

## Wavelets MATLAB Assignment 3

### 3.1: The Folding Lemma

Recall from class that if we have filters  $u_1, v_1 \in \ell^2(\mathbb{Z}_N)$  such that the system matrix  $A_1(n)$  is unitary for all  $n$ , we can obtain  $u_2, v_2 \in \ell^2(\mathbb{Z}_{N/2})$  so that the system matrix  $A_2(n)$  at the next level is also unitary for all  $n$  via the “Folding Lemma”: let

$$u_2(n) = u_1(n) + u_1(n + N/2),$$

and similarly for  $v_2$  in terms of  $v_1$ . This folding operation is easy to do in MATLAB. For example, given a vector

```
>> z = [0 2 3 -2 1 4]
```

we let

```
>> a = z(1:3)
```

and

```
>> b = z(4:6)
```

Then the fold of  $z$  is

```
>> w = a + b.
```

Since we will use this many times, it is worth writing a function file called “fold” which applies the folding operation to a general vector of even length. To do this, write

```
>> n = length(z);
```

```
>> a = z(1:n/2);
```

```
>> b = z(1 + n/2:n);
```

```
>> w = a + b;
```

where  $z$  is the name of your input variable and  $w$  is the name of the output. Write and save such a function file and test it with a few simple examples.

### 3.2: The $p^{\text{th}}$ Stage Wavelet Filters

We want to carry out the  $4^{\text{th}}$  stage wavelet transform on  $\ell^2(\mathbb{Z}_{512})$  with both *the real Shannon wavelets* and *Daubechies’s D6 wavelets*. In each case we will need the filters  $u_1, v_1, u_2, v_2, u_3, v_3, u_4, v_4$  and all their tildes. For example, suppose we start with the real Shannon wavelets. If you saved the first stage filters  $u_1, v_1, \tilde{u}_1, \tilde{v}_1$  for the real Shannon wavelets in a file called “shanwavefilt”, as suggested last time, they can all be recalled by typing

```
>> load shanwavefilt
```

Then we can obtain  $u_2, v_2, u_3, v_3, u_4, v_4$ , which we call “shanu2”, “shanv2”, and so on in MATLAB to identify them, via

```
>> shanu2 = fold(shanu1)
```

```
>> shanv2 = fold(shanv1)
```

```
>> shanu3 = fold(shanu2)
```

```
>> shanv3 = fold(shanv2)
```

etc.

To obtain the tildes of these, one can use the usual method to compute tildes:

```
>> shanu2til = real(iffc(conj(fft(shanu2))));
```

etc., or more simply one can note that for any vector  $z$  of even length, the fold of  $\tilde{z}$  is the same as the tilde of the fold of  $z$ .

Proof: First,  $\tilde{z}(n) = \overline{z(N-n)}$ , so

$$[\text{fold}(\tilde{z})](n) = \tilde{z}(n) + \tilde{z}(n + N/2) = \overline{z(N-n)} + \overline{z(N-(n+N/2))} = \overline{z(N-n)} + \overline{z(N/2-n)}.$$

On the other hand,  $(\text{fold}(z))(n) = z(n) + z(n + N/2)$ , so

$$[\text{fold}(z)]^{\sim}(n) = \overline{(\text{fold}(z))(N-n)} = \overline{z(N-n) + z(N-n+N/2)} = \overline{z(N-n)} + \overline{z(N/2-n)},$$

which agrees with  $\text{fold}(\tilde{z})$ .

Hence  $\tilde{u}_2 = (\text{fold}(u_1))^{\sim} = \text{fold}(\tilde{u}_1)$ , and so on, so one can just use

```
>> shanu2til = fold(shanu1til)
```

```
>> shanv2til = fold(shanv1til)
```

```
>> shanu3til = fold(shanu2til)
```

and so on.

