INTERNATIONAL ATOMIC ENERGY AGENCY
UNITED NATIONS EDUCATIONAL, SCIENTIFIC AND CULTURAL ORGANIZATION
INTERNATIONAL CENTRE FOR THEORETICAL PHYSICS
I.C.T.P., P.O. BOX 586, 34100 TRIESTE, ITALY, CABLE: CENTRATOM TRIESTE

H4.SMR/854-24

College on Computational Physics

15 May - 9 June 1995

Wavelets and Applications

J. Annett

University of Bristol
Bristol, United Kingdom

Wavelets and Applications

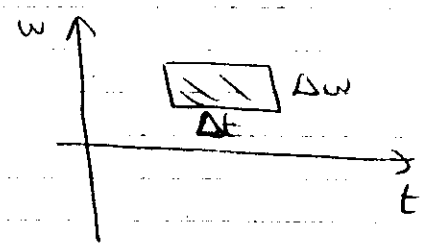
College on Computational Physics May/June 1995
ICTP Trieste

J.F. Annett
University of Bristol
H.H. Wills Physics Labor
Royal Fort, Tyndall Ave
Bristol BS8 1TL, U
James.Annett@bristol.ac.uk

1.1 Introduction

'Wavelets' are a rapidly growing field.

Its origins lie in time/frequency analysis of signals.



Rather than Fourier series, which give a decomposition of a signal into pure frequencies, it is often useful to have both time and frequency information.

Provided we obey $\Delta\omega\Delta t \geq 1$ both time and frequency can be known.
Example: a musical score tells you what note to play, and when to play it!

In physics the ideas of scaling, renormalization group and fractals also partly motivated the development of wavelets. How do you describe a fractal curve (eg Mandelbrot set etc) compactly? Since it has structure on all length scales even Fourier series would be not very helpful.

The key mathematical results in "Wavelet theory" came in the mid 1980's in the work of Mallat, Meyer and Daubechies especially.

Now wavelets are finding many practical applications in:

- Signal processing
- Image compression
- Mathematics

In physics applications are still developing. Up till now they are limited to

- electronic structure
- statistical mechanics and field theory
- non-linear wave propagation

Hopefully there is scope for the student to develop new ones!

1.2 Key concepts

"Wavelets" employ two key ideas. The first is multiresolution analysis, the second is the wavelet itself.

A <u>multiresolution</u> analysis of a function is an analysis which separates the function into components on different <u>scales</u>.

The resolution needed can vary from place to place. Very fine resolution may be needed near singularities, much coarser resolution would be adequate elsewhere.

The idea of different resolutions is not new, eg multigrid, finite element or adaptive grid methods.

What is unique to wavelets is the existence of scaling transformations which map the function from one resolution scale to another.

This is somewhat like the Kadanoff "block spin" renormalization method mapping a fine resolution Ising system to one on a coarser resolution.

The second key concept is the wavelet itself.

$\psi(x)$ is a possible wavelet if it is localized in some region (or at least rapidly decaying at infinity), oscillates $\left(\int \psi(x)\,dx = 0\right)$, and is reasonably continuous or differentiable.

Some wavelets have additional orthogonality properties

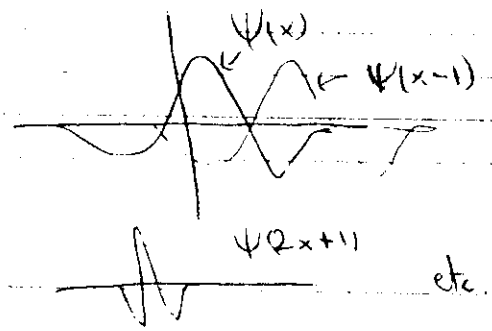
translation $\int \psi(x)\,\psi(x-n) = 0$ n any integer, except 0

scaling by 2 $\int \psi(x)\,\psi(2x-n) = 0$ n any integer

scaling by any power of 2 $\int \psi(x)\,\psi(2^{\ell}x-n) = 0$ n,ℓ any integers (except $\ell=0$ & $n=0$)

N.B Translations by different distances, scaling by different powers also possible - but 2 is fine for most uses.

eg:



The Daubechies wavelets are zero outside a finite interval and satisfy these orthogonality properties. Essentially form an orthonormal basis, with all scales included!

1.3 Some good references are:

"An introduction to wavelets" C.K. Chui

(this is Volume 1 in a series called "Wavelets and Applications". Now up to Volume 5 exist)

"Ten lectures on wavelets" I. Daubechies

"Wavelets and Operators" Y. Meyer (more mathematical)

See also "Numerical Recipes" in the 2nd edition a short section on wavelets was added; it is not in the 1st edition.

This includes example programs.

