CS 6810/7810: Wavelets and Wavelet Algorithms
Assignment 5
Multiresolution Analysis with HWT and CDF(2, 4)

Vladimir Kulyukin
Department of Computer Science
Utah State University

February 18, 2017

Learning Objectives

1. CDF(2, 4)
2. Multiresolution Analysis with HWT and CDF(2, 4)

Introduction

In this assignment, you will implement forward and inverse CDF(2, 4) and then compare it with HWT in the multiresolution analysis of a synthetic signal.

Problem 1

Implement a class $CDF24.java$ with the following two static methods:

```
public static void ordDWTForNumIters(double[] signal, int num_iters) {}
public static void ordInvDWTForNumIters(double[] signal, int num_iters) {}
```

The first method computes the `CDF(2, 4)` transform of the signal for a given number of iterations. The second method inverses the `CDF(2, 4)` transform of the signal for a given number of iterations. Both methods destructively modify their first arguments. Below are a few test methods implementing two examples from lecture 5.

```
public class WaveletAlgos_S17_HW05 {

    static void displaySignal(double[] sig){
        for(double d: sig)
            System.out.println(d);
        System.out.println();
    }
    // example on slides 30-31 in lecture 5
    static void test01() {
        double[] signal = {1, 2, 3, 4};
        CDF24.ordDWTForNumIters(signal, 1);
        displaySignal(signal);
        CDF24.ordInvDWTForNumIters(signal, 1);
        displaySignal(signal);
    }
    // example on slides 45-48 in lecture 5: 1 iteration
    static void test02() {
        double[] signal = {1, 2, 3, 4, 5, 6, 7, 8};
        CDF24.ordDWTForNumIters(signal, 1);
        displaySignal(signal);
        CDF24.ordInvDWTForNumIters(signal, 1);
        displaySignal(signal);
    }
    // example on slides 45-48 in lecture 5: 2 iterations
```

```
    static void test03() {
        double[] signal = {1, 2, 3, 4, 5, 6, 7, 8};
        CDF24.ordDWTForNumIters(signal, 2);
        displaySignal(signal);
        CDF24.ordInvDWTForNumIters(signal, 2);
        displaySignal(signal);
    }
}
```

Here is the output of `test01()` without rounding.

```
public static void main(String[] args) { test01(); }

2.3107890345411484
4.760278777324325
-4.440892098500626E-16
-1.414213562373095

0.9999999999999999
1.9999999999999996
2.9999999999999987
3.9999999999999987
```

Here is the output of `test02()` without rounding.

```
public static void main(String[] args) { test02(); }

2.3107890345411484
5.139216159287337
7.967643284033528
10.038195644853694
-4.440892098500626E-16
-8.881784197001252E-16
-8.881784197001252E-16
-2.82842712474619

1.0000000000000004
1.9999999999999996
2.9999999999999987
3.9999999999999987
4.999999999999998
5.999999999999999
6.999999999999998
7.999999999999997
```

Here is the output of `test03()` without rounding.

```
public static void main(String[] args) { test03(); }

5.901923788646682
12.098076211353314
0.3660254037844384
-3.8301270189221923
-4.440892098500626E-16
-8.881784197001252E-16
-8.881784197001252E-16
-2.82842712474619

1.0
1.9999999999999998
2.999999999999998
3.999999999999997
4.9999999999999964
5.999999999999997
6.999999999999997
7.999999999999998
```

Problem 2

Consider the curve given in equation (1). The graph of this curve on $[0, 511]$ is plotted in the left image of Figure 1.

$$f(x) = \begin{cases} sin(4\pi x) & \text{if } 0 \leq x < \frac{1}{4} \\ 1 + sin(4\pi x) & \text{if } \frac{1}{4} \leq x < \frac{3}{4} \\ sin(4\pi x) & \text{if } \frac{3}{4} \leq x \leq 1 \end{cases} \tag{1}$$

