# Advanced Image Analysis
# Wavelet Homework

Emre Ozan Alkan
{emreozanalkan@gmail.com}
MSCV-5

January 16, 2015

## 1 Introduction

In this homework, we investigated the wavelet transform and its applications in image denoising. We developed two functions. They are; j-level wavelet transform of an NxN image and inverse j-level wavelet transform of and NxN array of wavelet coefficients.

## 2 Implementation

In the implementation part, I created two matlab functions and a matlab script for test. Functions are: 'jLevelWaveletTransform' and 'inverseJLevelWaveletTransform', and script called 'RUNME'.

### 2.1 Wavelet Transform

We developed a function to calculate wavelet coefficients. It takes 3 input arguments: an input image, the number of levels J, and low pass filter. It outputs an array of NxN wavelet coefficients.

Listing 1: jLevelWaveletTransform.m

```
function [ waveletCoefficients ] = jLevelWaveletTransform( image, J, lowPassFilter )
%JLEVELWAVELETTRANSFORM J-level wavelet transform
%   A Matlab function for computing the J-level wavelet transform of an NxN image (assume
     N is a power of 2).

if ~isempty(J) && J < 1
    error('J is not valid');
end

[row, col] = size(image);

if row ~= col
    error('Image should be NxN');
elseif mod(row, 2) || mod(col, 2)
    error('assume N is a power of 2');
end

% High Pass Filter
for ii = 1 : length(lowPassFilter)
```

```
    highPassFilter(ii) = lowPassFilter(ii) * power(-1, ii);
end

% Flip the high pass filter
highPassFilter = fliplr(highPassFilter);

% Initialization
waveletCoefficients = zeros(row, row);
temp = zeros(row, row);

for ii = 1 : row
    % Low Pass Filtering
    imageRowLowPass = pconv(lowPassFilter, image(ii, :));
    % Downsampling
    downSampledRowLow = imageRowLowPass(1 : 2 : length(imageRowLowPass));
    % High Pass Filtering
    imageRowHighPass = pconv(highPassFilter, image(ii, :));
    % Downsampling
    downSampleRowHigh = imageRowHighPass(1 : 2 : length(imageRowHighPass));
    % Storing rows for jth level
    temp(ii, :) = [downSampledRowLow, downSampleRowHigh];
end

for ii = 1 : col
    % Low Pass Filtering
    imageColLowPass = pconv(lowPassFilter, temp(:, ii)'); % temp used
    % Downsampling
    downSampledColLow = imageColLowPass(1 : 2 : length(imageColLowPass));
    % High Pass Filtering
    imageColHighPass = pconv(highPassFilter, temp(:, ii)'); % temp used
    % Downsampling
    downSampleColHigh = imageColHighPass(1 : 2 : length(imageColHighPass));
    % Output for jth level
    waveletCoefficients(:, ii) = [downSampledColLow, downSampleColHigh];
end

% Recursive Call for (j-1)th Level
if J > 1
    waveletCoefficients(1 : (row / 2), 1 : (col / 2)) = jLevelWaveletTransform(
        waveletCoefficients(1 : (row / 2), 1 : (col / 2)), J - 1, lowPassFilter);
end

end
```

## 2.2 Inverse Wavelet Transform

We also developed inverse wavelet transformation function that reconstructs images from wavelet coefficients. It takes 3 inputs: array of wavelet coefficients, the number of levels J and low pass filter. It outputs a reconstructed image.

