WAVELIB

Wavelet Transform Implementation in ANSI C

http://rafat.github.io/wavelib

Author : Rafat Hussain

Contact : rafat.hsn@gmail.com

Table Of Contents

01 Introduction

Wavelib is an ANSI C implementation of decimated and undecimated 1D Fast Discrete Wavelet Transforms. Wavelet Packet Transform and Tree Decomposition have also been added to the package.

Discrete Wavelet Transform Methods Implemented

DWT/IDWT A decimated Discrete Wavelet Transform implementation using implicit signal extension and up/downsampling so it is a fast implementation. A FFT based implementation is optional but will not be usually needed. Both periodic and symmetric options are available.

SWT/ISWT Stationary Wavelet Transform. It works only for signal lengths that are multiples of 2^J where J is the number of decomposition levels. For signals of other lengths see MODWT implementation.

MODWT/IMODWT Maximal Overlap Discrete Wavelet Transform is another undecimated transform. It is implemented for signals of any length but only orthogonal wavelets (Daubechies, Symlets and Coiflets) can be deployed. This implementation is based on the method laid out in "Wavelet Methods For Wavelet Analysis" by Donald Percival and Andrew Walden.

Discrete Wavelet Packet Transform Methods Implemented

WTREE A Fully Decimated Wavelet Tree Decomposition. This is a highly redundant transform and retains all coefficients at each node. This is not recommended for compression and denoising applications.

DWPT/IDWPT Is a derivative of WTREE method which retains coefficients based on entropy methods. This is a non-redundant transform and output length is of the same order as the input.

How To Obtain The Library

Git Repository

```
git clone https://code.google.com/p/ctsa/
```

or

```
git clone git://git.code.sf.net/p/ctsa/code ctsa-code
```

or

```
git clone http://git.code.sf.net/p/ctsa/code ctsa-code
```

Or Download Zip File From

https://code.google.com/p/ctsa/source/browse/

- or -

https://sourceforge.net/projects/ctsa/files/

License : BSD 3 Clause

02 Usage : How To Integrate WAVELIB In Your Code

WAVELIB code consists of C source files and their corresponding headers. You can directly use these files in your code by including "wavelib.h" header in your code. Just make sure that all the files are in the same folder and your program can "see" them. For example, something like

```
gcc -Wall -c *.c
```

will work with GNU gcc compiler. It will build object files of all the files in the /src folder and you can link your project against these object files. This is more straightforward if you are using one of the modern IDEs as you can just plug all the files in your code and link "wavelib.h" to your project files. The IDE will do the rest. If you are an expert programmer then you may want to skip the rest of this section.

Building Shared and Static Libraries on Linux

A Simple Static Library

If you are using GNU GCC compiler then something like

```
gcc -c *.c
```

will build the object files in the src folder. You can then package the object files in a libwavelib.a static library package using

```
ar rcs libwavelib.a *.o
```

A Simple Shared Library

```
gcc -fPIC -c *.c
gcc -shared -Wl,-soname,libwavelib.so.1 -o libwavelib.so.1.0 *.o
```

You may want to move lipwavelib.so.1.0 to a separate folder before creating symlinks.

```
ln -sf libwavelib.so.1.0 libwavelib.so
ln -sf libwavelib.so.1.0 libwavelib.so.1
```

If your folder is not on the path then you will have to export the path before executing your program.

```
export LD_LIBRARY_PATH=/wavelibFOLDERLOCATION/
```

Some useful links.

http://www.yolinux.com/TUTORIALS/LibraryArchives-StaticAndDynamic.html

http://codingfreak.blogspot.com/2010/01/creating-and-using-static-libraries-in.html

http://www.techytalk.info/c-cplusplus-library-programming-on-linux-part-one-static-libraries/

http://www.techytalk.info/c-cplusplus-library-programming-on-linux-part-two-dynamic-libraries/

Building Shared and Static Libraries on Windows

Use IDE to build the libraries

I am mentioning this approach as most Windows programmers use one or more IDEs for their programming. All modern IDEs can create static and DLLs from source codes. eg., In Visual Studio you start out by creating an empty project, then by adding all the source and header files followed by "Build Solution".

Link - http://msdn.microsoft.com/en-us/library/ms235636.aspx

http://msdn.microsoft.com/en-us/library/ms235627.aspx

It is equally straightforward to create libraries in Eclipse , Codeblocks and other IDEs.

Working with Cygwin

A Static library (.a) build is identical to that with linux.

