

Wavelets for Agriculture and Biology: A Tutorial with Applications and Outlook

Xuejun Dong, Paul Nyren, Bob Patton, Anne Nyren
Central Grasslands Research Extension Center, North Dakota State
University
4824 48th AVE SE, Streeter, ND 58483

Jim Richardson
Department of Soil Sciences, North Dakota State University
Fargo, ND 58105

Thomas Maresca
824 Regina Street, Philadelphia, PA 19116

April 8, 2008

Checklist of files:

1. “NDHaar.exe” is a stand-alone program for Haar-like discrete wavelet transform;
2. “Suppl_dong.pdf” is the documentation of NDHaar;
3. “pdsi.1.txt” is the Palmer Drought Severity Index for the Northwest of North Dakota, which is the testing data set;
4. “out_1.txt” and “out_2.txt” are two output files from “NDHaar.exe”.

A computer program to do Haar-like discrete wavelet transform. We developed a **True Basic** program, **NDHaar**, to implement the simplified interpretation of the discrete wavelet transform (DWT) procedure as discussed in the main text of our paper (*BioScience*, 58(5), doi:10.1641/B580512). Instead of using the eight-element artificial signal, we used the drought severity signals of the Northwest climate region of North Dakota, for which the Coilet4 wavelet was used to do the same DWT with **TimeStat**, as shown in the main text.

Why this new program? Because many biologists have some familiarity with the **BASIC** program language, casting the DWT procedure in **BASIC** provides an opportunity for many biologists to deeply understand the wavelet technique, as well as to appreciate its beauty and simplicity.

Background information. As shown in the main text (*BioScience* Vol. 58, No. 5), our purpose is to provide a simplified interpretation of the main calculation steps of DWT with elementary mathematics. In the first part of our main text (i.e., the section using an artificial signal to illustrate wavelet MRA), we use a slightly different way of presenting DWT from that used by Walker (1999). While we adopted many basic terminologies from Walker (1999), in our usage, however, the wavelets and scaling functions are not explicitly stretched in order to “measure” the signal variability (or trend) at coarser resolutions. Instead, we keep the wavelet (or scaling function) unchanged but, when necessary, we may drop some zeros from the wavelet (or scaling function) so that its length always equals the length of the signal (or sub-signal) with which it has a scalar product. This avoids defining the higher level wavelets (or scaling functions) in stretched form. As one can see clearly in box 1 of the main text, the DWT is essentially a hierarchical process, in which the successive decompositions are conducted on different trend sub-signals and leaving the fluctuation sub-signals unaltered. As the length of the trend sub-signal shrinks by the power of 2 (compared with the length of the signal from which it is derived), we can always “measure” the variability of this newly generated trend sub-signal using the same wavelet but with the tail shortened in order to match the length of the newly generated sub-signal. As evidenced in the essential list of the program, the computation implementation of NDHaar is quite simple, including three subroutines for calculating the decomposition, reconstruction started from a fluctuation sub-signal, and reconstruction started from a trend sub-signal, respectively.

Some testing results of NDHaar. The results on selected PDSI data are shown in figure 1. We can see that these figures are very similar to those generated using **TimeStat** software. However, big differences occur at coarser resolutions (larger scales), where the simple Haar-like wavelet gives a poor representation of the transitions of the signal values. Despite their differences, both the use of Coiflet4 wavelet through **TimeStat** and the modified Haar wavelet through **NDHaar** provide a perfect reconstruction of the original signal. This is to verify that our DWT implementation based on our simplified interpretation and representation was successful. As such, we can avoid using some terms of DSP and advanced mathematics in our preliminary use of DWT, at least for the simple Haar-like wavelets.

More about NDHaar. The program was written with **True Basic** silver edition (www.truebasic.com) and can be run on a PC with the Windows operating system without True Basic installed. The program can accept one dimensional data series of only one of the following lengths: 64, 128, 256, 512, 1024, 2024, 4096. The input data points must be one of these 7 types. The user can choose to use either the standard Haar wavelet, or a modified Haar wavelet to conduct discrete wavelet transform and multiresolution analysis, as discussed in detail in the main text. However, at this stage the out results are text data files only. One must use a graphics program to visualize the results. In this Supplementary material, as well as in our main text, we used **MINITAB** to draw graphs. While the program itself is to be included with this paper as an executable file, some of the major calculation procedures are listed in this supplementary text, which only considers the case of 1024 data points and only for using the modified Haar wavelet (see the main text for more explanation). The most important parts of this simple program are three