At this point it is worth computing and saving  $u_1, v_1, u_2, v_2, u_3, v_3, u_4,$  and  $v_4$  and all their tildes for the real Shannon wavelets in a file called “shanfil” and similarly for the Daubechies D6 wavelets in a file called “D6fil”. The easiest way to save a large number of items is to make sure that only the vectors you want are in the workspace, and then just type

```
>> save filename
```

where in our case “filename” is “shanfil”; this command saves all current workspace variables in the file “shanfil”. Try saving all the filters and then clearing your workspace, then loading them to make sure you have all the necessary filters.

### 3.3: The $p^{\text{th}}$ Stage Wavelets

Recall that the wavelets  $f_1, f_2, \dots, f_p$  and  $g_p$  are defined inductively by

$$f_1 = v_1, \quad g_1 = u_1, \quad f_\ell = g_{\ell-1} * U^{\ell-1}(v_\ell), \quad \text{and} \quad g_\ell = g_{\ell-1} * U^{\ell-1}(u_\ell).$$

Recall also that

$$\psi_{-j,k} = R_{2^j k} f_j \quad \text{and} \quad \varphi_{-j,k} = R_{2^j k} g_j.$$

Using this, one can produce Figures 22 a-e and 26 a-e (see Exercise 1). For example, to start, try

```
>> f2=realcon(shanu1,up(shanv2));
>> g2=realcon(shanu1,up(shanu2));
>> f3=realcon(g2,up(up(shanv3)));
```

and so on. Note that the filters  $g_j$  are needed for the next stage every time.

There is no particular need to save these wavelets, since they are not directly used later. However, if your versions of these match the ones in the text, you can be confident that your  $p^{\text{th}}$  stage wavelet filters are correct.

### 3.4: Script Files in MATLAB

In section 2.3 of the previous MATLAB assignment, we considered function files, which are MATLAB programs that you store and use to carry out operations that you will repeat often. There is another kind of program file in MATLAB, called a script file, that is also useful. A script file is just a sequence of MATLAB commands. When you call the file, these commands will be executed just as if you typed them into MATLAB at that point. In particular, any input variable used in the script file needs to already be in your workspace with the same name at the time you execute the file, or else you get an error message. Moreover, the variables created by the script file will be part of your workspace after you execute the file. Script files are written in a texteditor and saved as “filename.m”, the same as function files, where “filename” is your choice of name for the script file. They are called via the command

```
>> filename
```

Unlike function files, you do not specify any variable names for the file to operate on; it operates on variables already in the workspace the same as if you type its commands directly. For example, start with

```
>> a=[1 2 3]
>> b=[5 4 3]
>> ! textedit
```

When the texteditor window comes up, type in it the following:

```
c=a+b
```

Comment lines can and should be used the same as for function files, but the first line starting with “function” that is needed in function files is not used for script files. The first line of a script file is usually a comment line starting with %; the comment lines are followed by the actual commands of the program.

Now save this file with some name, say “simple.m”, in your same place that you save your function files (either your “a:” drive, or the “work” folder). Then go back to the matlab prompt (you may have to close the texteditor window) and type

```
>> simple
```

The result should be the vector  $c = [6 \ 6 \ 6]$ . If you check (via `>> who`), you should now have  $c$  as a workspace variable. If you try to run “simple” but you don’t have vectors  $a$  and  $b$  in your workspace, you will get an error message.

The reason that script files are sometimes more useful than function files is that function files do not save the output variables to your workspace. When programming the wavelet transform, we will want to save the output vectors.

### 3.5: The $p^{\text{th}}$ Stage Wavelet Transform

At this point we can compute the wavelet transform of a signal  $z \in \ell^2(\mathbb{Z}_{512})$ . Recall that the  $p^{\text{th}}$  stage wavelet transform of  $z$  consists of the components of the vectors  $x_1, x_2, \dots, x_p, y_p$ , defined by

$$x_1 = D(z * \tilde{v}_1), \quad y_1 = D(z * \tilde{u}_1)$$

and then inductively for  $\ell = 2, 3, \dots, p$  by

$$x_\ell = D(y_{\ell-1} * \tilde{v}_\ell), \quad y_\ell = D(y_{\ell-1} * \tilde{u}_\ell).$$