A Simple Static Library

If you are using GNU GCC compiler then something like

```
gcc -c *.c
```

will build the object files in the src folder. You can then package the object files in a libwavelib.a static library package using

```
ar rcs libwavelib.a *.o
```

A Simple DLL in Cygwin

This will build a simple standalone DLL.

```
gcc -c -fPIC *.c
gcc -shared -o libwavelib.dll *.
```

Check this link for more options, specially if you want to build the DLL as an export library.

http://cygwin.com/cygwin-ug-net/dll.html

03 Wavelet Object, Parameters and Functions

```
wave_object wave_init(char* wname); // Initialize wave object
wname - is the name of the wavelet. See below.
```

Available Wavelets

Haar : haar

Daubechies : db1,db2,.., ,db15

Biorthogonal : bior1.1 ,bior1.3 ,bior1.5 ,bior2.2 ,bior2.4 ,bior2.6 ,bior2.8 ,bior3.1 ,bior3.3 ,bior3.5 ,bior3.7 ,bior3.9 ,bior4.4 ,bior5.5 ,bior6.8

Coiflets : coif1,coif2,coif3,coif4,coif5

Symmlets: sym2,........, sym10 (Also known as Daubechies' least asymmetric orthogonal wavelets and represented by the alphanumeric la)

wave Object Parameters

```
char wname; // Wavelet Name
int filtlength;// Length of filters. They are of identical length and may be zeropadded to the same length when they are not.
int lpd_len;// Length of Low Pass Decomposition Filter
int hpd_len;// Length of High Pass Decomposition Filter
int lpr_len;// Length of Low Pass Reconstruction Filter
int hpr_len;// Length of High Pass Reconstruction Filter
double *lpd; //Low Pass Decomposition Filter
double *hpd; //High Pass Decomposition Filter
double *lpr; //Low Pass Reconstruction Filter
double *hpr; //High Pass Reconstruction Filter
```

Print wave summary

```
wave_summary(wave_object object);
```

Free wave Object

```
wave_free(wave_object object);
```

04 Wavelet Transform Class (wt) and Functions

wt Initialization

```
wt_object wt_init(wave,method,N,J);
// wave - Wavelet object created using wave_object
// method - Takes char values - "dwt", "swt" and "modwt"
// N - Length of Signal/Time Series
// J - Decomposition Levels
```

Wavelet Transform Execution

```
dwt(wt, inp);// Discrete Wavelet Transform (Decimated)
swt(wt, inp);// Stationary Wavelet Transform (Undecimated)
modwt(wt, inp); // Maximal Overlap Discrete Wavelet Transform (Undecimated)
// obj - wt object
// inp – Input signal/ Time series of length N
```

Inverse Wavelet Transform Execution

```
idwt(wt, dwtop);// Inverse Discrete Wavelet Transform (Decimated)
iswt(wt, dwtop);// Inverse Stationary Wavelet Transform (Undecimated)
imodwt(wt, dwtop); // Inverse Maximal Overlap Discrete Wavelet Transform
(Undecimated)
// obj - wt object
// dwtop – Output of length N
```

wt Object Parameters

```
wave_object wave; // wavelet object
char method; // "dwt", "swt" or "modwt"
int siglength;// Length of the original signal.
int outlength;// Length of the output DWT vector
int lenlength;// Length of the Output Dimension Vector "length"
int J; // Number of decomposition Levels
```

`int MaxIter;// Maximum Iterations J <= MaxIter`
`char ext[10];// Type of Extension used - "per" or "sym". Only available for method "dwt". Undecimated transforms use periodic extension only.`
`char cmethod[10]; // Convolution Method - "direct" or "FFT". Default is "direct". "FFT" not available for method "modwt"`
`int length[102];// Length Vector`
`double *output; // DWT Output Vector`

Accessing DWT output

1D vector wt->output stores Output of Discrete Wavelet Transform. It stores coefficients in following format:

```
[A(J) D(J) D(J-1) ..... D(1)]
```

where A(J) is the approximation coefficient vector at the Jth level while D(n) are the detail coefficient vectors at the nth level. wt->length contains the lengths of corresponding vectors. Last entry of the length vector is the length of the original signal.

