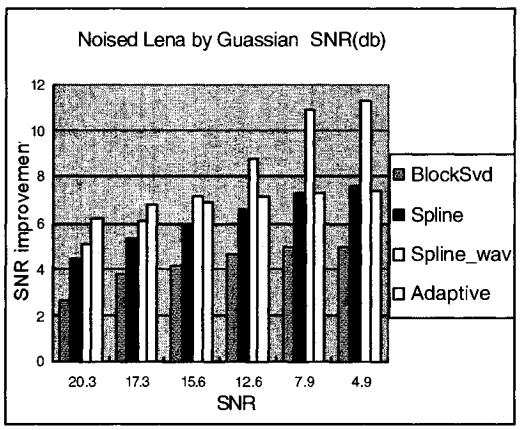
1. For zero-mean Gaussian noise, the new hybrid algorithm's performance is close to the adaptive filter and much better than the original spline and BlockSvd alone.

2. For salt and pepper noise, although the hybrid algorithm still cannot beat the median filter, it has been improved a lot from the original spline and BlockSvd approaches. Since salt and pepper noise is a very unique type of noise, We think that the current result is completely acceptable.

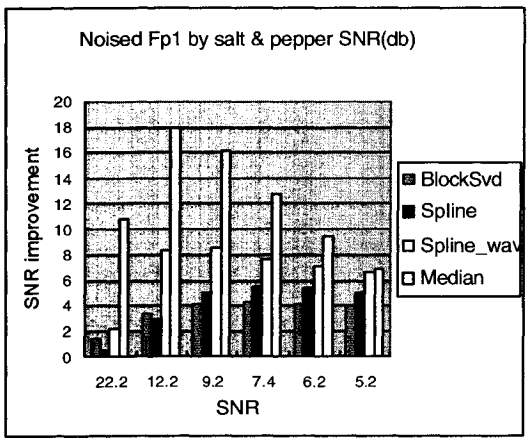
3. For speckle noise, since the hybrid method combines splines with wavelets, we see that it inherits advantages from both methods. For light noise, the new hybrid algorithm appears to have a similar behavior as the spline method, whereas the performance of the wavelet in heavy noise, suggests it is a valuable new approach.



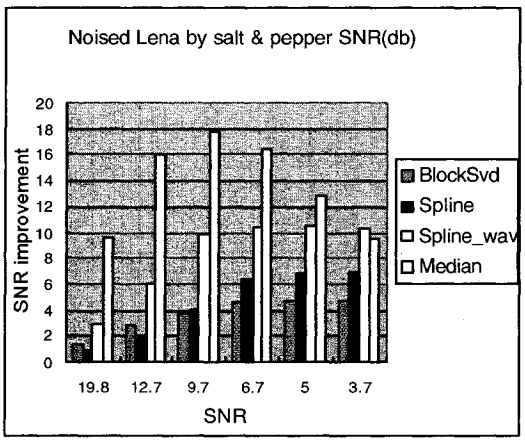
Boat-gaussian



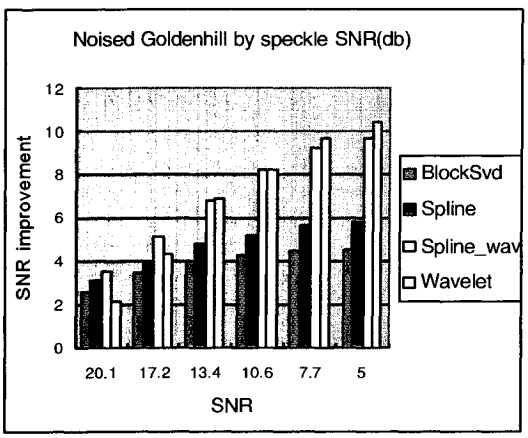
Lena-gaussian



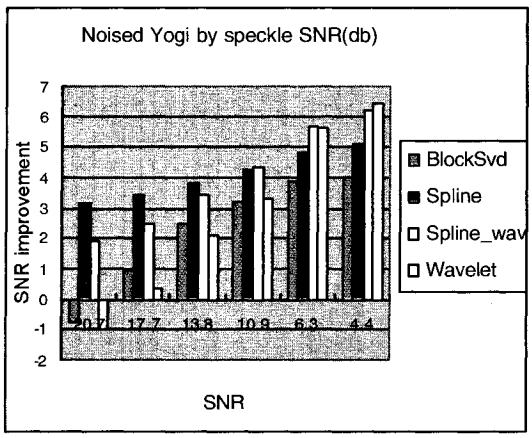
Fp1-salt



Lena-salt



Golden-speckle



Yogi-speckle

Figure 24 Spline_Wavelet and other methods

3.4.4 Summary for the hybrid algorithm

1. Our spline method based on SVD decomposition depends on the data and cannot deal with data in time-frequency domain. Because our hybrid method contains wavelet analysis, which is a kind of time-frequency analysis, in most cases, our hybrid spline-wavelet method can be a better approach than spline. However it still depends on the noise type and image quality.

2. Generally speaking, the spline-wavelet method is well suited for different noise situation. For some specific images, such as Yogi, the spline-wavelet method does not work very well. It is even worse than spline alone. This is quite reasonable since different images behave differently.

3. Median filtering is a nonlinear operation designed for reducing "salt and pepper" noise. Median filtering is more effective than convolution when the goal is to simultaneously reduce noise and preserve edges. So, in most of the salt and pepper cases, median filtering is the best approach.

4. Compared with other approaches, the average and adaptive filters are fast and stable algorithms for most noise situations.