1.4 Quick review of discrete Fourier transforms

The matrix $U_{nm} = \frac{1}{\sqrt{N}} e^{-2\pi i nm/N}$

is a unitary $N \times N$ matrix, where $n = 0, 1, \ldots N-1$
$m = 0, 1, \ldots N-1$

ie it obeys $U^\dagger U = U U^\dagger = 1$ or $U^{-1} = U^\dagger$

Proof is simple:

$$[U^\dagger U]_{nm} = U^\dagger_{np} U_{pm} = U^*_{pn} U_{pm} = \frac{1}{N} \sum_{p=0}^{N-1} e^{2\pi i p(n-m)} = \delta_{nm}$$

If f_n is a vector of N numbers $\left(\text{say } f(x_n) \quad x_n = \frac{nL}{N}\right)$

then $$g_m = \frac{1}{\sqrt{N}} U_{mn} f_n = \frac{1}{N} \sum_{n=0}^{N-1} e^{-ik_m x_n} f_n$$

$$f_n = \sqrt{N} [U^{-1}]_{nm} g_m = \sum_{m=0}^{N-1} e^{ik_m x_n} g_m$$

the usual Fourier transform and its inverse $\left(\text{Here } k_m = \frac{2\pi m}{L}\right)$

Parseval's theorem is a simple consequence of unitarity

The norm of a vector is unchanged by unitary transformations

$$\sum_m |g_m|^2 = \frac{1}{N} f_n^* U^\dagger_{nm} U_{mn'} f_{n'}$$

$$= \frac{1}{N} \sum |f_n|^2 \quad \text{since } U^\dagger U = \mathbb{1}$$

Also a unitary matrix can be viewed as a set of orthonormal vectors
$U_{nm} = (x_n^{(0)}, x_n^{(1)}, x_n^{(2)} \ldots)$ $x_n^{(m)}$ is the mth column, row n.

Here the orthonormal vectors are $\frac{1}{\sqrt{N}} e^{-ik_m x_n}$

1.5 Quick review of Fourier series

going now to a continuous variable x $\quad 0 \le x < L$

$$f(x) = \sum_m e^{ik_m x} g_m \qquad k_m = \frac{2\pi m}{N}$$

now an infinite series.

now $\frac{1}{\sqrt{L}} e^{ik_m x}$ are an orthonormal set of functions

using the norm, or inner product

$$\langle f | g \rangle \text{ or } (f, g) = \int_0^L f^*(x)\, g(x)\, dx$$

$$\|f\|^2 \equiv (f, f) = \int_0^L |f(x)|^2 dx$$

clearly $\left(\frac{1}{\sqrt{L}} e^{ik_m x}, \frac{1}{\sqrt{L}} e^{ik_n x} \right) = \delta_{nm}$

using orthogonality we find

$$g_m = \frac{1}{\sqrt{L}} \left(\frac{1}{\sqrt{L}} e^{ik_m x}, f \right)$$ a component along one basis vector

$$= \frac{1}{L} \int_0^L e^{-ik_m x} f(x)\, dx$$ the usual formula

Now the above is fine for mathematics, but numerically we always have to truncate the series to a finite number of terms.

Let $f^{(N)}(x)$ be the sum of the truncated series

$$f^{(N)}(x) = \sum_{-N/2}^{N/2} e^{ik_m x} g_m$$

(a sum of $N+1$ terms).

We can examine how $F^{(N)}$ approaches f as $N \to \infty$.

It's easy to see that

$$\|f - F^{(N)}\|^2 = \|f\|^2 - \|F^{(N)}\|^2 \quad \text{(Pythagoras theorem!)}$$

$$= \|f\|^2 - L \sum_{m=-\frac{N}{2}}^{N/2} |g_m|^2$$

So iff $\int |f - F^N|^2 \, dx \to 0$ as $N \to \infty$

we get Parseval's theorem $\int_0^L |F(x)|^2 \, dx = L \sum_{m=-\infty}^{\infty} |g_m|^2$

Discussion of the mathematical conditions on f such that this converges are beyond our scope.

However it will be worth examining $F^{(N)}(x)$ more carefully.

By truncating the series we have effectively multiplied g_m by a step function! $g_m \to g_m \Theta(|k_m - k_{N_2}|)$

Therefore, by convolution theorem $F^{(N)}$ is a convolution of f!

This is easy to check

$$F^{(N)}(x) = \sum_{m=-\frac{N}{2}}^{N/2} e^{ik_m x} g_m$$

$$= \frac{L}{L} \int_0^L \sum_{m=-\frac{N}{2}}^{\frac{N}{2}} e^{ik_m (x - x')} f(x') \, dx'$$

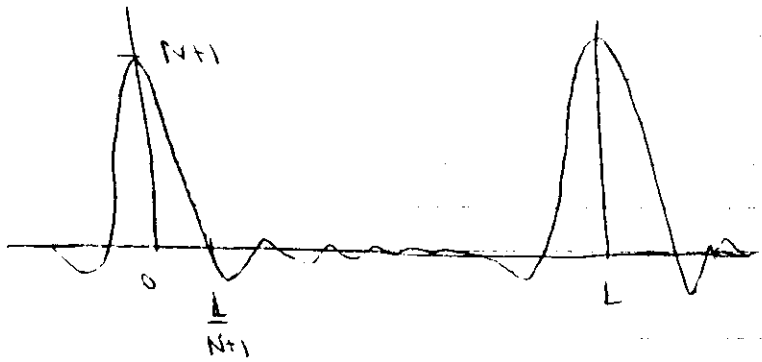
$$= \frac{1}{L} \int_0^L K(x - x') f(x') \, dx'$$

The function $K(x - x')$ is just

$$K(x - x') = \sum_{m=-\frac{N}{2}}^{N/2} e^{ik_m (x - x')} = \frac{\sin(k_{N+1} x/2)}{\sin(k_1 x/2)}$$

by summing a geometric series.