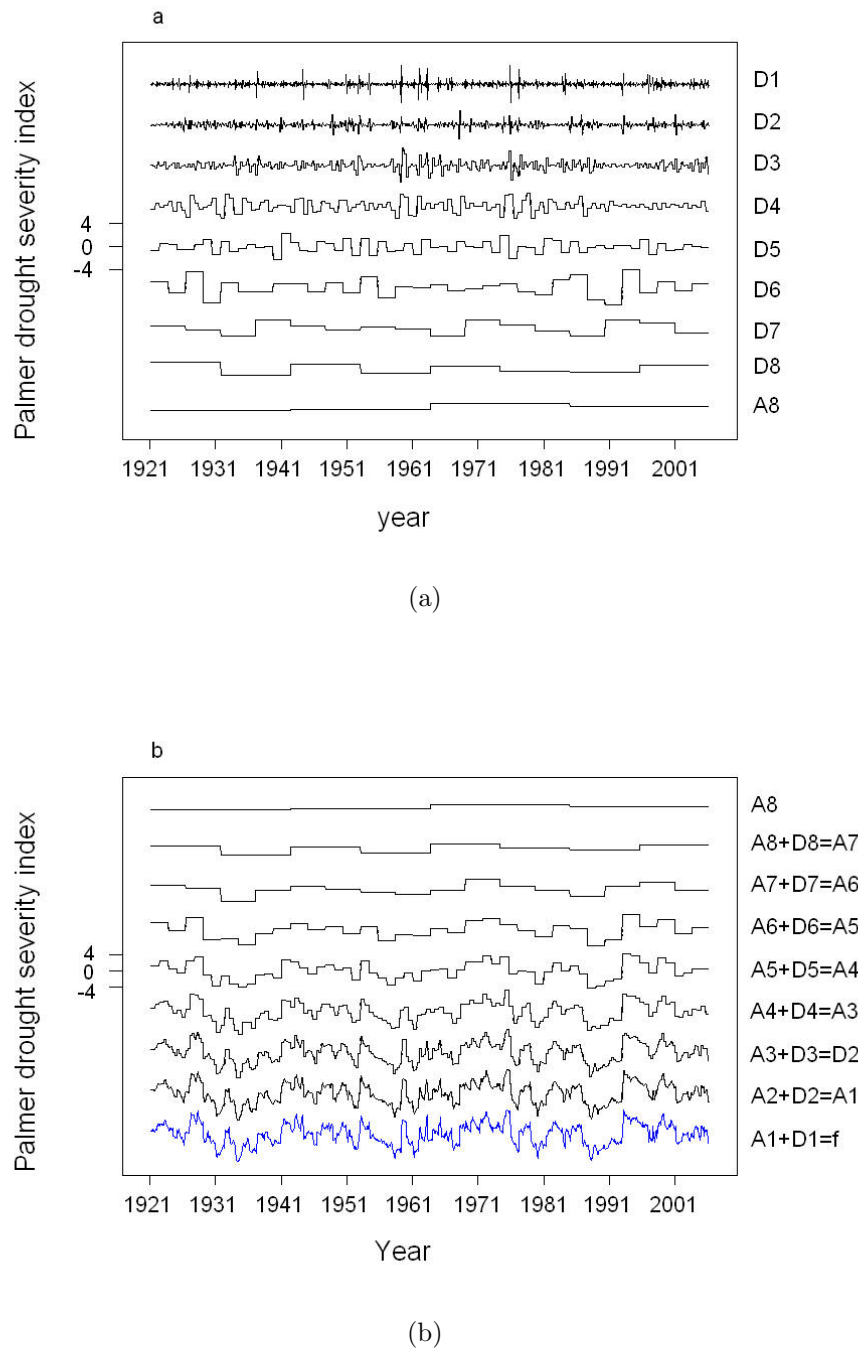


Figure 1: A decomposition and lossless reconstruction of the Palmer Drought Severity Index for the northwest climate region of North Dakota (Jun 1921- Sep 2006). A computer program NDHaar was used with the modified Haar wavelet for the discrete wavelet transform (see text for detail). (a) The original signal was subjected to eight level decompositions, resulting in the first detail signal D1, second detail signal D2,..., eighth detail D8 and averaged A8 signals. (b) The component signals were synthesized successively to obtain a perfect reconstruction of the original signal f (series in blue).

external subroutines: `decom (mother(), sonA(), sonD()), fromD(up1(), D()),` and `fromA(up1(), A())`. Anyone who has some familiarity with Fortran-like structured programming languages should understand the program easily.