3.4.5 Figures of noised and denoised boats

The original Boats image has been noised with additive zero-mean Gaussian noise with variance 0.13 (see Table 3) to produce a noisy image at SNR 5.2 (see Table 10). The noisy image has been denoised by the following filters: average, adaptive, BlockSvd, wavelet, spline and spline-wavelet. The results are shown in Figs. 25 and 26 at the SNR indicated in the figures as computed from the SNR improvements given in Table'10. Although SNR improvement is large, the restored image are still far below SNR 20. It is considered that images at SNR 20 or above are quite good as compared to the original.

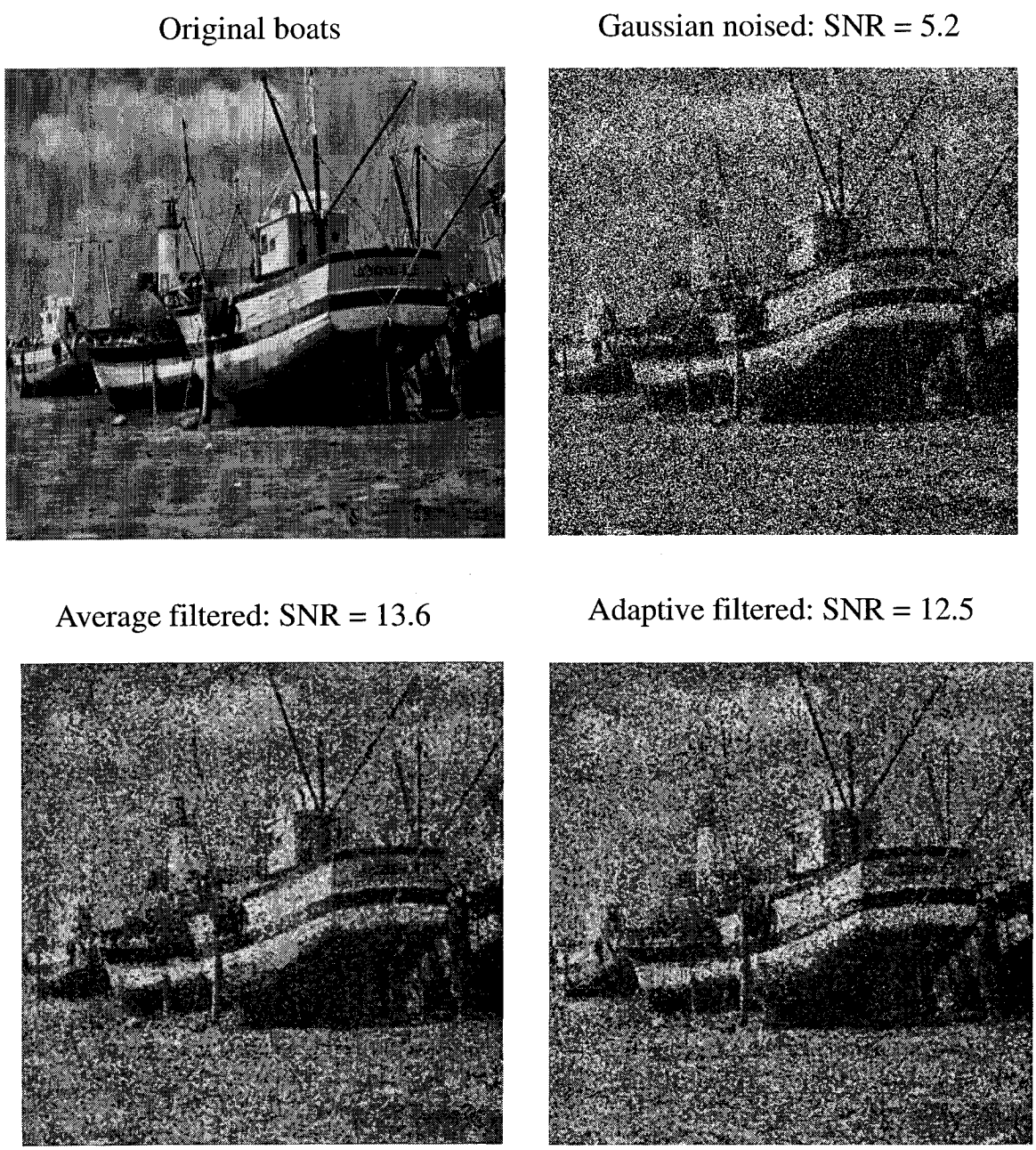
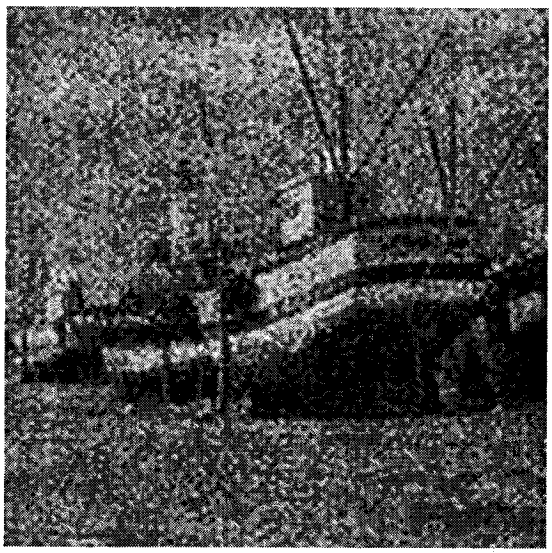


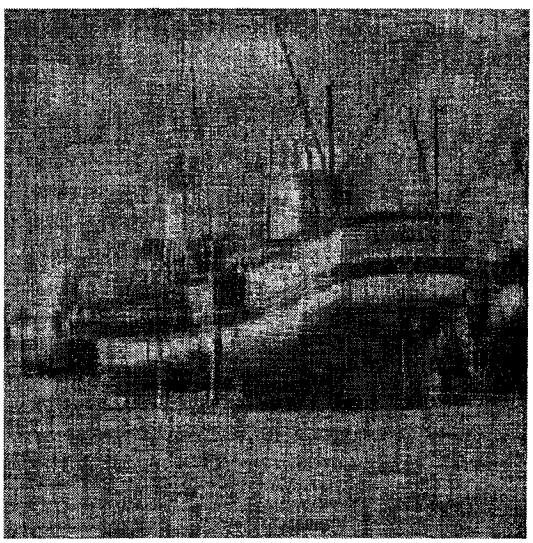
Figure 25: Original, Gaussian noised, average and adaptive filtered Boats image.