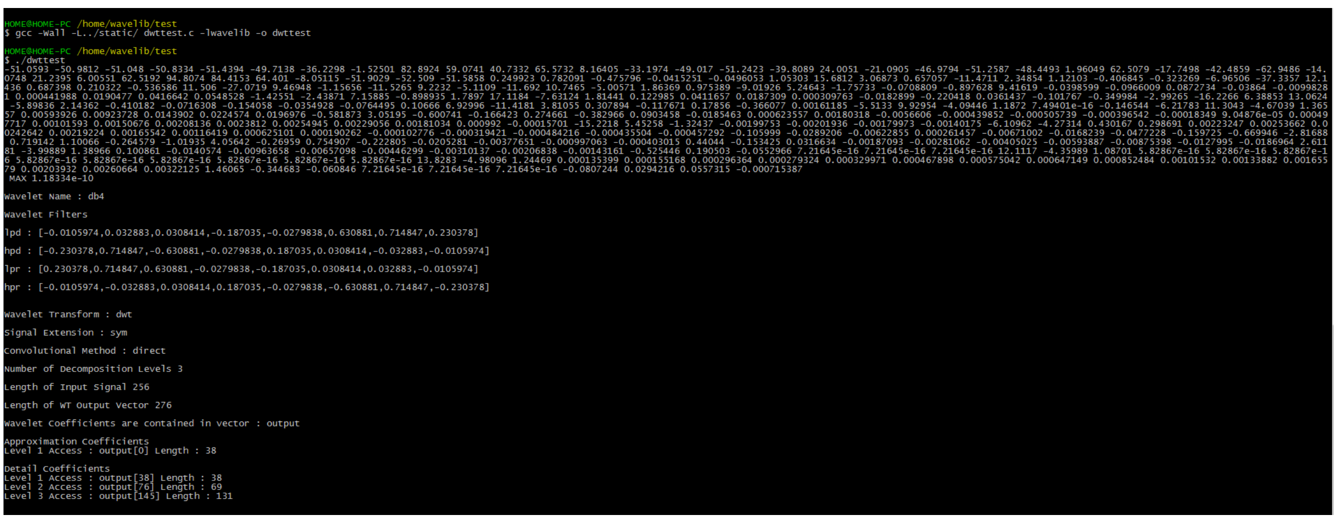
wt Functions

`setDWTExtension(wt_object wt, char *extension);// works only for dwt. Options "per" and "sym"`
`setWTConv(wt_object wt, char *cmethod);// Options "direct" and "fft". Not implemented for modwt`
`wt_summary(wt_object wt);// Print summary`
`wt_free(wt_object object);// Frees wt object`

Example 1 : DWT/IDWT

```
double absmax(double *array, int N) {
        double max;
        int i;

        max = 0.0;
        for (i = 0; i < N; ++i) {
                if (fabs(array[i]) >= max) {
                        max = fabs(array[i]);
                }
        }

        return max;
}

int main() {
        wave_object obj;
        wt_object wt;
        double *inp,*out,*diff;
        int N, i,J;

        FILE *ifp;
        double temp[1200];

        char *name = "db4";
        obj = wave_init(name);// Initialize the wavelet

        ifp = fopen("signal.txt", "r");
        i = 0;
        if (!ifp) {
                printf("Cannot Open File");
                exit(100);
        }
        while (!feof(ifp)) {
                fscanf(ifp, "%lf \n", &temp[i]);
                i++;
        }
        N = 256;

        inp = (double*)malloc(sizeof(double)* N);
        out = (double*)malloc(sizeof(double)* N);
        diff = (double*)malloc(sizeof(double)* N);
        //wmean = mean(temp, N);

        for (i = 0; i < N; ++i) {
                inp[i] = temp[i];
                //printf("%g \n",inp[i]);
        }
        J = 3;

        wt = wt_init(obj, "dwt", N, J);// Initialize the wavelet transform object
        setDWTExtension(wt, "sym");// Options are "per" and "sym". Symmetric is
the default option
        setWTConv(wt, "direct");
```

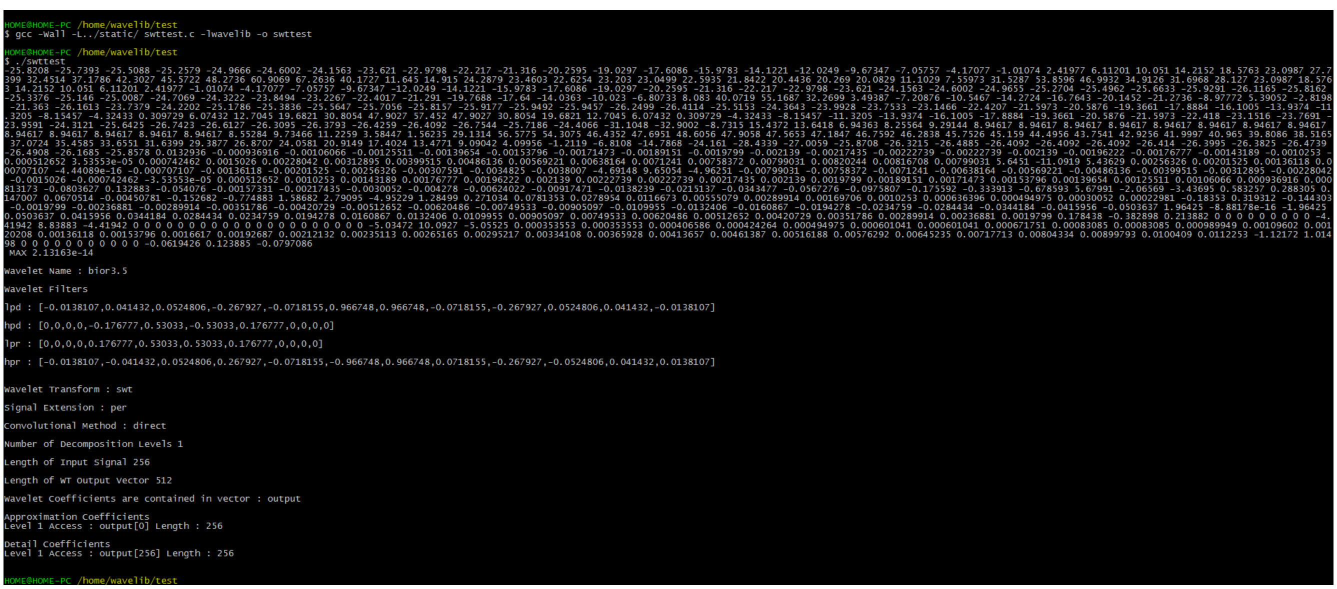
```
        dwt(wt, inp);// Perform DWT
        //DWT output can be accessed using wt->output vector. Use wt_summary to
find out how to extract appx and detail coefficients

        for (i = 0; i < wt->outlength; ++i) {
                printf("%g ",wt->output[i]);
        }

        idwt(wt, out);// Perform IDWT (if needed)
        // Test Reconstruction
        for (i = 0; i < wt->siglength; ++i) {
                diff[i] = out[i] - inp[i];
        }

        printf("\n MAX %g \n", absmax(diff, wt->siglength)); // If Reconstruction
succeeded then the output should be a small value.

        wt_summary(wt);// Prints the full summary.
        wave_free(obj);
        wt_free(wt);

        free(inp);
        free(out);
        free(diff);
        return 0;
}
```