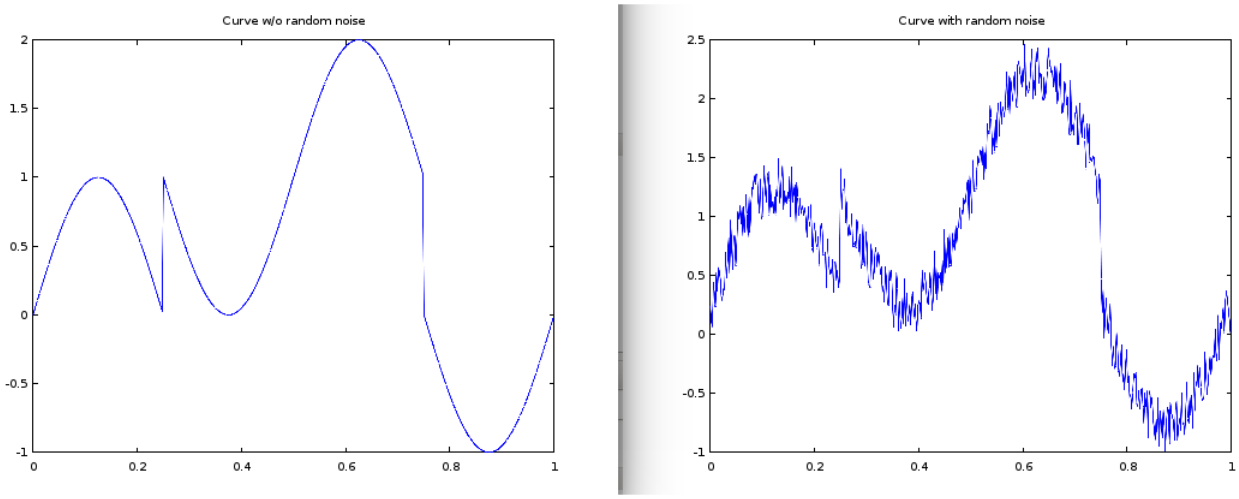

Figure 1: Curve w/o noise (left); curve with random noise added

I used the following JAVA code to generate the values for the graph in Figure 1.

```
public class Function {

    public Function() {}
    public double v(double x) { return 0; }
    public double[] generateRangeInterval(double[] domain_values) {
        return null;
    }

}

public class Ripples_F_ex_4_4_p34 extends Function {

    public Ripples_F_ex_4_4_p34() {}
    @Override
    public double v(double t) {
        if ( 0 <= t && t < 0.25 ) {
            return Math.sin(4*Math.PI*t);
        }
        else if ( 0.25 <= t && t < 0.75 ) {
            return 1 + Math.sin(4*Math.PI*t);
        }
        else if ( 0.75 <= t && t <= 1 ) {
            return Math.sin(4*Math.PI*t);
        }
        else {
            throw new IllegalArgumentException("t == " + t);
        }
    }
}
```

Let us infuse some small random noise into this curve. The noisy curve is shown in the right image of Figure 1. I used this method to add noise.

```
public static void addRandomNoiseToSignal(double[] signal) {
    for(int i = 0; i < signal.length; i++) {
        signal[i] += Math.random()/2.0;
    }
}
```

Use your implementation of `HWT` and `CDF(2, 4)` to do the multiresolution analysis of $f(x)$ with random noise in terms of $D8$, $D7$, $D6$, and $S6$. Figure 2 shows the $D6$ components of `HWT` and `CDF(2, 4)`. Figure 3 shows the $S6$ components of both transforms.