Listing 2: jLevelWaveletTransform.m

```
function [ reconstructedImage ] = inverseJLevelWaveletTransform( waveletCoefficients, J,
    lowPassFilter )
%INVERSEJLEVELWAVELETTRANSFORM Inverse J-level wavelet transform
%    Inverse J-level wavelet transform of an NxN array of wavelet coefficients.

if ~isempty(J) && J < 1
    error('J is not valid');
end

[row, col] = size(waveletCoefficients);

if row ~= col
    error('Image should be NxN');
```

```
elseif mod(row, 2) || mod(col, 2)
    error('assume N is a power of 2');
end

% High Pass Filter
for ii = 1 : length(lowPassFilter)
    highPassFilter(ii) = lowPassFilter(ii) * power(-1, ii);
end

% Flip the high pass filter
highPassFilter = fliplr(highPassFilter);

% Recursive Call for (j-1)th Level
if J > 1
    waveletCoefficients(1 : (row / 2), 1 : (col /2)) = inverseJLevelWaveletTransform(
        waveletCoefficients(1 : (row / 2), 1 : (col /2)), J - 1, lowPassFilter);
end

% Initialization
reconstructedImage = zeros(row, col);
temp = zeros(row, col);

for ii = 1 : col
    % Upsampling for cols
    downSampledLow = waveletCoefficients(1 : (col / 2), ii);
    upSampledLow = zeros(1, 2 * length(downSampledLow));
    upSampledLow(1 : 2 : length(upSampledLow)) = downSampledLow;
    % Upsampling for cols
    downSampledHigh = waveletCoefficients((col / 2) + 1 : col, ii);
    upSampledHigh = zeros(1, 2 * length(downSampledHigh));
    upSampledHigh(1 : 2 : length(upSampledHigh)) = downSampledHigh;
    % Low pass filter
    upSampledLow = pconv(lowPassFilter, fliplr(upSampledLow));
    % High pass filter
    upSampledHigh = pconv(highPassFilter, fliplr(upSampledHigh));
    % Storing constructed cols for jth level
    temp(:, ii) = fliplr(upSampledLow + upSampledHigh);
end

for ii = 1 : row
    % Upsampling for rows
    downSampledLow = temp(ii, 1 : (row / 2));
    upSampledLow = zeros(1, 2 * length(downSampledLow));
    upSampledLow(1 : 2 : length(upSampledLow)) = downSampledLow;
    % Upsampling for rows
    downSampledHigh = temp(ii, (row / 2) + 1 : row);
    upSampledHigh = zeros(1, 2 * length(downSampledHigh));
    upSampledHigh(1 : 2 : length(upSampledHigh)) = downSampledHigh;
    % Low pass filter
    upSampledLow = pconv(lowPassFilter, fliplr(upSampledLow));
    % High pass filter
    upSampledHigh = pconv(highPassFilter, fliplr(upSampledHigh));
    % Stroing reconstructed rows for jth level
    reconstructedImage(ii, :) = fliplr(upSampledLow + upSampledHigh);
end

end
```

# 3 Results

We used famous Lena image for tests. Forward and inverse wavelet transform applied on Lena with Daubechies D4 filter. Also we tested adding noise and using hard and soft thresholds. Results were reasonable for reconstruction. Here you see the results on the figures.


Figure 1: Lena Image

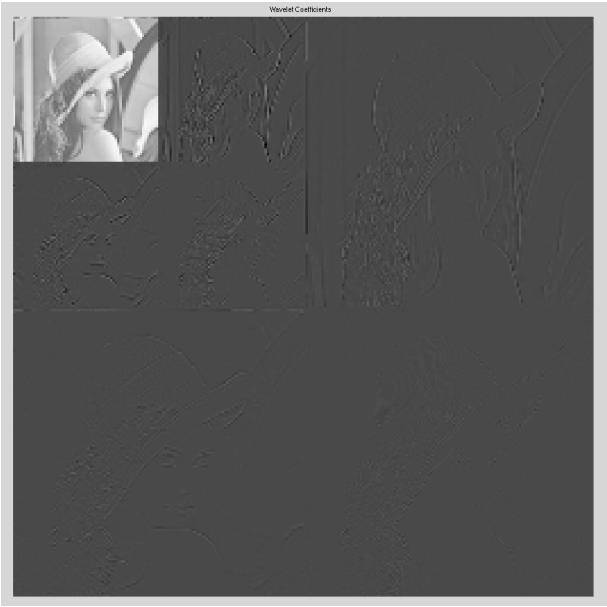


Figure 2: Lena Wavelet Coefficients


Figure 3: Lena Reconstructed from Wavelet Coefficients


Figure 4: Lena Noise Added


Figure 5: Noisy Lena Wavelet Coefficients


(a) Soft Threshold


(b) Hard Threshold

Figure 6: Noisy Lena Image Reconstructed with Threshold