Example 2 : SWT/ISWT

```
double absmax(double *array, int N) {
        double max;
        int i;

        max = 0.0;
        for (i = 0; i < N; ++i) {
                if (fabs(array[i]) >= max) {
                        max = fabs(array[i]);
                }
        }

        return max;
}

int main() {
        wave_object obj;
        wt_object wt;
        double *inp, *out, *diff;
        int N, i, J;

        FILE *ifp;
        double temp[1200];

        char *name = "bior3.5";
        obj = wave_init(name);// Initialize the wavelet

        ifp = fopen("signal.txt", "r");
        i = 0;
        if (!ifp) {
                printf("Cannot Open File");
                exit(100);
        }
        while (!feof(ifp)) {
                fscanf(ifp, "%lf \n", &temp[i]);
                i++;
        }
        N = 256;

        inp = (double*)malloc(sizeof(double)* N);
        out = (double*)malloc(sizeof(double)* N);
        diff = (double*)malloc(sizeof(double)* N);
        //wmean = mean(temp, N);

        for (i = 0; i < N; ++i) {
                inp[i] = temp[i];
                //printf("%g \n",inp[i]);
        }
        J = 1;

        wt = wt_init(obj, "swt", N, J);// Initialize the wavelet transform object
        setWTConv(wt, "direct");
```