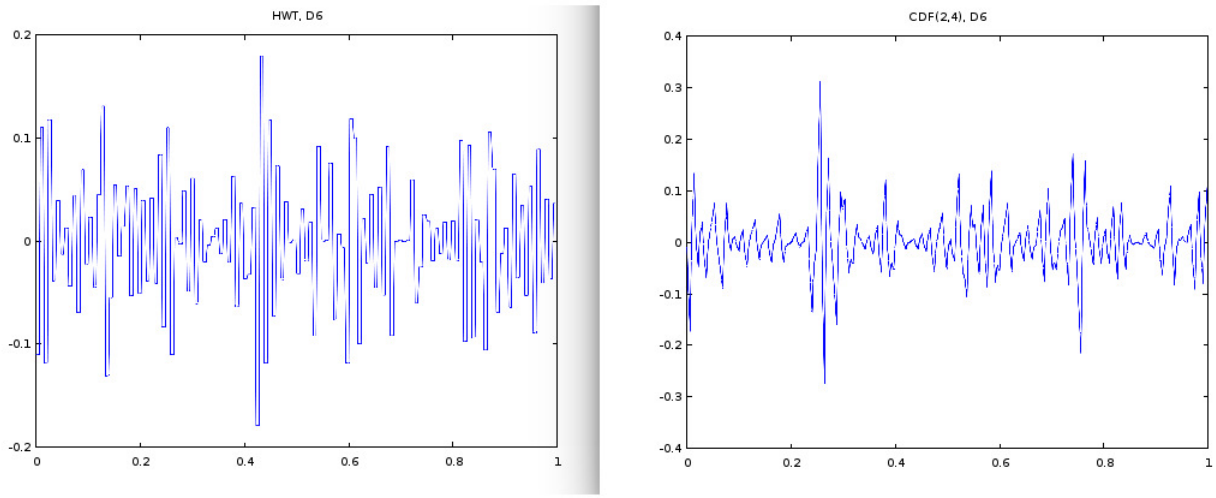

Figure 2: D6 HWT (left); D6 CDF(2, 4) (right)

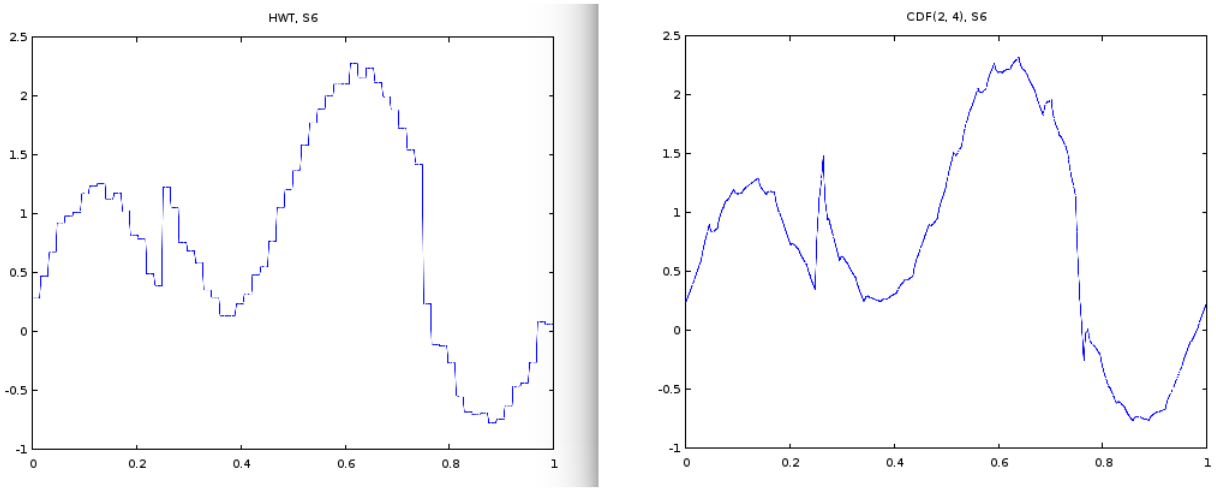

Figure 3: S6 HWT (left); S6 CDF(2, 4) (right)

What To Submit

Submit your `CDF24.java` or `CDF24.py` via Canvas. Also, submit the png images of your plots of $S6$, $D6$, $D7$, and $D8$ components of the multiresolution analysis for both transforms. You do not have to submit your JAVA/Python source for multiresolution analysis. The plots are sufficient. I used Octave to generate my plots. If you do not feel comfortable with Octave, you can use your favorite plotting software (Matlab, matplotlib, R, etc.) to generate your graphs and save them as pngs. Your graphs, of course, may be slightly different due to the random nature of the added noise. Name your

graphs as `S6_HWT.png`, `S6_CDF.png`, `D6_HWT.png`, `D6_CDF.png`, etc.

Happy Hacking and Plotting!