**31.1 Definitions**

The continuous wavelet transform and its inverse are defined by the relations,

w(s,\tau) = \int f(t) \* \psi^\*\_{s,\tau}(t) dt

and,

f(t) = \int \int\_{-\infty}^\infty w(s, \tau) \* \psi\_{s,\tau}(t) d\tau ds

where the basis functions \psi\_{s,\tau} are obtained by scaling and translation from a single function, referred to as the *mother wavelet*.

The discrete version of the wavelet transform acts on equally-spaced samples, with fixed scaling and translation steps (s, \tau). The frequency and time axes are sampled *dyadically* on scales of 2^j through a level parameter j. The resulting family of functions {\psi\_{j,n}} constitutes an orthonormal basis for square-integrable signals.

The discrete wavelet transform is an O(N) algorithm, and is also referred to as the *fast wavelet transform*.

**31.2 Initialization**

The gsl\_wavelet structure contains the filter coefficients defining the wavelet and any associated offset parameters.

— Function: gsl\_wavelet \* **gsl\_wavelet\_alloc** (*const gsl\_wavelet\_type \* T, size\_t k*)

This function allocates and initializes a wavelet object of type *T*. The parameter *k* selects the specific member of the wavelet family. A null pointer is returned if insufficient memory is available or if a unsupported member is selected.

The following wavelet types are implemented:

— Wavelet: **gsl\_wavelet\_daubechies**  
— Wavelet: **gsl\_wavelet\_daubechies\_centered**

This is the Daubechies wavelet family of maximum phase with k/2 vanishing moments. The implemented wavelets are k=4, 6, ..., 20, with *k* even.

— Wavelet: **gsl\_wavelet\_haar**  
— Wavelet: **gsl\_wavelet\_haar\_centered**

This is the Haar wavelet. The only valid choice of k for the Haar wavelet is k=2.

— Wavelet: **gsl\_wavelet\_bspline**  
— Wavelet: **gsl\_wavelet\_bspline\_centered**

This is the biorthogonal B-spline wavelet family of order (i,j). The implemented values of k = 100\*i + j are 103, 105, 202, 204, 206, 208, 301, 303, 305 307, 309.

The centered forms of the wavelets align the coefficients of the various sub-bands on edges. Thus the resulting visualization of the coefficients of the wavelet transform in the phase plane is easier to understand.

— Function: const char \* **gsl\_wavelet\_name** (*const gsl\_wavelet \* w*)

This function returns a pointer to the name of the wavelet family for *w*.

— Function: void **gsl\_wavelet\_free** (*gsl\_wavelet \* w*)

This function frees the wavelet object *w*.

The gsl\_wavelet\_workspace structure contains scratch space of the same size as the input data and is used to hold intermediate results during the transform.

— Function: gsl\_wavelet\_workspace \* **gsl\_wavelet\_workspace\_alloc** (*size\_t n*)

This function allocates a workspace for the discrete wavelet transform. To perform a one-dimensional transform on *n* elements, a workspace of size *n* must be provided. For two-dimensional transforms of *n*-by-*n* matrices it is sufficient to allocate a workspace of size *n*, since the transform operates on individual rows and columns. A null pointer is returned if insufficient memory is available.

— Function: void **gsl\_wavelet\_workspace\_free** (*gsl\_wavelet\_workspace \* work*)

This function frees the allocated workspace *work*.

### 31.3 Transform Functions

This sections describes the actual functions performing the discrete wavelet transform. Note that the transforms use periodic boundary conditions. If the signal is not periodic in the sample length then spurious coefficients will appear at the beginning and end of each level of the transform.

#### 31.3.1 Wavelet transforms in one dimension

— Function: int **gsl\_wavelet\_transform** (const gsl\_wavelet \* w, double \* data, size\_t stride, size\_t n, gsl\_wavelet\_direction dir, gsl\_wavelet\_workspace \* work)  
— Function: int **gsl\_wavelet\_transform\_forward** (const gsl\_wavelet \* w, double \* data, size\_t stride, size\_t n, gsl\_wavelet\_workspace \* work)  
— Function: int **gsl\_wavelet\_transform\_inverse** (const gsl\_wavelet \* w, double \* data, size\_t stride, size\_t n, gsl\_wavelet\_workspace \* work)

These functions compute in-place forward and inverse discrete wavelet transforms of length n with stride stride on the array data. The length of the transform n is restricted to powers of two. For the transform version of the function the argument dir can be either forward (+1) or backward (-1). A workspace work of length n must be provided.

For the forward transform, the elements of the original array are replaced by the discrete wavelet transform f\_i -> w\_{j,k} in a packed triangular storage layout, where j is the index of the level j = 0 ... J-1 and k is the index of the coefficient within each level, k = 0 ... (2^j)-1. The total number of levels is J = \log\_2(n). The output data has the following form,

(s\_{-1,0}, d\_{0,0}, d\_{1,0}, d\_{1,1}, d\_{2,0}, ...,

d\_{j,k}, ..., d\_{J-1,2^{J-1}-1})

where the first element is the smoothing coefficient s\_{-1,0}, followed by the detail coefficients d\_{j,k} for each level j. The backward transform inverts these coefficients to obtain the original data.

These functions return a status of GSL\_SUCCESS upon successful completion. GSL\_EINVAL is returned if n is not an integer power of 2 or if insufficient workspace is provided.

### 31.4 Examples

The following program demonstrates the use of the one-dimensional wavelet transform functions. It computes an approximation to an input signal (of length 256) using the 20 largest components of the wavelet transform, while setting the others to zero.

#include <stdio.h>

#include <math.h>

#include <gsl/gsl\_sort.h>

#include <gsl/gsl\_wavelet.h>

int

main (int argc, char \*\*argv)

{

int i, n = 256, nc = 20;

double \*data = malloc (n \* sizeof (double));

double \*abscoeff = malloc (n \* sizeof (double));

size\_t \*p = malloc (n \* sizeof (size\_t));

FILE \* f;

gsl\_wavelet \*w;

gsl\_wavelet\_workspace \*work;

w = gsl\_wavelet\_alloc (gsl\_wavelet\_daubechies, 4);

work = gsl\_wavelet\_workspace\_alloc (n);

f = fopen (argv[1], "r");

for (i = 0; i < n; i++)

{

fscanf (f, "%lg", &data[i]);

}

fclose (f);

gsl\_wavelet\_transform\_forward (w, data, 1, n, work);

for (i = 0; i < n; i++)

{

abscoeff[i] = fabs (data[i]);

}

gsl\_sort\_index (p, abscoeff, 1, n);

for (i = 0; (i + nc) < n; i++)

data[p[i]] = 0;

gsl\_wavelet\_transform\_inverse (w, data, 1, n, work);

for (i = 0; i < n; i++)

{

printf ("%g\n", data[i]);

}

gsl\_wavelet\_free (w);

gsl\_wavelet\_workspace\_free (work);

free (data);

free (abscoeff);

free (p);

return 0;

}

The output can be used with the gnu plotutils graph program,

$ ./a.out ecg.dat > dwt.dat

$ graph -T ps -x 0 256 32 -h 0.3 -a dwt.dat > dwt.ps