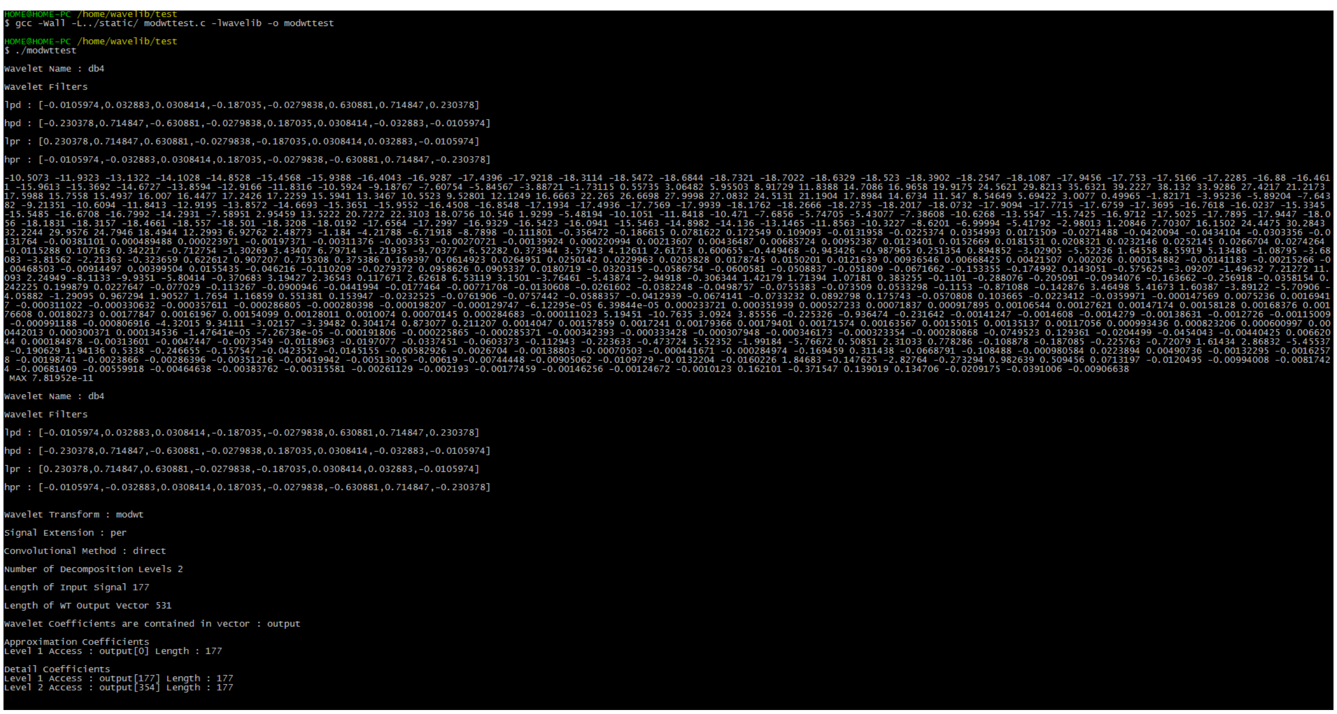
```
        swt(wt, inp);// Perform SWT
        //SWT output can be accessed using wt->output vector. Use wt_summary to
find out how to extract appx and detail coefficients

        for (i = 0; i < wt->outlength; ++i) {
                printf("%g ",wt->output[i]);
        }

        iswt(wt, out);// Perform ISWT (if needed)
        // Test Reconstruction


        for (i = 0; i < wt->siglength; ++i) {
                diff[i] = out[i] - inp[i];
        }

        printf("\n MAX %g \n", absmax(diff, wt->siglength));// If Reconstruction
succeeded then the output should be a small value.

        wt_summary(wt);// Prints the full summary.


        wave_free(obj);
        wt_free(wt);

        free(inp);
        free(out);
        free(diff);
        return 0;
}
```



Example 3 : MODWT/IMODWT

```
double absmax(double *array, int N) {
        double max;
        int i;

        max = 0.0;
        for (i = 0; i < N; ++i) {
                if (fabs(array[i]) >= max) {
                        max = fabs(array[i]);
                }
        }

        return max;
}

int main() {
        wave_object obj;
        wt_object wt;
        double *inp, *out, *diff;
        int N, i, J;

        FILE *ifp;
        double temp[1200];

        char *name = "db4";
        obj = wave_init(name);
        wave_summary(obj);

        ifp = fopen("signal.txt", "r");
        i = 0;
        if (!ifp) {
                printf("Cannot Open File");
                exit(100);
        }
        while (!feof(ifp)) {
                fscanf(ifp, "%lf \n", &temp[i]);
                i++;
        }
        N = 177;

        inp = (double*)malloc(sizeof(double)* N);
        out = (double*)malloc(sizeof(double)* N);
        diff = (double*)malloc(sizeof(double)* N);
        //wmean = mean(temp, N);

        for (i = 0; i < N; ++i) {
                inp[i] = temp[i];
                //printf("%g \n",inp[i]);
        }
        J = 2;

        wt = wt_init(obj, "modwt", N, J);// Initialize the wavelet transform
object
```

```
        modwt(wt, inp);// Perform MODWT
        //MODWT output can be accessed using wt->output vector. Use wt_summary to
find out how to extract appx and detail coefficients

        for (i = 0; i < wt->outlength; ++i) {
                printf("%g ",wt->output[i]);
        }

        imodwt(wt, out);// Perform ISWT (if needed)
        // Test Reconstruction


        for (i = 0; i < wt->siglength; ++i) {
                diff[i] = out[i] - inp[i];
        }

        printf("\n MAX %g \n", absmax(diff, wt->siglength));// If Reconstruction
succeeded then the output should be a small value.

        wt_summary(wt);// Prints the full summary.

        wave_free(obj);
        wt_free(wt);

        free(inp);
        free(out);
        free(diff);
        return 0;
}
```