BSVD filtered: SNR = 10.0

Wavelet filtered: SNR = 14.9



Spline filtered: SNR = 12.4



Spline-wavelet filtered: SNR = 15.7

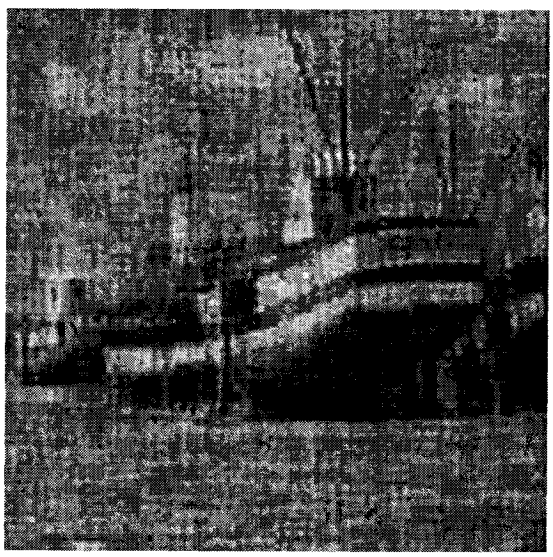


Figure 26: BlockSvd, wavelet, spline and spline-wavelet filtered Boats.

Chapter 4

Conclusion

The first major contribution of this thesis is to propose an improved image denoising method based on singular value decomposition and spline interpolation. Furthermore, we also construct a hybrid method which combines the new spline method with a wavelet denoising method in order to have a more stable and more efficient performance. After many simulation tests, we think the new spline and spline-wavelet algorithms are worthy tools to be used for image denoising. My main purpose for writing this thesis was to introduce two new methods for scientific study, However, it is still difficult to come to any firm conclusions about which approach is the best one, since different approaches behave differently with different images, noise levels and noise types.

Secondly, comparative studies of various image denoising methods are implemented in this thesis, including the new spline and spline-wavelet methods. These methods are briefly reviewed in Chapter 3 and widely tested through numerical experiments on several well-known images. It was found that these methods differ little from each other, and each one has its own character and advantage as listed in Tables 12 and 13.

Finally, I developed a friendly and open-source image denoising demo program, which was used for testing and comparing the performance of each of the image denoising methods. Figures 27 and 28 in Appendix B show the GUI windows for two images. The entire source program contains more than 1300 lines and is listed

Table 12: Speed, code complexity (C.C.) and summary of the different methods.

Method	Speed	C.C.	Summary
Block-Svd	Normal	Easy	1. The performance of the BlockSvd approach is a little weak. 2.BlockSvd performs better with Gaussian noise than with other noises.
Spline	Faster	Normal	1.In most cases, the spline performance is better than BlockSvd and more stable with different noises.
Wavelet	Good	Easy	1. With global thresholding, wavelets are very good for image denoising with good speed and high performance, They can stably work in different noise situations.
Spline-wavelet	Slow	Normal	1. Generally, the spline-wavelet approach is suited for most noise. 2. On the average, it works better than BlockSvd and the Spline method, but this is not an absolute.

Table 13: Speed, code complexity (C.C.) and summary of the different methods. (Continued)

Method	Speed	C.C.	Summary
Median	Fast	Easy	1.Median filtering is a nonlinear operation designed for reducing "salt and pepper" noise. It is more effective than convolution when the goal is to simultaneously reduce noise and preserve edges. So in most of the salt and pepper cases, median filtering is the best approach.
Averaging	Fast	Easy	1. Averaging is another good denoising filter. It is easy to implement and works better in the middle noise level.
Adaptive	Fast	Easy	1. It is a highly recommended image denoising approach with fastest speed, easiest use and best performance in most of the denoise situations.

in Appendix B. Also, its opened architecture provides an easy-to-use and extensible function package.

Chapter 5

Future Work

The proposed new approaches can fix the weighting function problem and improve the denoising performance. Further investigation issues that could be of interest are as follows:

1. Obtain target spline curves that adapt to the level of noise for a better match with the experimental bar graphs shown in Figs. 16, 17 and 18.
2. Use the bargraphs themselves as discrete target curves.
3. Apply the wavelet denoising approach to singular vector with new weight functions.
4. In view of the performance degradation caused by wavelet decomposition levels, it is necessary to find a method that can estimate the degradation state due to different decomposition levels.
5. Develop other simple denoising strategies (possibly using neural networks) that may provide a better performance compared to the ones suggested in this thesis.