Decomposition. The process of the decomposition of a signal using the conventional discrete wavelet transform can be easily understood by realizing that the process goes iteratively, that is, the same rule applies to all levels, and decompositions occur only at the original or the trend sub-signals. This rule is expressed in equations 1 and 2 in the main text. In *NDHaar*, the subroutine `decom` accepts the input from a signal (either the original signal or one of the trend sub-signals) and decomposes it into a trend sub-signal (`sonA()`) and a fluctuation sub-signal (`sonD()`), the length of either of which is half the length of the “mother” signal. This is the common rule used for all the decomposition steps of *NDHaar*.

Reconstruction. Why reconstruct? Because the decomposed sub-signals (as in box 1 of main text) are only intermediate coefficients, they have to be re-scaled to the whole signal level in order to see their “contribution” to the original signal. In *NDHaar*, the two subroutines `fromD` and `fromA` are both needed for a successful reconstruction. Actually, any single reconstruction step in our program uses one of these two subroutines (`fromD` and `fromA`). The subroutine `fromD` conducts a one-step reconstruction starting from a fluctuation sub-signal by assuming the elements of other sub-signals at this level or lower are set to zero. The results of the reconstruction are stored in the array `up1()`, meaning the trend sub-signal located at the immediate upper level of this current starting fluctuation sub-signal. By the same token, subroutine `fromA` conducts a reconstruction from a trend sub-signal and put the final results on the immediate trend sub-signal above it (see box 1 of the main text for detail).

How to use NDHaar. (1) For those who use **True Basic**, simply copy the program list to the **True Basic** source editor, do some minimum editing to correct possible problems caused by line breaks, etc., and run the program. Be sure that only 1024 data points are input in this sample program. (2). For all users, to download the stand-alone program, *NDHaar.exe*, put it in a folder where the user’s data is stored (the length of the data series must be one of the 7 types mentioned above). The user’s data files should be named as `???.txt` and stored as a one column text file with no heading. The user will also need to provide two output file names during the program run, so that *NDHaar* will store the results of DWT on these files and save them in the same folder where the program as well as the original data series are located.

The user is not required to study manuals (as usually the case in using many computer packages), and with a few mouse clicks and answers for a few questions following the computer prompts, the desired results will be generated. The amount of output data depends on the length of the original data series. For example, if one has only 64 data points, then the output file contains only 5 columns of data; on the other hand, if one has 1024 data points, one would expect to have 9 columns of data output, representing the nine decomposed frequency bands, similar those as discussed in the main text of this paper.

A partial list of NDHaar, which can also be used directly for True Basic users.

```
! True Basic program for discrete wavelet transform (DWT)
! using a modified Haar wavelet.
!
! The purpose of this program is just to show an alternative way of
! generating
! a DWT without using an extensive DSP (digital signal processing) terminology.
! The program is designed for
! easy understanding of DWT as detailed in the main text.
! We use a "modified" Haar wavelet throughout,
! that is, the wavelet is defined as (1/2, -1/2, 0,0,0,0,0,0) and the
! scaling function is defined as (1/2, 1/2, 0,0,0,0,0,0) for an
! eight-element signal, both of which can be shifted to occupy other
! locations of the signal.

! by Xuejun Dong
! North Dakota State University
! Central Grassland Research Extension Center
! Streeter, ND 58483
! USA
! E-mail: xuejun.dong@ndsu.edu

! April 8, 2008

! What does this program do? It calculates the nine frequency
! bands,(after eight times of decomposition) of an original one
! dimensional signal of 1024 data points.

! What information is needed from the user? (1) The full name of the
! text file storing the original data (signal); (2) The names for two
! output files (user provided arbitrary names).

! Description of variables
!
! f(), a True Basic array of size 1024, stores the original signal.

! a1(), d1(), a2(),d2(),..., d8(),d8(), are pairs of the first to
! eighth trend and fluctuation sub-signals, respectively.

! sa1(), sa2(), ..., sa7() are seven scratch variables for storing
! intermediate
! results of updated a1(), a2(),..., a7(), so that the original array
! variables
```

```

! of a1(), a2(),..., a7() are not erased when the program runs.

! kd1, kd2(),..., kd8, ka8(), are nine array variables for storing the
! first detail signal, the second detail signal,..., eighth detail
! signal
! and the eighth averaged signal, respectively.

DIM f(1024), a1(512), d1(512), a2(256), d2(256), a3(128), d3(128)
DIM a4(64), d4(64), a5(32), d5(32), a6(16), d6(16), a7(8), d7(8)
DIM a8(4), d8(4)
DIM sa1(512), sa2(256), sa3(128), sa4(64), sa5(32), sa6(16), sa7(8)
DIM kd1(1024), kd2(1024), kd3(1024), kd4(1024), kd5(1024)
DIM kd6(1024), kd7(1024), kd8(1024), ka8(1024)

! Need one column of data (text file) with 1024 rows, no heading

INPUT prompt "Enter the file containing the original signal:": fni$
OPEN #3: name fni$, access input, org text

! There are two output files:
! The first output file contains 5 columns of data
! representing D1, D2, D3, D4, D5.

! The second output file contains 4 columns of data
! representing D6, D7, D8, A8.

INPUT prompt "Enter the first output file name:": fno1$
OPEN #2: name fno1$, create newold, access output, org text

INPUT prompt "Enter the second output file name:": fno2$
OPEN #1: name fno2$, create newold, access output, org text

FOR i = 1 to 1024
    INPUT #3: f(i)
NEXT i

! First do the hierarchical decomposition
! of the original signal f:

CALL decomp (f(), a1(), d1())
CALL decomp (a1(), a2(), d2())
CALL decomp (a2(), a3(), d3())
CALL decomp (a3(), a4(), d4())
CALL decomp (a4(), a5(), d5())
CALL decomp (a5(), a6(), d6())