05 Wavelet Tree Decomposition (wtree)

wtree initialization

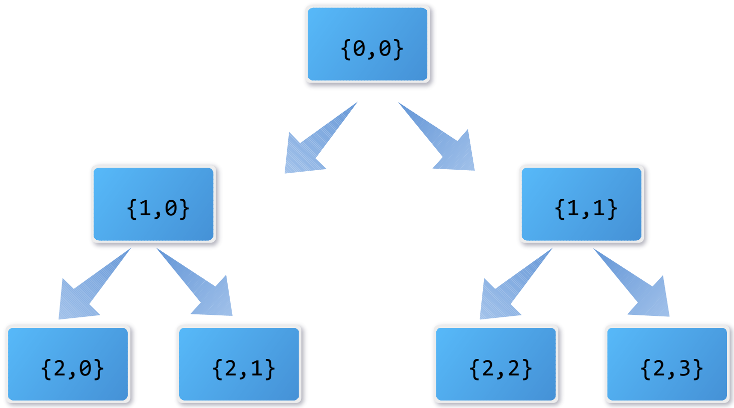
```
wtree_object wtree_init(wave,N,J);
// wave - Wavelet object created using wave_object
// N - Length of Signal/Time Series
// J - Decomposition Levels
```

wtree Execution

```
wtree(wt, inp); // Wavelet Tree Decomposition
// obj - wtree object
// inp – Input signal/ Time series of length N
```

wtree object Parameters

```
wave_object wave; // wavelet object
int siglength;// Length of the original signal.
int outlength;// Length of the output DWT vector
int J; // Number of decomposition Levels
int MaxIter;// Maximum Iterations J <= MaxIter
char ext[10];// Type of Extension used - "per" or "sym".
int *coeflength;// Size J+1 Vector containing lengths of Coefficients at each
level. All coefficients at each level have the same length. The first value is
the length of the signal (siglength). The last value is the length at the Jth
level of decomposition.
double *output; // WTREE Output Vector of size outlength
```



Wavelet Tree Full decomposition (J = 2)

wtree Functions

setWTREEExtension(wtree_object wtree, char *extension);// Options "per" and "sym"
int getWTREENodelength(wtree_object wt, int X);// Returns the length of the coefficients node at the level X of decomposition. 0 < X <= J. All coefficents at each level have the same length.
void getWTREECoeffs(wtree_object wt, int X, int Y, double *coeffs, int N);// The function return coefficents coeffs at the node {X,Y} of length N [obtained from getWTREElength(wt,X)] at level X. 0 < X <= J. 0 < Y < 2**J
wtree_summary(wtree_object wt);// Print summary
wtree_free(wtree_object object);// Frees wt object

Full Wavelet Tree decomposition is a highly redundant transformation and retains coefficients at every decomposition node. Following functions are useful in extracting coefficients.

1. <u>wtree_summary</u> : prints out how each node is stored in the output vector and how you can access it. This is a print to screen command and is not recommended to be

used in applications where speed is the primary concern.

2. getWTREENodelength & getWTREECoeffs : will give you a.) the length of the nodes at each level and b.) the node coefficients.

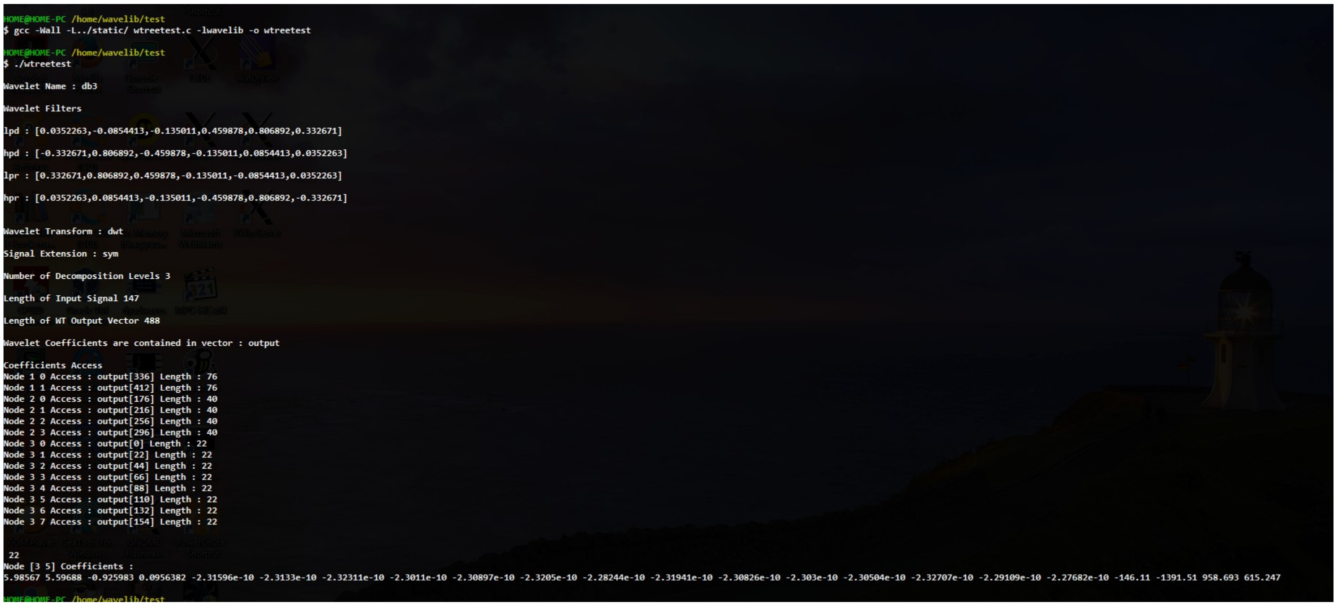
wt->output stores node coefficents beginning with Jth level from left to right. For a two level decomposition, as shown in the figure, the coefficients are stored as -

```
[{2,0} {2,1} {2,2} {2,3} {1,0} {1,1}]
```