Recall that

$$x_1(k) = \langle z, \psi_{-1,k} \rangle, \quad x_2(k) = \langle z, \psi_{-2,k} \rangle, \dots, x_p(k) = \langle z, \psi_{-p,k} \rangle, \quad \text{and} \quad y_p(k) = \langle z, \varphi_{-p,k} \rangle.$$

In our case  $p$  will be 4. Let's start with a signal  $z$ , say the one from Figure 23a that we saved and called "testvec":

```
>> clear
>> load testvec
>> z=testvec;
```

Let's write a script file called "shantrans" to compute the real Shannon wavelet transform of  $z$ . Open a texteditor and put in the appropriate comment lines (such as "% This is a script file to compute the 4th level real Shannon wavelet transform of a vector  $z$  of length 512"). Then proceed as suggested by the definitions:

```
load shanfil
x1=down(realcon(z,shanv1til));
y1=down(realcon(z,shanu1til));
x2=down(realcon(y1,shanv2til));
```

and so on, down to  $x_4$  and  $y_4$ . To plot these vectors, recall that  $x_1$  and  $y_1$  are vectors of length 256, and we want to plot their components at the points 0, 2, 4, ... , 510, while  $x_2$  and  $y_2$  have length 128, and should be plotted at 0, 4, 8, ... 508, and so on. I suggest defining (still as part of your script file)

```
n=0:1:511;
n1=0:2:511;
n2=0:4:511;
```

etc. Then  $z$  should be plotted against  $n$ , but  $x_1$  and  $y_1$  should be plotted against  $n_1$ , whereas  $x_2$  and  $y_2$  should be plotted against  $n_2$ , etc.

Save your script file with the filename "shantrans.m".

Now return to the matlab prompt. Check your workspace variables to make sure  $z$  is there. Then just type

```
>> shantrans
>> who
```

You should have the vectors  $x_1, x_2, x_3, x_4, y_1, y_2, y_3, y_4, n, n_1, n_2, n_3$  and  $n_4$  in your workspace. By plotting these, you should be able to reproduce Figure 24.

Now if you want to compute the real Shannon wavelet transform of another vector, you only need to define

```
>> z= that vector
and type
>> shantrans
```

One advantage of using script files is that you can make corrections in the file as you go along or after you test it. With a few modifications you can use your "shantrans.m" file to create a script file for computing the Daubechies D6 wavelet transform. Let's do that and call it "D6trans.m". (You may need to right-click to find the option "File" and the sub-option "Store as New File" in order to store the edited file with the new filename.) With these, you should be able to reproduce Figure 27 (see Exercise 2).

### 3.6: The Inverse Wavelet Transform

Suppose we are given the wavelet transform of a vector  $z$ , in other words the vectors  $x_1, x_2, \dots, x_p, y_p$ , where in our case again,  $p$  will be 4. We want to reconstruct  $z$ . This is done by a sequence of upsampling and convolution steps, as described in the text (see Figure 19), via

$$y_3 = U(y_4) * u_4 + U(x_4) * v_4,$$

$$y_2 = U(y_3) * u_3 + U(x_3) * v_3,$$

and so on, until the final step

$$y_0 = U(y_1) * u_1 + U(x_1) * v_1.$$

Then  $y_0$  should agree with the original vector  $z$ . These steps are easy to do in MATLAB:

```
>> y3=realcon( up(y4), shanu4) + realcon( up(x4), shanv4);
```

and so on, for the real Shannon wavelets, and similiary with the Daubechies filters dauu4 etc., for the Daubechies wavelets. Check the result for our favorite example “testvec” via

```
>> load testvec
```

```
>> z=testvec;
```

```
>> shantrans
```

and then carry out the inverse transform as described above. Check it via

```
>> plot(n,y0)
```

to see if  $y_0$  looks like  $z$ , and more precisely by

```
>> plot(n,z-y0)
```

You should get a factor of  $10^{-14}$  along the vertical axis, indicating that the error is only due to roundoff. Since the inverse transform will be used repeatedly when we do compression, it is worthwhile to write script files called “invshantrans.m” and “invD6trans.m” that carry out the inverse transform in each case.