```

```
CALL decom (a6(), a7(), d7())
CALL decom (a7(), a8(), d8())
```

! Construct the 9 frequency bands kd1(), kd2(), ..., kd8(), ka8():

```
CALL fromD (kd1(), d1())
```

```
CALL fromD (sa1(), d2())
CALL fromA (kd2(), sa1())
```

```
CALL fromD (sa2(), d3())
CALL fromA (sa1(), sa2())
CALL fromA (kd3(), sa1())
```

```
CALL fromD (sa3(), d4())
CALL fromA (sa2(), sa3())
CALL fromA (sa1(), sa2())
CALL fromA (kd4(), sa1())
```

```
CALL fromD (sa4(), d5())
CALL fromA (sa3(), sa4())
CALL fromA (sa2(), sa3())
CALL fromA (sa1(), sa2())
CALL fromA (kd5(), sa1())
```

```
CALL fromD (sa5(), d6())
CALL fromA (sa4(), sa5())
CALL fromA (sa3(), sa4())
CALL fromA (sa2(), sa3())
CALL fromA (sa1(), sa2())
CALL fromA (kd6(), sa1())
```

```
CALL fromD (sa6(), d7())
CALL fromA (sa5(), sa6())
CALL fromA (sa4(), sa5())
CALL fromA (sa3(), sa4())
CALL fromA (sa2(), sa3())
CALL fromA (sa1(), sa2())
CALL fromA (kd7(), sa1())
```

```
CALL fromD (sa7(), d8())
CALL fromA (sa6(), sa7())
CALL fromA (sa5(), sa6())
CALL fromA (sa4(), sa5())
CALL fromA (sa3(), sa4())
CALL fromA (sa2(), sa3())
```

```
CALL fromA (sa1(), sa2())
CALL fromA (kd8(), sa1())

CALL fromA (sa7(), a8())
CALL fromA (sa6(), sa7())
CALL fromA (sa5(), sa6())
CALL fromA (sa4(), sa5())
CALL fromA (sa3(), sa4())
CALL fromA (sa2(), sa3())
CALL fromA (sa1(), sa2())
CALL fromA (ka8(), sa1())

PRINT #2: "d1","d2","d3","d4","d5"
FOR i = 1 to 1024
    PRINT #2: kd1(i), kd2(i), kd3(i), kd4(i), kd5(i)
NEXT i

PRINT #1: "d6","d7","d8","a8"
FOR i = 1 to 1024
    PRINT #1: kd6(i), kd7(i), kd8(i), ka8(i)
NEXT i

CLOSE #1
CLOSE #2
CLOSE #3

END

! Subroutines start here.

SUB decom (mother(), sonA(), sonD())
    LET length = ubound(sonA)
    FOR i = 1 to length
        LET j = 2 * i - 1
        LET k = 2 * i
        LET sonA(i) = (mother(j)+mother(k))/2
        LET sonD(i) = (mother(j)-mother(k))/2
    NEXT i
END SUB

SUB fromD (up1(), D())
    LET ld = ubound(D)
    FOR i = 1 to ld
        LET j = 2 * i - 1
        LET k = 2 * i
        LET up1(j) = D(i)
```



```
        LET up1(k) = -D(i)
    NEXT i
END SUB

SUB fromA (upp(), A())
    LET la = ubound(A)
    FOR i = 1 to la
        LET j = 2 * i - 1
        LET k = 2 * i
        LET upp(j) = A(i)
        LET upp(k) = A(i)
    NEXT i
END SUB
```

References

Walker, J. S.: 1999, *A Primer on Wavelets and Their Scientific Applications*, Studies in Advanced Mathematics, 1 edn, CRC Press LLC, 2000 N. W. Corporate Blvd., Boca Raton, Florida, U.S.A.