Example wtree

```
int main() {
	int i, J, N, len;
	int X, Y;
	wave_object obj;
	wtree_object wt;
	double *inp, *oup;

	char *name = "db3";
	obj = wave_init(name);// Initialize the wavelet
	N = 147;
	inp = (double*)malloc(sizeof(double)* N);
	for (i = 1; i < N + 1; ++i) {
		inp[i - 1] = -0.25*i*i*i + 25 * i *i + 10 * i;
	}
	J = 3;

	wt = wtree_init(obj, N, J);// Initialize the wavelet transform object
	setWTREEExtension(wt, "sym");// Options are "per" and "sym". Symmetric is
the default option

	wtree(wt, inp);
	wtree_summary(wt);
	X = 3;
	Y = 5;
	len = getWTREENodelength(wt, X);
	printf("\n %d", len);
	printf("\n");
	oup = (double*)malloc(sizeof(double)* len);

	printf("Node [%d %d] Coefficients : \n",X,Y);
	getWTREECoeffs(wt, X, Y, oup, len);
	for (i = 0; i < len; ++i) {
		printf("%g ", oup[i]);
	}
	printf("\n");
```

```
        free(inp);
        free(oup);
        wave_free(obj);
        wtree_free(wt);
        return 0;
}
```



06 Discrete Wavelet Packet Transform (dwpt)

wtree initialization

```
wpt_object wpt_init(wave,N,J);
// wave - Wavelet object created using wave_object
// N - Length of Signal/Time Series
// J - Decomposition Levels
```

dwpt Execution

```
dwpt(wt, inp); // Discrete Wavelet Packet Transform
// obj - wpt object
// inp – Input signal/ Time series of length N
```

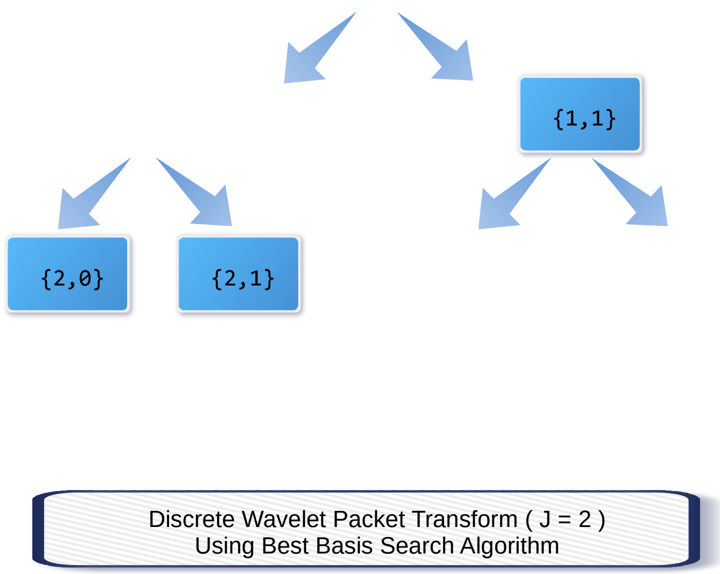
idwpt Execution

```
idwpt(wt, dwtop); // Inverse Discrete Wavelet Packet Transform
// obj - wtree object
// dwtop – DWPT output
```

dwpt object parameters

```
wave_object wave; // wavelet object
int siglength;// Length of the original signal.
int outlength;// Length of the output DWT vector
int J; // Number of decomposition Levels
int MaxIter;// Maximum Iterations J <= MaxIter
char ext[10];// Type of Extension used - "per" or "sym".
char entropy[20];// One of four values “shannon” (default), “threshold”, “norm”
and “logenergy”. These values can be set using setDWPTEntropy().
int nodes; // number of nodes retained by the Best Basis Search algorithm
int *nodeindex;// Index of retained nodes. The length of this vector is 2 * wt-
```

>nodes. See the example below. wt->nodes = 3 , the nodeindex vector is [2,0,2,1,1,1] where {2,0},{2,1} and {1,1} are the nodes in the pruned tree.
int *coeflength;// Size J+1 Vector containing lengths of Coefficients at each level. All coefficients at each level have the same length. The first value is the length of the signal (siglength). The last value is the length at the Jth level of decomposition.
double *output; // DWPT Output Vector of size outlength