### 3.7: Compression

Suppose we have any orthonormal basis  $v_0, v_1, \dots, v_{N-1}$  for an  $N$ -dimensional vector space  $V$ . Then for any vector  $z \in V$ , we have the expansion

$$z = \sum_{n=0}^{N-1} \langle z, v_n \rangle v_n.$$

(Recall that this is just a decomposition of  $z$  in the basis  $V$ .) We are interested in approximating  $z$  using a sum of the same type but containing only  $K$  terms, where  $K < N$ . For that, we will find the  $K$  largest values of  $\langle z, v_n \rangle$ . Call  $S$  the set of indices where these largest  $K$  values occur. The best approximation to  $z$  using only  $K$  terms is  $\sum_{n \in S} \langle z, v_n \rangle v_n$ . We are interested in comparing these approximations for different bases, to see which basis gives the most accurate approximation to  $Z$  with  $K$  terms. That would be the basis that best compresses  $z$  at level  $K$  among the bases considered. We will consider the Euclidean basis, the Fourier basis, the real Shannon basis, and the Daubechies D6 basis.

To carry out the compression numerically, we will compute the coefficients in any given basis, then find the  $K$  largest in magnitude, set all the others to 0, and then carry out the inverse transform. This gives the same result as  $\sum_{n \in S} \langle z, v_n \rangle v_n$ , since the other values have been replaced by 0.

To see how to do this in MATLAB, consider the following example.

```
>> z= [-3 2 4 -5 -1 6]
```

Fortunately MATLAB has a very simple command to sort vectors in increasing order:

```
>> sort(z)
```

However, we want to consider absolute values, so try

```
>> h= sort(abs(z))
```

Suppose we want to keep the 4 components of  $z$  with the largest magnitude and zero out the two with smaller magnitude. MATLAB has a command called “find” for identifying the components of a vector where something happens. For example, type

```
>> q=find( abs(z) < h(3) )
```

This will give the indices of the components of  $z$  for which the magnitude is smaller than  $h(3)$ , the third largest magnitude in  $z$ . (MATLAB gives the indices as 2 and 5 in this case, since MATLAB

regards  $z(2)$  as 2 and  $z(5)$  as -1 here.) We would like to zero out the components of  $z$  with these indices. There is a command for this also:

```
>> z(:,q)=0
```

Such a command sets to 0 all components of  $z$  with indices in the vector  $q$ .

Let's try to write a general script file to zero out all but the  $K$  largest values in magnitude of a vector called  $w$  of length  $N$ , where our program will work for any given  $K > 0$ . We are tempted to just write

```
h=sort(abs(w));
q=find(abs(w) < h(N+1-K));
w(:,q)=0
```

The term  $N + 1 - K$  may seem strange, but this is needed because MATLAB indexes its vectors from 1 to  $N$ . So for example if we take  $K = 1$ , meaning we only want to keep one component of  $w$ , we want to throw away all values  $w(j)$  where  $|w(j)| < h(N)$ , since  $h(N)$  is the largest absolute value of any component in  $w$ . The only difficulty with this is the problem of "ties". For example, if  $w$  is a constant vector, then no component will have absolute value less than  $h(N - K)$ , so no values will be zeroed out, no matter what  $K$  is.

To fix this, consider the example with length 9 and with  $K=4$ :

```
>> w= [ 2 3 -4 4 -3 3 3 -2 1]
>> h=sort(abs(w))
>> q1=find(abs(w) < h(6))
```

This gives the indices 1, 8, and 9 where  $|w(n)| < h(6) = 3$ . Then try

```
>> q2=find(abs(w) ==h(6))
```

The double equals sign is necessary to distinguish from the case where a vector is being defined. This gives 2, 5, 6, 7, the indices where  $|w(n)| = h(6) = 3$ . Since there are only 3 components in  $q1$ , we need to find two additional components of  $w$  to zero out, from among the four in  $q2$ . It doesn't matter which ones we take, so just take the first 2 by defining

```
>> q3=q2(1:2)
```

Then let

```
>> q= [ q1 q3]
```

and proceed with

```
>> w(:,q)=0
```

as before.