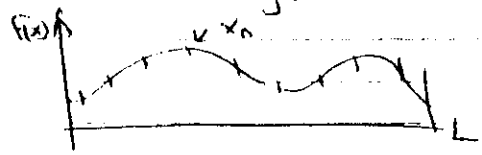
$K(x-x')$



$K(x-x')$ is a $\frac{\sin x}{x}$ type function. Height $N+1$ at $x = 0, L$ and width $L/N+1$. As $N \to \infty$ it approaches a δ function. For finite N your function is convolved with this. The oscillations lead to the "Gibbs phenomenon" in the series for a square wave.

1.6 The above result can also be viewed another way:

as <u>Fourier interpolation</u>



Suppose we only knew $f(x)$ by sampling a discrete set of data points $f(x_n)$ $\quad x_n = \frac{nL}{N+1}$ ($N+1$ data points for easiest algebra!)

We could Fourier transform by discrete F.T and get g_m for $-N/2, \ldots N/2$ ($N+1$ Fourier components)

Then put

$$f(x) = \frac{1}{N+1} \sum_{m=-N/2}^{N/2} e^{ik_m x} g_m$$

$$= \frac{1}{N+1} \sum_n K(x - x_n) f(x_n)$$

$f(x)$ exactly interpolates the data points, and is infinitely differentiable. The sum is just a discrete form of convolution!

(?)

Clearly $K(x-x_n)$ is also a basis function

and $\left(K(x-x_n) , K(x-x_m) \right) = (N+1) L \, \delta_{nm}$

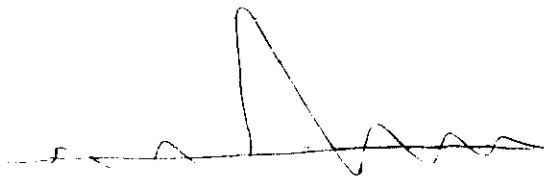
so $\frac{1}{\sqrt{L(N+1)}} K(x-x_n)$ are an orthonormal set.

N.B. $K(x-x_n)$ also vanishes at the other data points

$$K(x_m - x_n) = 0 , \quad n \neq m$$

Finally, imagine the limit $L \to \infty$ keeping the spacing of the data points $\Delta x = \frac{L}{N+1}$ fixed

$$\frac{1}{\sqrt{(N+1)L}} K(x-x_n) \to \frac{1}{\sqrt{\Delta x}} \frac{\sin \pi (x-x_n)/\Delta x}{\pi (x-x_n)/\Delta x}$$



but this is a very long ranged function $\sim \frac{1}{x}$ as $x \to \pm \infty$

It is also not normalizable

$$\int_{-\infty}^{\infty} dx \left| \frac{\sin \pi x/\Delta x}{\pi x/\Delta x} \right|^2 \to \infty$$

It is not useful as a 'wavelet', despite its orthogonality properties.

1.7 Fourier series thus have the following disadvantages:

1) To interpolate a function $f(x)$ based on a discrete sampling $f(x_n)$ need to use <u>all</u> the data. Since $\frac{\sin x}{x}$ is long ranged cannot usefully work "on the fly" on part of an essentially infinite sequence of f's. For example $f(x)$ might be data from a telescope or other apparatus essentially infinite.

To analyse by Fourier have to restrict to a finite number of points N.

2) The resolution - or spacing between points has to be uniform for Fourier interpolation. Must also be set fine enough to capture all the most detailed aspects of the signal.

If the signal is smooth followed by rapid varying short sections a lot of effort is wasted sampling the smooth parts on a fine scale.

Cannot add more points where $f(x)$ interesting (although spline interpolation lets you do this - more on that later)

Wavelets address these deficiencies. The wavelet functions $\psi(x)$ are well localized, solving problem 1. The wavelets exist on all resolution scales, solving problem 2.

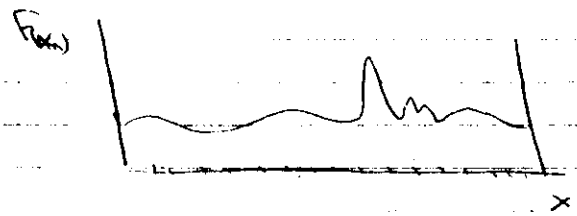
1.8 Multiresolution analysis

The idea : separate a function $f(x)$ into its "fast" and slow components (cf. renormalization group)

The slow components can be represented on a coarser mesh.