DWPT Best Basis Search Algorithm

DWPT best basis search algorithm is an entropy based algorithm as described in *Ripples in Mathematics: The Discrete Wavelet Transform* by Jensen and la Cour-Harbo, Springer Verlag, 2001. The program accepts four entropy options – shannon, threshold, norm and logenergy. These values can be set using setDWPTEntropy function and the default value is “shannon”. This selected entropy is used to calculate the cost function associated with every node and a best basis is selected based on the cost function. Unlike wtree, the wpt object retains only the selected nodes and is a non-redundant transform suitable for compression and denoising applications.

dwpt functions

`setDWPTExtension(wtree_object wtree, char *extension);// Options "per" and "sym"`
`int getDWPTNodelength(wtree_object wt, int X);// Returns the length of the coefficients node at the level X of decomposition. 0 < X <= J. All coefficents at each level have the same length.`
`void getDWPTCoeffs(wtree_object wt, int X, int Y, double *coeffs, int N);// The function return coefficents coeffs at the node {X,Y} of length N [obtained from getWTREElength(wt,X)] at level X.  0 < X <= J.  0 < Y < 2**J. Also {X,Y} needs to be one of the retained nodes.`
`void setDWPTEntropy(wpt_object wt, char *entropy, double eparam);//`
`wpt_summary(wtree_object wt);// Print summary`
`wpt_free(wtree_object object);// Frees wt object`

The coefficient access is exactly the same way as explained in the wtree chapter.

Example dwpt

```
double absmax(double *array, int N) {
        double max;
        int i;

        max = 0.0;
        for (i = 0; i < N; ++i) {
                if (fabs(array[i]) >= max) {
                        max = fabs(array[i]);
                }
        }

        return max;
}

int main() {
      int i, J, N;
      wave_object obj;
      wpt_object wt;
      double *inp, *oup, *diff;

      char *name = "db4";
      obj = wave_init(name);// Initialize the wavelet
      N = 788 + 23;
      inp = (double*)malloc(sizeof(double)* N);
      oup = (double*)malloc(sizeof(double)* N);
      diff = (double*)malloc(sizeof(double)* N);
      for (i = 1; i < N + 1; ++i) {
            //inp[i - 1] = -0.25*i*i*i + 25 * i *i + 10 * i;
            inp[i - 1] = i;
      }
```

```
        J = 4;

        wt = wpt_init(obj, N, J);// Initialize the wavelet transform Tree object
        setDWPTExtension(wt, "per");// Options are "per" and "sym". Symmetric is the
default option
        setDWPTEntropy(wt, "logenergy", 0);

        dwpt(wt, inp); // Discrete Wavelet Packet Transform

        idwpt(wt, oup); // Inverse Discrete Wavelet Packet Transform

        for (i = 0; i < N; ++i) {
                diff[i] = (inp[i] - oup[i])/inp[i];
        }

        wpt_summary(wt); // Tree Summary

        printf("\n MAX %g \n", absmax(diff, wt->siglength)); // If Reconstruction
succeeded then the output should be a small value.

        free(inp);
        free(oup);
        free(diff);
        wave_free(obj);
        wpt_free(wt);
        return 0;
}
```

```
HOME@HOME-PC /home/wavelib/test
$ gcc -Wall -L../static/ dwpttest.c -lwavelib -o dwpttest

HOME@HOME-PC /home/wavelib/test
$ ./dwpttest

Wavelet Name : db4

Wavelet Filters

lpd : [-0.0105974,0.032883,0.0308414,-0.187035,-0.0279838,0.630881,0.714847,0.230378]

hpd : [-0.230378,0.714847,-0.630881,-0.0279838,0.187035,0.0308414,-0.032883,-0.0105974]

lpr : [0.230378,0.714847,0.630881,-0.0279838,-0.187035,0.0308414,0.032883,-0.0105974]

hpr : [-0.0105974,-0.032883,0.0308414,0.187035,-0.0279838,-0.630881,0.714847,-0.230378]


Signal Extension : per

Entropy : logenergy

Number of Decomposition Levels 4

Number of Active Nodes 11

Length of Input Signal 811

Length of WT Output Vector 815

Wavelet Coefficients are contained in vector : output

Coefficients Access
Node 4 0 Access : output[0] Length : 51
Node 4 1 Access : output[51] Length : 51
Node 4 2 Access : output[102] Length : 51
Node 4 3 Access : output[153] Length : 51
Node 4 4 Access : output[204] Length : 51
Node 4 5 Access : output[255] Length : 51
Node 4 8 Access : output[306] Length : 51
Node 4 9 Access : output[357] Length : 51
Node 3 3 Access : output[408] Length : 102
Node 3 5 Access : output[510] Length : 102
Node 2 3 Access : output[612] Length : 203


 MAX 1.4636e-09

HOME@HOME-PC /home/wavelib/test
```