Appendix A

Program to draw Figs. 16, 17 and 18

```
% produce drawdiffgau
% use for plotting the curve of differences between the singular values
% of original images and Gaussian noised images
%
% Author; Qi Weibin 2003/09/11

close all;clear;

load boats;
F=boats;

BlockSize=32;
P1=4.2;
P2=0.75;

% ct=6;
% K2=0.41;
% K31=-0.04;
```

```
figure(1);

% Variance range
Var=[0.001,0.005,0.01,0.02,0.04, 0.06,0.08,0.1,0.2,0.3];

Col=1;
for i=1:1:10

Variance=Var(1,i)

% Ordinary Gaussian noise with Mean=0
 Mean=0;

%Step 1: Read the natural image and generate noisy image.

NFkl = F;
XSize = size(NFkl);

% Generate Gaussian and white noisy image.
GFkl = 255*imnoise(NFkl/255.,'gaussian',Mean,Variance);
%GFkl = imnoise(NFkl,'gaussian');

XSize = size(GFkl);

%Step 2:(A) default setting: calculate the paramater ns

NumRows = XSize(1)/BlockSize;

NumCols = XSize(2)/BlockSize;
```

```
t=3;
h=BlockSize-t+1;
ns=0; % initialize ns

% assign memory first
Diffsv=zeros(1,BlockSize); % initialize difference of sing. values
NormDV=zeros(1,BlockSize);
NormSDV=zeros(1,BlockSize);

for i = 1:NumRows   % loop for  i
    r = (i-1) * BlockSize + 1;
    for j = 1:NumCols  % loop for j

        c = (j-1) * BlockSize + 1;

         % Decompose the noised image by svd
         % GSklr singluar values of each noised image block matrix
         % DSklr singluar value of each denoised image block matrix
         % NSklr singluar value of each natural image block matrix
         GFklr = GFkl(r:r+BlockSize-1, c: c+BlockSize-1);
         [GUklr,GSklr,GVklr] = svd(GFklr);

         %(A) calcualte the paramter ns over the last t sing. values
         for z=h:BlockSize
           ns=ns+GSklr(z,z);
         end % end of z loop

         % Decompose the natural image by svd
         NFklr = NFkl(r:r+BlockSize-1, c: c+BlockSize-1);
         [NUklr,NSklr,NVklr] = svd(NFklr);
```

```
        %(B) average difference of singalar values
        for z=1:BlockSize
          Diffsv(z)=Diffsv(z)+GSklr(z,z)-NSklr(z,z);
        end % end of z loop

    end % end of j loop

 end    % end of i loop

% get final ns

ns=ns/(NumRows*NumCols);

p1=P1;
p2=P2;

% get average difference and normalized difference of singular values

maxSV=max(abs(Diffsv));

ShifftDiffsv(1)=0;
ShifftDiffsv(2:32)=Diffsv(2:32);
[maxSDV,topc]=max(abs(ShifftDiffsv));

for z=1:BlockSize-1
    NormSDV(z)=abs(ShifftDiffsv(z))/maxSDV;
%     NormDV(z)=abs(Diffsv(z))/maxSV;
end

subplot(5,2,Col);
x=1:BlockSize;
```

```
stem(x,NormSDV,'filled');
title(['V=' num2str(Variance)]);

Col=Col+1;

end  % for mean
```

Appendix B

Demo and Source Program

B.1 Demo

The demo is illustrated by Lena with Gaussian noise and Barb with salt and pepper noise in Figs. 27 and 28.

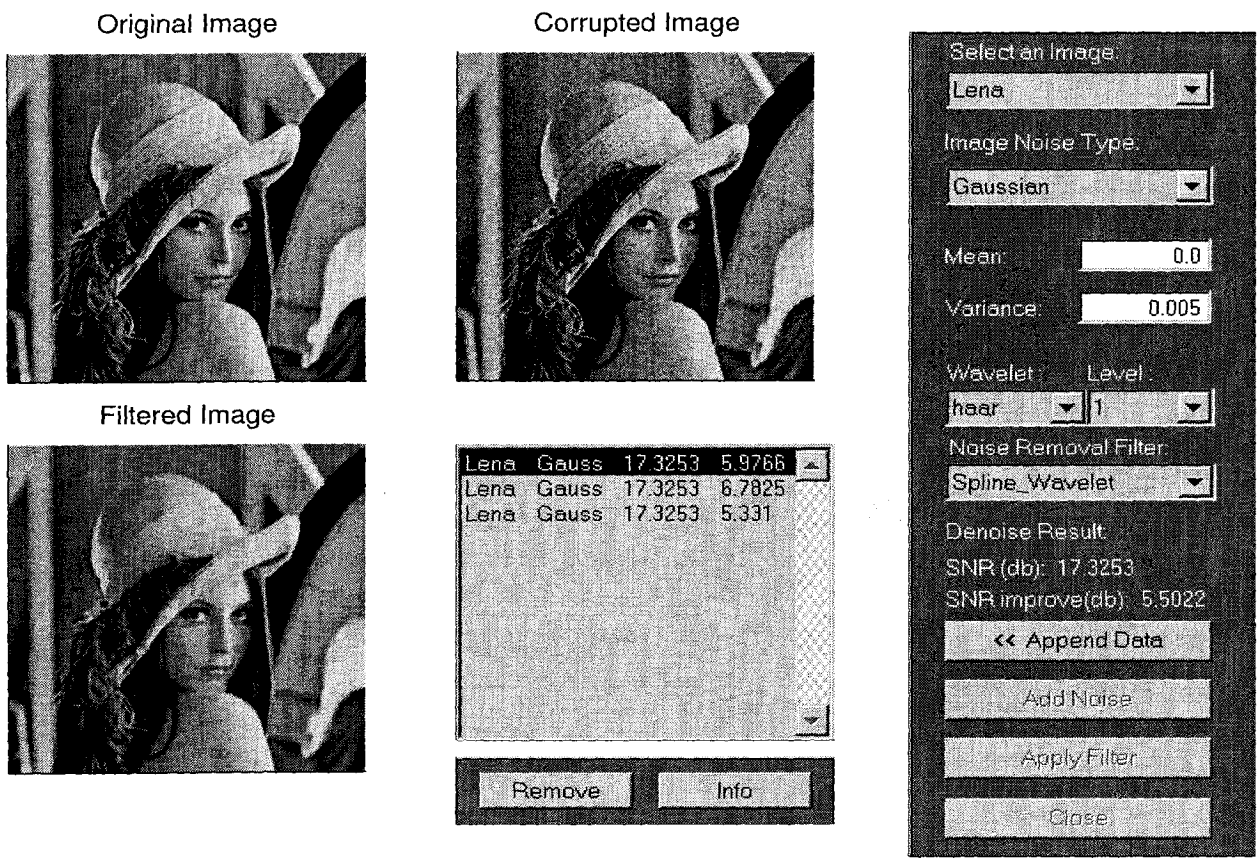


Figure 27: Lena with Gaussian noise.