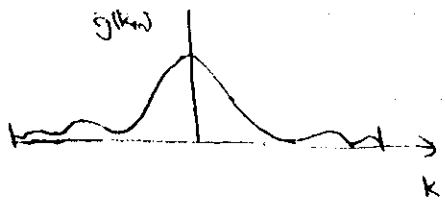
First lets give a heuristic approach

Suppose $f(x_n)$ known on a set of N points from 0 to L



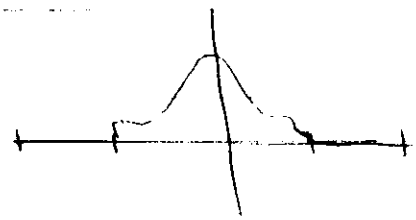
F has some smooth regions and some sharp regions

Now take the discrete Fourier transform

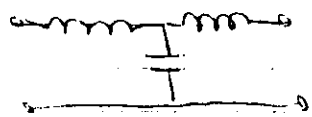


for clarity I've put $k=0$ in the center of the plot. For a discrete F.T. k is periodic in $\frac{2\pi N}{L}$

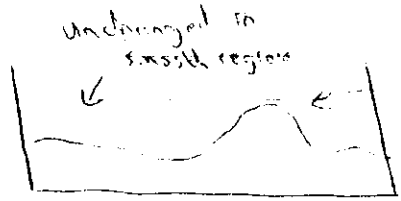
Now "window" the data by dropping Fast Fourier components



this is a "low-pass" filter like the circuit



Now FFT back we get a smoothened $f(x)$



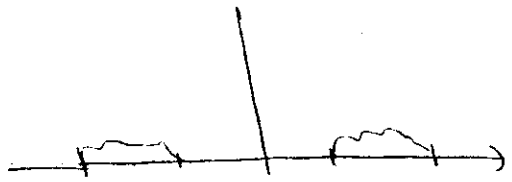
(N.B the smoothed $f(x)$ could also be found by convolving with $k(x_m - x_n)$)

Note that $N/2$ point suffice to exactly represent the smooth $f(x)$

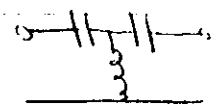
Have we lost all the high frequency information?

We can store that separately

Keep only the high frequencies

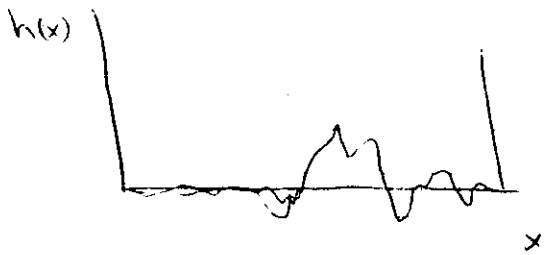


again $N/2$ components



"high pass filter"

transform back to real space get a function $h(x)$



$h(x)$ has all the information about the high frequencies. No low frequency information.
Eg its average is zero.
Again $h(x)$ has $N/2$ data points.

Now the <u>key point</u>

We can apply this scaling <u>recursively</u>!

We get a sequence of smoother and smoother functions

with $N/2$, $N/4$, $N/8$, ... data points

We can exactly reconstruct f from this information

if we keep the <u>last</u> f

and the differences h on <u>each</u> scale

$$N = \underbrace{N/2}_{\text{first } h} + \underbrace{N/4}_{\text{second } h} + N/8 + \cdots + \underbrace{N/2^L}_{\text{last } h} + \underbrace{N/2^L}_{\text{last } f}$$

Lecture 2

2.1 In practice we do not perform multiresolution analysis using Fourier transforms, but the idea always works.

Decompose a function $f(x)$ with N data points

into a second function $f_1(x)$ "smooth" components
and differences $h_1(x)$ the oscillating components

$f(x)$ — N points

$\downarrow \searrow$

$f_1(x)$ $h_1(x)$ — $N/2 + N/2$ points.

But we need not stop there, we can go on $\log_2 N$ times!

$f(x)$

$\downarrow \searrow$

$f_1(x)$ $h_1(x)$ — $N/2$ points in h_1

$\downarrow \searrow$

$f_2(x)$ $h_2(x)$ — $N/4$ in h_2

$\downarrow \searrow$

$f_3(x)$ $h_3(x)$ — $N/8$ in h_3

$\downarrow \searrow$

$f_L(x)$ $h_L(x)$ — $N/2^L$ in h_L

$+ N/2^L$ in f_L

We can go down to $N/2^L = 1$, when there is only one function value left!

(By convention the last value is always the sum of the initial data values)

2.2 The process for doing this and generating a hierarchical tree of functions is quite simple!

We shall use the simplest example: the Haar wavelet transform.

We do not need to do Fourier transform / Filter / inverse Fourier, since this is equivalent to convolution with $\frac{\sin k_{N+1} x/2}{\sin k_i x/2}$.

We shall a) do the convolution directly in real space

b) use a more short range convolving function.