In general, we have a vector of length  $N$  and we want to zero out  $N - K$  components, and  $\text{length}(q1)$  is always less than or equal to  $N - K$ . So we make up the difference using  $q2$  via

```
>> q3=q2(1:N-K-length(q1))
```

If there are no ties, then  $N - K = \text{length}(q1)$  and so  $q3 = q2(1:0)$ . Then  $q3$  is the empty matrix in MATLAB, and it does no harm, since then  $q = [q1 \ q3]$  just equals  $q1$ , which it should in this case. So altogether we have

```
N=length(w);
h=sort(abs(w));
q1=find(abs(w) < h(N+1-K));
q2=find(abs(w) ==h(N+1-K));
q3=q2(1:N-K-length(q1));
q=[q1 q3];
w(:,q)=0;
```

To see the output (the compressed approximation to the original), type

```
>> w
```

Write a script file (called, say, "keeplarge.m") that does this and try it on a few examples (with and without ties) to make sure it works right. Remember that one needs to define  $w$  and  $K$  prior to calling the file.

With the "keeplarge" script, compression is now relatively easy. For example, to compress a vector  $w$  of length 512 using  $K$  terms in its Euclidean expansion, it is enough to define  $w$  and  $K$  and then use

```
>> keeplarge
>> n=0:1:511
>> plot(n,w)
```

To compress  $w$  in the Fourier basis, first take the DFT

```
>> fft(w);
>> w=fft(w);
```

(This step is necessary because “keeplarge” at the next step must be applied to  $w$ .)

```
>> keeplarge
>> y=real(iff(w));
```

Here  $y$  is the compressed approximation to  $w$ . By taking  $\text{iff}(w)$ , you have zeroed out all but the  $K$  largest DFT coefficients. Here it may not be true (even theoretically) that  $\text{iff}(w)$  is real, since zeroing out coefficients before taking the  $\text{iff}$  may destroy the symmetry property needed for the  $\text{iff}$  to be real. But since  $w$  is real, we know that any imaginary part is part of the error, so we simply do away with it.

For wavelet compression, take the wavelet transform (say the real Shannon transform) of the input vector, which we must name  $z$  because that is the required name of the input vector to the script file “shantrans”, by

```
>> shantrans
```

Then assimilate the wavelet coefficients into one vector by concatenation:

```
>> w=[ x1 x2 x3 x4 y4]
```

Then zero out:

```
>> keeplarge
```

Then to run the inverse transform, one needs to separate the compressed versions of  $x_1, x_2, x_3, x_4$ , and  $y_4$  back out of the new  $w$ :

```
>> x1=w(1:256);
>> x2=w(257:384);
```

etc., up to

```
>> y4=w(481:512);
```

Then run

```
>> invshantrans
>> plot(n,y0)
```

since  $y0$  is the signal reconstructed from the compressed wavelet coefficients. You might want to write script files titled “shancompr.m” and “D6compr.m” to carry out these procedures automatically.

### Assignment 3

Reproduce the following figures from the text. With each figure, please print out the MatLab code you used in creating that figure. In all cases except Figure 26c, the limits on the horizontal axis are 0 to 511.

1.) Figures 22b, 22d, 26c, and 26e. The limits on the vertical axis in Figure 22b are -.6 and .6, in Figure 22d they are -.3 and .3. In Figure 26c they are -.6 and .6.

2. Figures 24c, 24e, 27d, and 27f. In Figure 24c the limits on the vertical axis are -1.2 and 1.2.

3. Figures 31b, 31c, 31d, and 31e. In all these figures, the limits on the horizontal axis are -1.2 and 1.2.

4. Figures 33b, 33c, 33d, and 33e. (To obtain the vector used in Figure 33, use  $\exp(v)$  to obtain the term by term exponential of the vector  $v$ .) In all these figures, the limits on the vertical axes are -12 and 12.