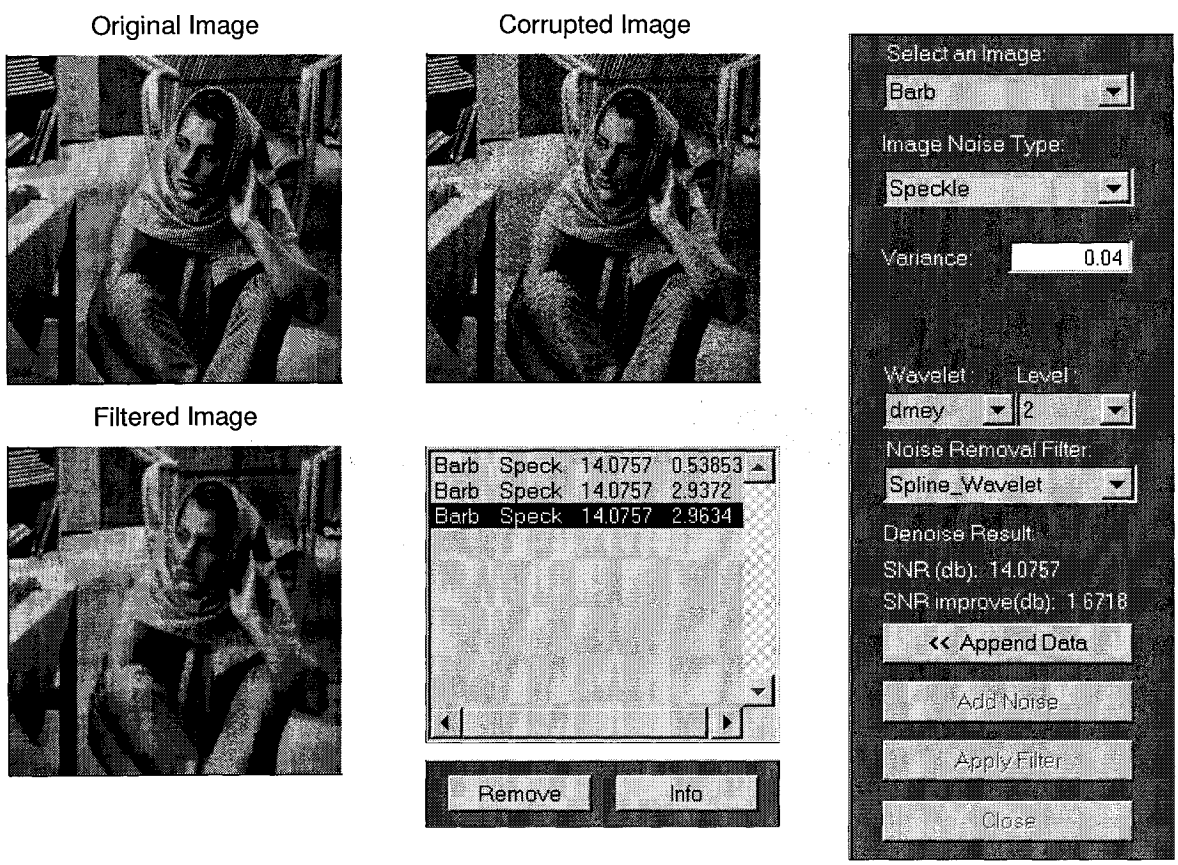


Figure 28: Barb with salt and pepper noise.