The Haar wavelet transform corresponds to the simplest "block-spin" coarse graining of the function. Just take the sum of 2 adjacent points to be the new value!

$$f_1(x_n) = f_0(x_n) + f_0(x_{n+1})$$

I am labelling the original function as f_0, its first scaled copy f_1, and so on.

In order to retain the information, we need to keep information about the difference

$$h_1(x_{n+1}) = f_0(x_n) - f_0(x_{n+1})$$

Ie we just have the matrix multiplication

$$\begin{pmatrix} f_1(x_n) \\ h_1(x_{n+1}) \end{pmatrix} = \begin{pmatrix} 1 & 1 \\ 1 & -1 \end{pmatrix} \begin{pmatrix} f_0(x_n) \\ f_0(x_{n+1}) \end{pmatrix}$$

The original function has N data points.
$f_1(x_n)$ has $N/2$ data points - which I will take as the even points

$h_1(x_n)$ has $N/2$ data points - which I will take as the odd points

Repeat this iteratively:

$$F_{L+1}(x_n) = F_L(x_n) + f_L(x_n + d)$$

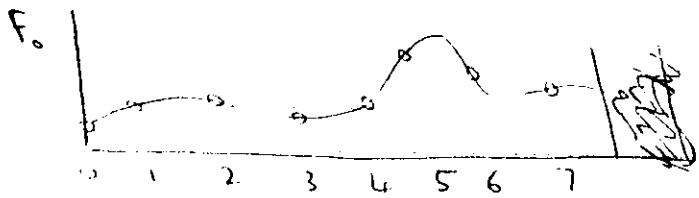
$$h_{L+1}(x_n + d) = F_L(x_n) - f_L(x_n + d)$$

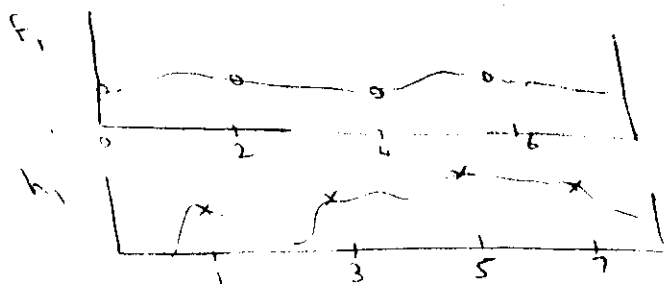
d is the grid spacing at level L, which is $L/N \times 2^{L-1}$

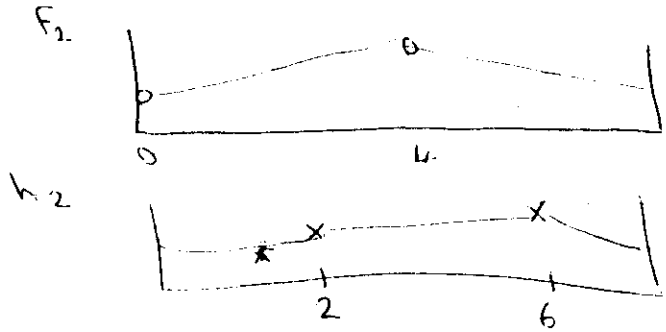
so $F_L(x_n)$ exists on points $0, d, 2d, \ldots$ $N/2^{L-1}$ points

$f_{L+1}(x_n)$ exists on points $0, 2d, 4d, \ldots$ $N/2^L$ points

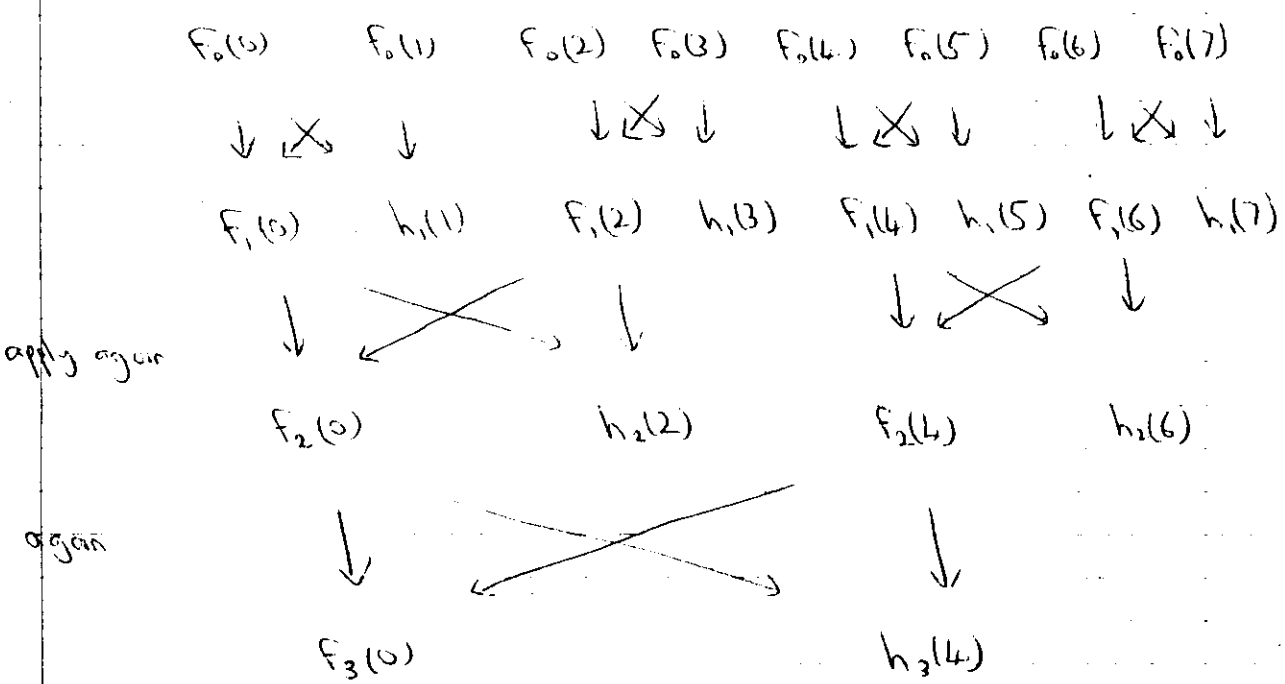
$h_{L+1}(x_n)$ exists on points $d, 3d, 5d, \ldots$ $N/2^L$ points.







