Introduction to Wavelets

Fourier and Wavelet Analysis CTH & GU

October 31, 2006

1 Introduction

The purpose of this assignment is to get familiar with the MatLab toolbox Wavelet containing filter banks, wavelet transforms, and related utilities. You will compute the wavelet transform of some simple signals which should lead to a better understanding of wavelets and filter banks. Also, we want to give a first glimpse of the compression potential of filter banks and wavelet transforms, something that we will exploit further in the assignment on image compression.

1.1 The Wavelet Toolbox

The Wavelet toolbox is a collection of MatLab-files for wavelet analysis, synthesis, and related processing algorithms. To get a first impression of the resources in the toolbox, start the wavelet demo with the MatLab command

```
wavedemo
```

Using the desktop variant of MatLab, you will be able to perform wavelet related operations, and view results, without having to give explicit commands in a command window. This is done by starting the Wavelet Main Menu window.

1.2 The Signals

The signals we will analyze are:

- `pulse` A Dirac impulse at $t = 1/2$.
- `step` A unit step function that jumps from 0 to 1 at $t = 1/2$.
- `triangle` A triangle or hat function.

- `wnoise` A white noise signal
- `sine` A pure harmonic sine wave with frequency 25 Hz.
- `realw` A real-world signal

All these signals are sampled on the unit interval $[0, 1]$ with $512 = 2^9$ samples. The signals can be downloaded in a MatLab session by executing

```
>> load ~bergh/fwa/Lab2/signals.mat
```

In our discussion on filter banks, we have always assumed the signals to be of infinite length. But, in practice, all signals are finite, and this must be taken into account when implementing filter banks. One solution is to make a periodic extension of the signal. This is equivalent to using periodic or circular convolution. In this form, the filter bank is fed with 2^J samples and it then splits the signal into two frequency components with 2^{J-1} samples each. A disadvantage with this approach is that it introduces a discontinuity at the boundaries of the signal if the boundary values do not match each other. This will affect the fine-scale wavelet coefficient close to the boundary, but the situation is not as bad as with the Fourier transform where this phenomenon is global. Another method is to extend the signal symmetrically around the boundaries, which is often used in image compression. It is also possible to modify the filters at the boundary, but this turns out to be rather complicated. We will only use the default symmetric extension mode here.

2 Filter Banks

2.1 Filter Banks in the Wavelet Toolbox

The first thing you have to do is to choose the filters/wavelets you want to work with, *e.g.*, Haar or Daubechies or some others. (The command `waveinfo` produces information about the available wavelets.)

We start with a signal containing 2^9 sample values. We split this signal into a high- and low-frequency part with 2^8 samples each. We then recursively split the low-frequency part and, after L steps, we end up with a low-frequency part containing 2^{9-L} samples. Usually one chooses the number of steps so that 2^{9-L} is greater than the filter length. The output is a vector containing the wavelet coefficients organized as in Figure 1. In other words, the first 2^{9-L} elements of `wc` contains the scaling (approximation) coefficients, then comes the wavelet (detail) coefficients at successively finer scales (higher frequencies).

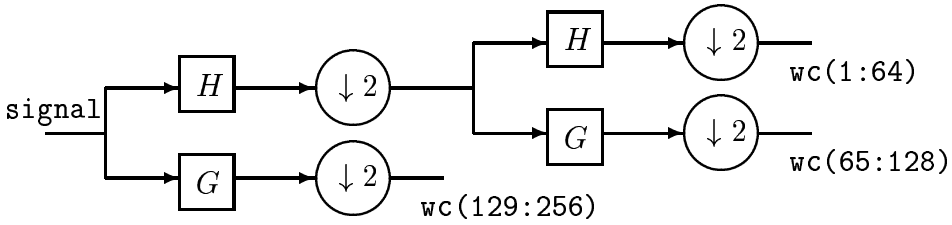

Figure 1: Organization of wavelet coefficients.

2.2 Displaying Wavelet Decompositions

There are several ways to display the wavelet decomposition; one is first to use the `wavedec` command:

```
[wc,l]=wavedec(signal,L,'wname');
```

This gives a vector, of which the first 2^9 entries, `wc`, are the coefficients as described in the previous section. (The `l` part contains information about the lengths of the parts of `wc`.)

2.3 Exercise 1

For the signals `pulse` and `wnoise`, compute the wavelet transform and display the results from `wavedec` followed by `detcoef`. The command

```
wcj=detcoef(wc,l,j)
```

returns the wavelet (detail) coefficients at level j, $1 \leq j \leq L$.

Use the Haar filter and the Daubechies filters with filter length 4 and 6, and put $L = 4$. Plot the results at the different levels. Then answer the following questions:

1. Describe the behaviour of the wavelet coefficients of the Dirac pulse. Is there a difference depending on which filters (wavelets) you are using? If so, explain this difference.

2. Describe the wavelet coefficients of the white noise signal. Is there any pattern in the coefficient values?

3. Using the command `wfilters`, plot the frequency responses of the different filters H and G (coming from the scaling functions and the wavelets) you have used. What is the difference?

3 Wavelet Transforms

In this part, we will compute wavelet decompositions in the light of wavelet theory. In particular, we will see how vanishing moments is connected to the compression ability of the wavelet transform.

3.1 Multiresolution Plots

To display the multiresolution decomposition itself, rather than just the wavelet coefficients, use the `wavedec` command described above.

Then, from these data, the command `waverec` returns the different parts of the decomposed signal:

```
aL=wrcoef('a',wc,l,'wname',L);
```

gives the approximation at level L and

```
dj=wrcoef('d',wc,l,'wname',j);
```

gives the wavelet multiresolution component at level j, $1 \leq j \leq L$.

These components can then be viewed using the standard MatLab plot routines.

3.2 Exercise 2

For each of the signals `step`, `triangle`, `sine`, and `realw`, compute the wavelet transform using the same filters as before. Plot the multiresolution decomposition in each case. Please answer the following questions:

1. Describe the behaviour of the detail signals d_j for the `step` and `triangle` signal. Is there a difference depending on what filters you are using? If so, explain this difference.

2. Do the same thing for the `sine` signal.

3. For the `realw` signal, plot a histogram showing the distribution of the absolute value of the sample values. Here the `hist` command is useful. Do the same thing for the wavelet coefficients. What is the difference? Is there a difference depending on what signal or filter you are using? If so, explain this difference, and its possible use for compression purposes.

4. Finally, we want you to plot the scaling functions and wavelets corresponding to the Daubechies-4 and Daubechies-6 filters. To do this, you use the *cascade algorithm*. To plot a scaling function $\varphi_{j,k}$ in the

interval $[0, 1]$, choose the coarsest scale $L = j,\ j < 9$. Then define a vector

```
wc=zeros(1,2^9);
```

Now set the coefficient in `wc` corresponding to $s_{j,k}$ equal to one. Then compute the inverse wavelet transform of `wc` of the approximation part by

```
signal=wrcoef('a',wc,l,'wname');
```

where l can be copied from the computations in Exercise 1. The elements in `signal` will then be fair approximations to the sample values of $\varphi_{j,k}$. In the same way, setting the coefficient in `wc` corresponding to $w_{j,k}$ to one will give a approximation to $\psi_{j,k}$ using

```
signal=wrcoef('d',wc,l,'wname',j);
```

You will probably have to do some trial-and-error before the plots look nice. For instance, you have to choose j large enough and choose k properly so that the scaling function and wavelet are supported in the unit interval [0,1].

How could these approximations be improved?