B.2 Partial numerical code

```
Function Y=BlockSvdfiltering(GFkl,BlockSize, P1, P2)
%  BlockSvd filtering noise algorithm (second version)
%  Author: Qi weibin  made from 2003/09/11 to 2004/03/10
%  Limiation: this version only fit for square image
%  F: natural image file, (square)
%  BlockSize: 64, 32, 16, 8 (b*b)
%
%
%
%
%  GFklr  noised image block
%  DFklr denoised image block
%  NFklr original nature image block

XSize = size(GFkl);
t=3;
h=BlockSize-t+1;
ns=0;

NumRows = XSize(1)/BlockSize;
NumCols = XSize(2)/BlockSize;

for i = 1:NumRows   % loop for  i
    r = (i-1) * BlockSize + 1;
    for j = 1:NumCols  % loop for j
        c = (j-1) * BlockSize + 1;
        GFklr = GFkl(r:r+BlockSize-1, c: c+BlockSize-1);
        [GUklr,GSklr,GVklr] = svd(GFklr);
        for z=h:BlockSize
```

```
            ns=ns+GSklr(z,z);
        end
    end
end

ns=ns/(NumRows*NumCols);

for i = 1:NumRows
    r = (i-1) * BlockSize + 1;
    for j = 1:NumCols
        c = (j-1) * BlockSize + 1;
        GFklr = GFkl(r:r+BlockSize-1, c: c+BlockSize-1);
        [GUklr,GSklr,GVklr] = svd(GFklr);
        DSklr=GSklr;
        DUklr=GUklr;
        DVklr=GVklr;
        for z = 1:BlockSize  % loop for z
            w(z)=1-(1-(z-1)/(BlockSize/2))^2;
            DSklr(z,z)=GSklr(z,z)-P1*ns*w(z);
            if z<(BlockSize/2)+1
                GUklr(:,z)=fft(GUklr(:,z));
                GVklr(:,z)=fft(GVklr(:,z));
                for h=BlockSize/2+1:BlockSize
                  GUklr(h,z)=P2*GUklr(h,z);
                  GVklr(h,z)=P2*GVklr(h,z);
                end
                DUklr(:,z)=ifft(GUklr(:,z));
                DVklr(:,z)=ifft(GVklr(:,z));
            end
        end
        DFklr = DUklr* DSklr * DVklr';
```

```
        DFkl(r:r+BlockSize-1, c: c+BlockSize-1) = DFklr;
    end

end

Y=abs(DFkl);


Function Y=splinefiltering(GFkl,BlockSize, P1)
%  Spline filtering algorithm
%  Author: Qi Weibin  made in 2003/09/11--2004/03/11
%  Limiation: this version only fit for square image
%  F: natural image file, (square)
%  BlockSize: 64, 32, 16, 8 (b*b)
%
%
%
%  ct is the top point position on the curve
%  k2, k32 are slopes for the second point and thirty-second point
%  GFklr  noised image block
%  DFklr denoised image block
%  NFklr original nature image block

ct=6;
K2=0.31;
K32=-0.04;
XSize = size(GFkl);

NumRows = XSize(1)/BlockSize;
NumCols = XSize(2)/BlockSize;
```

```
t=3;
h=BlockSize-t+1;
ns=0; % initialize ns

for i = 1:NumRows   % loop for  i
    r = (i-1) * BlockSize + 1;
    for j = 1:NumCols
        c = (j-1) * BlockSize + 1;
        GFklr = GFkl(r:r+BlockSize-1, c: c+BlockSize-1);
        [GUklr,GSklr,GVklr] = svd(GFklr);
        for z=h:BlockSize
         ns=ns+GSklr(z,z);
        end
    end
end
ns=ns/(NumRows*NumCols);

% construction w(z) by spline interpolation
A=[ct^3,ct^2,ct,1,-ct^3,-ct^2,-ct,-1;
    3*(ct^2),2*ct,1,0,0,0,0,0;
    0,0,0,1,0,0,0,0;
    6*ct,2,0,0,-6*ct,-2,0,0;
    3*4,2*2,1,0,0,0,0,0;
    0,0,0,0,3*(ct^2),2*ct,1,0;
    0,0,0,0,33^3,33^2,33,1;
    0,0,0,0,3*(32)^2,2*32,1,0];

C=[0;0;0;0;K2;0;0;K32];

d=A\C;
```

```
a3=d(1,1);a2=d(2,1);a1=d(3,1);a0=d(4,1);
b3=d(5,1);b2=d(6,1);b1=d(7,1);b0=d(8,1);

for z = 1:BlockSize
    if z<ct+1
        w(z)=a3*(z^3)+a2*(z^2)+a1*z+a0;
    else
        w(z)=b3*(z^3)+b2*(z^2)+b1*z+b0;
    end
end

for i = 1:NumRows
    r = (i-1) * BlockSize + 1;
    for j = 1:NumCols
         c = (j-1) * BlockSize + 1;
         GFklr = GFkl(r:r+BlockSize-1, c: c+BlockSize-1);
         [GUklr,GSklr,GVklr] = svd(GFklr);
         DSklr=GSklr;
         DUklr=GUklr;
         DVklr=GVklr;
         for z = 1:BlockSize
             DSklr(z,z)=GSklr(z,z)-P1*ns*w(z);
         end
         DFklr = DUklr* DSklr * DVklr';
         DFkl(r:r+BlockSize-1, c: c+BlockSize-1) = DFklr;
    end
end

Y=DFkl;
```

```
Function Y=wavelet_de(X,Bw)
    %    Spline filtering algorithm
    %    X: noised image
    %    Y: denoise image
    %    Use wdencmp for 2D image matrix de-noising.
    %    find default values (see ddencmp).
    XX=X;
    type=Bw.type; level=Bw.level;
    [thr,sorh,keepapp] = ddencmp('den','wv',XX);
    % de-noise image using global thresholding option.
    Y = wdencmp('gbl',XX,type,level,thr,sorh,keepapp);
return
```

B.3 Additional testing data

Additional testing data are listed in Tables 15, 16 and 17 for the Fp2, Fp3 and Barb images with zero-mean Gaussian, salt and pepper and speckle noise to given SNR levels. The testing has been done with the seven denoising methods considered in this work.

Table 14 lists the values of the parameters used in the MATLAB commands described in Table 2 to obtain desired SNR levels in the noised images: Fp2, Fp4 and Barb.

Tables 15, 16 and 17 list the results in detail based on the parameter values listed in Table 14.

Table 14: Parameter values to obtain desired SNR levels in noised images.

Zero-mean Gaussian noise added to Fp2						
Variance	0.003	0.005	0.007	0.01	0.05	0.1
SNR of noised Fp2 (db)	20.1	17.9	16.5	15	8.6	6.3
Zero-mean Gaussian noise added to Fp4						
Variance	0.005	0.01	0.02	0.04	0.1	0.5
SNR of noised Fp4 (db)	21	18.3	15.5	12.8	9.5	5.4
Zero-mean Gaussian noise added to Barb						
Variance	0.002	0.004	0.006	0.01	0.04	0.1
SNR of noised Barb (db)	21	17.7	15.9	13.8	8.3	5.1
Salt and pepper noise added to Fp2						
Density	0.01	0.02	0.03	0.06	0.2	0.4
SNR of noised Fp2 (db)	20.4	17.4	15.6	12.6	7.3	4.3
Salt and pepper noise added to Fp4						
Density	0.02	0.04	0.05	0.1	0.3	0.5
SNR of noised Fp4 (db)	19.2	16.3	15.3	12.2	7.5	5.3
Salt and pepper noise added to Barb						
Density	0.008	0.02	0.04	0.1	0.15	0.3
SNR of noised Barb (db)	20.0	16.0	13.1	9.3	7.3	4.4
Speckle noise added to Fp2						
Variance	0.01	0.02	0.055	0.1	0.2	0.5
SNR of noised Fp2 (db)	20.1	17.0	13.1	10.3	7.5	4.9
Speckle noise added to Fp4						
Variance	0.01	0.025	0.07	0.15	0.3	0.7
SNR of noised Fp4 (db)	20.0	17.0	13.1	10.3	7.5	4.9
Speckle noise added to Barb						
Variance	0.01	0.02	0.05	0.1	0.2	0.45
SNR of noised Barb (db)	19.8	17.0	13.1	10.3	7.5	4.4

Table 15: SNR improvement of the noised Fp2 image with listed SNR level by the seven denoising methods considered in this work.