We have the dependencies



(i) Note that in practice the process can take place in a single array, each point in place according to its argument.

(ii) Check $f_3(0) = f_0(0) + \dots f_0(7)$

obeying the sum rule. Also if $f_0(n)$ were constant, all the h_L would be zero.

(iii) Each step is a matrix multiply:

$d = 2^L$

$$\begin{pmatrix} f_{L+1}(0) \\ h_{L+1}(d) \\ f_{L+1}(2d) \\ h_{L+1}(3d) \\ \vdots \end{pmatrix} \begin{pmatrix} 1 & 1 & & & \\ 1 & -1 & & & \\ & & 1 & 1 & \\ & & 1 & -1 & \\ & & & & \ddots \end{pmatrix} \begin{pmatrix} f_L(0) \\ f_L(d) \\ f_L(2d) \\ f_L(3d) \\ \vdots \end{pmatrix}$$

(iv)

This matrix is almost unitary!

$$M^+ M = 2\,\mathbb{1}$$

So to invert the matrix we just take the transpose and multiply by $\frac{1}{2}$.

$$\begin{pmatrix} f_i(0) \\ f_i(d) \\ f_i(2d) \\ \vdots \\ \end{pmatrix} = \frac{1}{2} \begin{pmatrix} 1 & -1 & & & \\ 1 & 1 & & & \\ & & 1 & -1 & \\ & & 1 & 1 & \\ & & & & \ddots \end{pmatrix} \begin{pmatrix} f_{i+1}(0) \\ h_{i+1}(d) \\ f_{i+1}(2d) \\ h_{i+1}(3d) \\ \vdots \end{pmatrix}$$

(v) The whole process of transformation is repeated

matrix multiplication

$$\begin{pmatrix} f(0) \\ h_1 \\ \vdots \end{pmatrix} = \cdots M^{(32)} M^{(21)} M^{(10)} \begin{pmatrix} f_0 \end{pmatrix}$$

where $M^{(ij)}$ goes from level j to i.

The inverse is just

$$\begin{pmatrix} f_0 \end{pmatrix} = \cdots M^{(01)} M^{(12)} M^{(23)} \cdots \begin{pmatrix} f(0) \\ h \\ \vdots \end{pmatrix}$$

where $M^{(ij)}$ goes from level i to j.

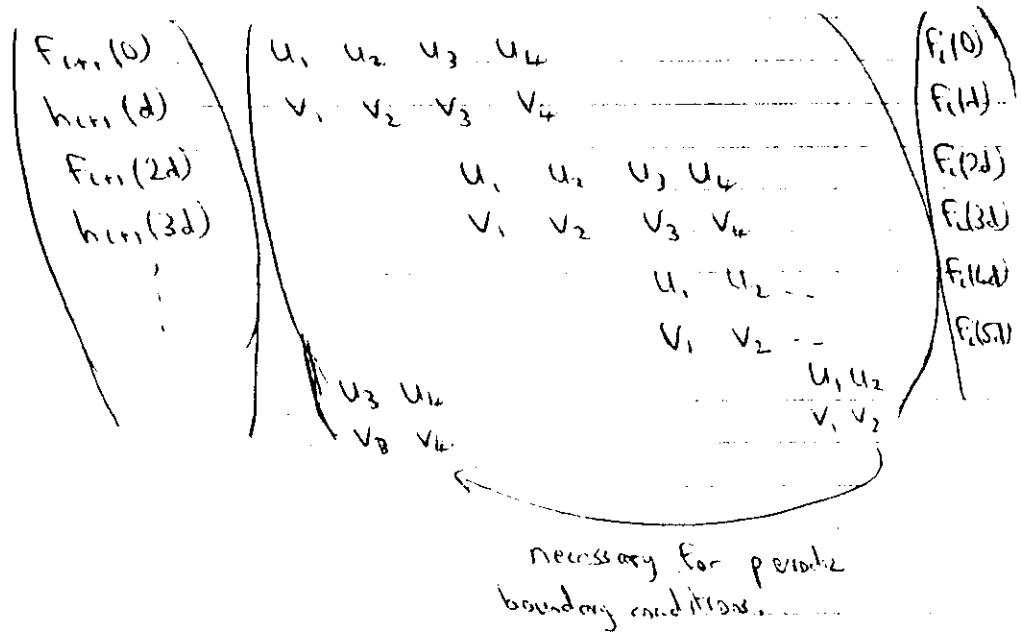
2.3 The Daubechies wavelet transform

is a generalization of the Haar transform. There are a sequence of possible transforms, of class M. Haar is class 0, we shall concentrate on class 1. Increasing class leads to smoother interpolations. Class 0 is not continuous.

From the Daubechies transform we shall construct wavelets which have all the orthonormality properties discussed in the introduction.

We make a smoother transformation, by using matrices of larger bandwidth.

The class 1 matrix is



We want to make the matrix unitary, or almost unitary, apart from normalization.

To do this the rows must be orthogonal vectors

(or the columns of the transpose)

this implies $(u_1, u_2, u_3, u_4) \cdot (v_1, v_2, v_3, v_4) = 0$

$(u_1, u_2, u_3, u_4) \cdot (0\ 0\ u_1\ u_2) = 0$

$(u_1, u_2, u_3, u_4) \cdot (0\ 0\ u_3\ v_2) = 0$

$(u_1\ u_2\ u_3\ u_4) \cdot (u_3\ u_4\ 0\ 0) = 0$

$(u_1\ u_2\ u_3\ u_4) \cdot (v_3\ v_4\ 0\ 0) = 0$

We also impose

$$\sum v_n = 0$$

$$\sum n v_n = 0$$

(For class m $\sum n^m v_n = 0$)

and normalization

$$2 = \sum u_n$$

(like Haar system gives sum rule)

8 conditions on 8 variables - a unique solution!

$u_1 = (1+\sqrt{3})/4$ $\quad v_1 = -u_4$

$u_2 = (3+\sqrt{3})/4$ $\quad v_2 = u_3$

$u_3 = 1 - u_1$ $\quad v_3 = -u_2$

$u_4 = 1 - u_2$ $\quad v_4 = u_1$

2.4 Wavelet.f90

READ.ME
/lecturer/annett/public/wavelet.F90
driver.f90

If everything seems abstract - its time to run it on the computer and see what it does!

Fortran90 subroutine 'wavelet' implements the class I Daubechies transform. By changing the u/v coefficients it can do Haar too by commenting/uncommenting some lines.

Use it by:

call wavelet(n, f, initial, final)

n is the length of the data array. It must be a power of 2.

(This could be relaxed with care.)

f(0:n-1) is a double precision array of data values input.

The transformed output is stored in f "in place".

The transform takes the data from resolution level "initial" to resolution level "Final".

For simplest use set initial = 0 Final = m $(2^m = n)$

and it will fully transform the array down to one f value $f_m(0)$ (in position 0 of the array) and n-1 h values $h_1, h_2, h_3, \ldots$

They are laid out as described for the Haar system

h_L occupies points $d, 3d, 5d, \ldots d = 2^L$

To find out what coefficient is in site i, just divide i by powers of 2 it is $2^L \times$ an odd number say $2^L j$, Then it is $h_L(2^L j)$ in my notation used before

To examine intermediate scaled functions

Using finest = initial + 1 repeatedly.

$F_L(0), F_L(d), \ldots$ are in positions $0, d, 2d, \ldots d = 2^L$

where L = finest.

An example program driver.f90

sets up a test function printing out the scaled f and 'wavelet' h values at each level.