Zero-mean Gaussian noise added to Fp2						
SNR of noised Fp2 (db)	20.1	17.9	16.5	15	8.6	6.3
Median improvement (db)	3.61	4.59	5.16	5.67	6.58	6.19
Averaging improvement (db)	1.98	3.68	4.7	5.62	8.05	8.14
Adaptive improvement (db)	4.41	5.5	6.05	6.46	7.1	7.16
BlockSvd improvement (db)	1.83	2.79	3.3	3.71	4.64	4.67
Wavelet (dmey) improvement (db)	4.36	4.9	5.2	5.38	9.0	9.5
Spline improvement (db)	2.52	3.42	3.91	4.4	6.17	6.5
Spline-wavelet improvement (db)	3.04	4.19	4.86	5.51	8.26	8.97
Salt and pepper noise added to Fp2						
SNR of noised Fp2 (db)	20.4	17.4	15.6	12.6	7.3	4.3
Median improvement (db)	7.35	10.07	11.73	13.52	13.38	8.49
Averaging improvement (db)	1.67	3.96	5.08	6.54	7.67	7.37
Adaptive improvement (db)	0.4	1.36	1.95	3.15	5.55	6.36
BlockSvd improvement (db)	0.89	1.72	2.12	3.03	4.24	4.37
Wavelet (dmey) improvement (db)	3.1	3.72	4.42	5.67	8.92	8.72
Spline improvement (db)	0.66	1.08	2.36	3.45	5.62	6.13
Spline-wavelet improvement (db)	3.38	4.32	4.91	6.71	8.83	8.75
Speckle noise added to Fp2						
SNR of noised Fp2 (db)	20.1	17.0	13.1	10.3	7.5	4.9
Median improvement (db)	3.1	3.39	4.61	4.85	4.93	4.79
Averaging improvement (db)	2.08	4.25	6.46	7.27	7.47	7.4
Adaptive improvement (db)	4.41	5.53	6.1	6.2	6.2	6.16
BlockSvd improvement (db)	1.79	2.87	3.88	4.25	4.33	4.36
Wavelet (dmey) improvement (db)	4.39	5.07	6.27	7.83	8.5	8.58
Spline improvement (db)	2.53	3.29	4.49	5.15	5.5	5.63
Spline-wavelet improvement (db)	3.3	4.47	6.45	6.82	7.53	7.78

Table 16: SNR improvement of the noised Fp4 image with listed SNR level by the seven denoising methods considered in this work.

Zero-mean Gaussian noise added to Fp4						
SNR of noised Fp4 (db)	21	18.3	15.5	12.8	9.5	5.4
Median improvement (db)	5.4	6.0	6.4	6.6	6.5	4.6
Averaging improvement (db)	5.2	6.6	7.4	7.6	7.4	6.1
Adaptive improvement (db)	6.27	6.56	6.5	6.3	6.1	5.7
BlockSvd improvement (db)	3.32	4.0	4.3	4.23	4.35	4.0
Wavelet (dmey) improvement (db)	5.3	6.7	8.02	8.7	8.5	7.0
Spline improvement (db)	4.6	5.22	5.55	5.65	5.53	4.9
Spline-wavelet improvement (db)	5.89	6.26	7.36	7.89	7.73	6.37
Salt and pepper noise added to Fp4						
SNR of noised Fp4 (db)	19.2	16.3	15.3	12.2	7.5	5.3
Median improvement (db)	13.82	16.06	16.57	17.39	12.53	6.69
Averaging improvement (db)	6.17	7.26	7.49	7.77	7.05	6.19
Adaptive improvement (db)	1.09	1.9	2.24	3.57	5.44	5.54
BlockSvd improvement (db)	1.96	2.4	2.59	3.34	4.22	4.02
Wavelet (dmey) improvement (db)	3.02	3.58	3.53	5.04	8.21	7.39
Spline improvement (db)	0.73	1.78	1.49	2.92	5.26	4.96
Spline-wavelet improvement (db)	3.39	4.66	5.39	8.14	7.71	6.45
Speckle noise added to Fp4						
SNR of noised Fp4 (db)	20.0	17.0	13.1	10.3	7.5	4.9
Median improvement (db)	4.33	4.66	4.95	5.03	5.09	4.2
Averaging improvement (db)	5.6	6.5	6.7	6.3	5.89	5.43
Adaptive improvement (db)	6.19	6.64	6.2	5.7	5.33	5.06
BlockSvd improvement (db)	3.45	3.95	4.09	3.98	3.8	3.6
Wavelet (dmey) improvement (db)	5.4	6.87	7.48	7.1	6.68	6.33
Spline improvement (db)	4.57	4.9	5.04	4.83	4.6	4.3
Spline-wavelet improvement (db)	5.84	6.95	6.35	6.25	5.9	5.45

Table 17: SNR improvement of the noised Barb image with listed SNR level by the seven denoising methods considered in this work.

Zero-mean Gaussian noise added to Barb						
SNR of noised Barb (db)	21	17.7	15.9	13.8	8.3	5.1
Median improvement (db)	-4.9	-2.2	-0.7	1.0	4.5	5.2
Averaging improvement (db)	-4.7	-1.9	-0.3	1.5	5.5	6.9
Adaptive improvement (db)	-1.5	0.8	2.0	3.3	5.6	6.4
BlockSvd improvement (db)	-0.8	1.16	2.1	2.9	4.3	4.6
Wavelet (dmey) improvement (db)	-0.4	-1.5	-0.22	1.2	4.0	4.8
Spline improvement (db)	-0.6	1.46	2.5	3.5	5.9	6.7
Spline-wavelet improvement (db)	-2.9	-0.5	0.8	2.35	6.2	7.88
Salt and pepper noise added to Barb						
SNR of noised Barb (db)	20.0	16.0	13.1	9.3	7.3	4.4
Median improvement (db)	-3.4	-0.2	2.4	6.3	7.8	8.7
Averaging improvement (db)	-3.4	-0.4	1.9	4.9	5.8	6.8
Adaptive improvement (db)	-1.95	-0.3	1.1	3.3	4.2	5.6
BlockSvd improvement (db)	-0.24	1.36	2.4	3.6	3.9	4.3
Wavelet (dmey) improvement (db)	-1.23	1.01	2.3	5.5	6.7	8.5
Spline improvement (db)	0.52	1.97	2.98	4.55	5.4	6.3
Spline-wavelet improvement (db)	-1.5	1.02	2.8	5.6	6.9	8.4
Speckle noise added to Barb						
SNR of noised Barb (db)	19.8	17.0	13.1	10.3	7.5	4.4
Median improvement (db)	-4.1	-1.4	-1.8	-0.9	-0.57	-0.32
Averaging improvement (db)	-3.9	-1.1	-1.7	-0.8	-0.49	-0.23
Adaptive improvement (db)	-0.74	1.49	-0.75	-0.36	-0.22	-0.1
BlockSvd improvement (db)	-0.15	1.68	-0.52	-0.27	-0.16	-0.07
Wavelet (dmey) improvement (db)	-3.0	-0.8	-1.2	-0.64	-0.56	-0.2
Spline improvement (db)	0.05	2.01	0.2	0.02	0.06	0.06
Spline-wavelet improvement (db)	-2.1	0.22	-1.01	-0.45	-0.2	-0.04

Bibliography

[1] I. Daubechies, *Ten Lectures on Wavelets*, SIAM, Philadelphia PA, 1992.

[2] A. Cohen, I. Daubechies, and J.-C. Feauveau, *Biorthogonal bases of compactly supported wavelets*, Commun. on Pure and Appl. Math., **45** (1992) 485–560.

[3] Ž. Devčić, S. Lončarić, *SVD block processing for non-linear image noise filtering*, J. of Computing and Information Technology, **7**(3) (1999) 255–259.

[4] D. L Donoho, *Denoising by soft thresholding*, IEEE Transactions on Information Theory, 41(3):613–627, May 1995.

[5] G. H. Golub and C. F Van Loan, *Matrix Computations*, 3rd ed., The John Hopkins University Press, Baltimore and London, 1996.

[6] C. J. C. Kruger, *Constrained Cubic Spline Interpolation. Applications for Chemical Engineering.*
`http://www.korf.co.uk/spline.pdf`

[7] K. Lees, *Image Denoising Using Wavelets*, page. 5–15. Undergraduate Project Dissertation, Department of Computer Science, University of Sheffield, June 2002.

[8] S. Mallat, *A wavelet tour of signal processing*, 2nd ed., Academic Press, San Diego CA, 1999.

[9] MATLAB, The MathWorks, Inc. Company web site:
`http://www-ccs.ucsd.edu/matlab/techdoc/basics/gs_prefa.html`.

[10] MATLAB, The MathWorks, Inc. Company web site:
`http://www.mathworks.com/access/helpdesk/help/toolbox/wavelet/preface5.shtml`.

[11] MATLAB, The MathWorks, Inc. Company web site:
`http://www.mathworks.com/access/helpdesk/help/toolbox/images/imnoise.shtml`.

[12] MATLAB, The MathWorks, Inc. Company web site:
`http://www.mathworks.com/access/helpdesk/help/toolbox/wavelet/ch06_a44.shtml`.

[13] MATLAB, The MathWorks, Inc. Company web site:
`http://www.mathworks.com/access/helpdesk/help/techdoc/ref/filter2sh.tml`.

[14] Y. Meyer, *Wavelets and Operators*, Translated by D. H. Salinger, Cambridge University Press, Cambridge UK, 1992.

[15] J. Portilla, S. Drexel, M. J. Wainwright and E. P. Simoncelli, *Image Denoising using Gaussian Scale Mixtures in the Wavelet Domain*, pp. 1–2, September 29, 2002.

[16] R. Rangarajan, R Venkataramanan and S Shan, *Image Denoising Using Wavelet (Wavelet and Time Frequency)*, page. 7, December 16, 2002.

[17] S. Rout, *Orthogonal vs. Biorthogonal Wavelets for Image Compression*, M.Sc. thesis, Virginia Polytechnic Institute and State University (Virginia Tech), Blaxksurg VI, August 21, 2003.

[18] G. Strang and T. Nguyen, *Wavelets and Filter Banks*, Wellesley-Cambridge Press, Box 812060, Wellesley MA 02181, 1997.

[19] The MathWorks, Inc. Company web site:
`http://www.mathworks.com/access/helpdesk/help/toolbox/images/wiener2.shtml`.

[20] K. Tsuda and G. Ratsch, *Image Reconstruction by linear programming*, pp. 1–2. `http://books.nips.cc/papers/files/nips16/NIPS2003_AA08.pdf`.