f_0 is in fort.10

f_1 fort.11 h_1 in fort.51

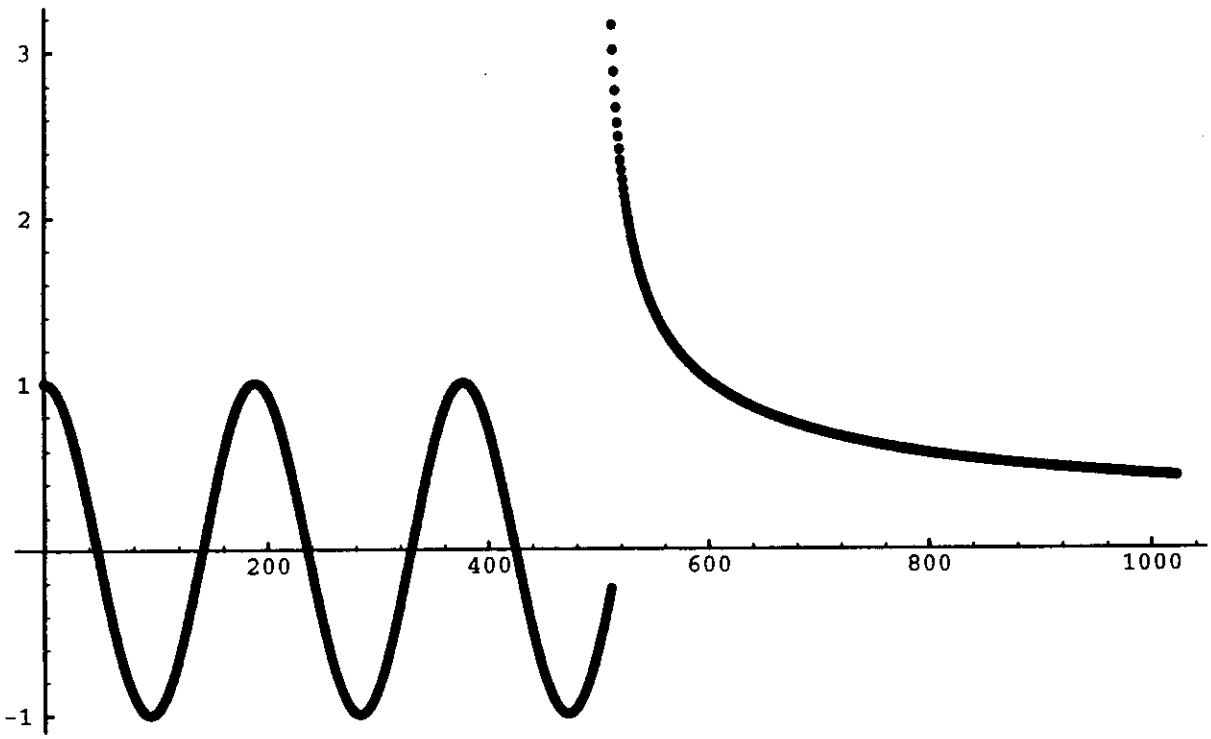
f_2 fort.12 h_2 in fort.52

⋮ ⋮

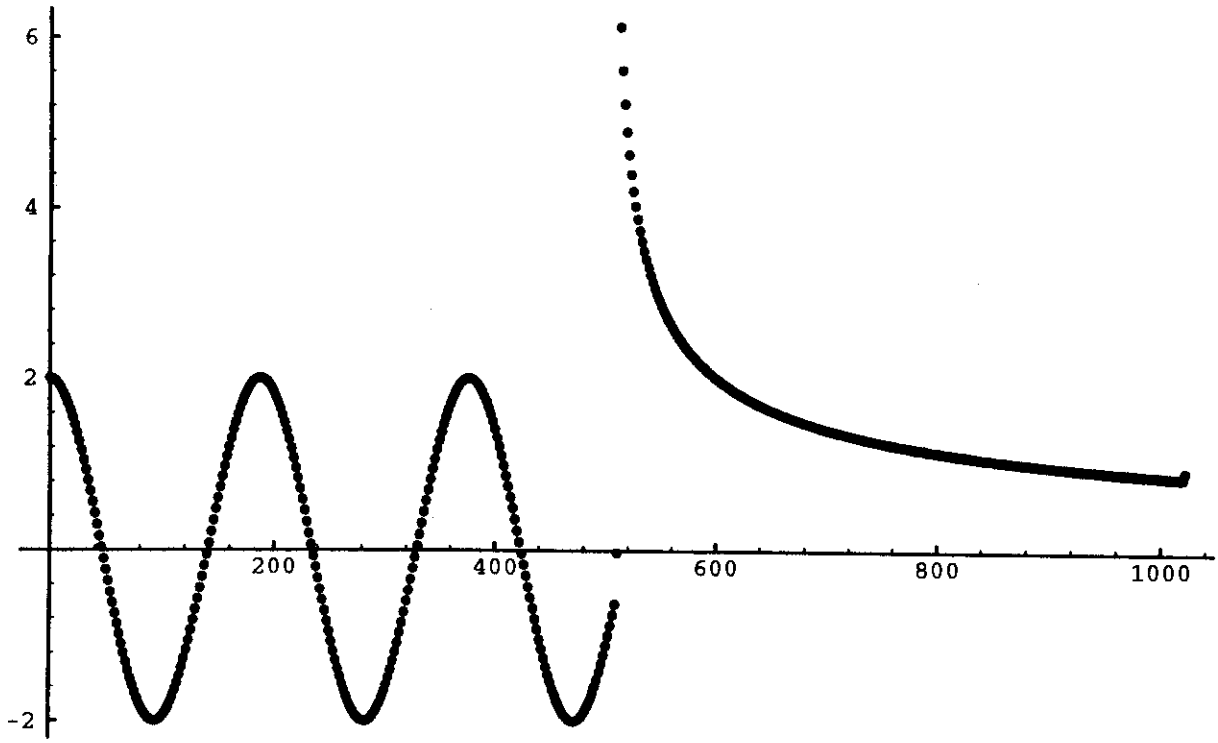
After the full transform it does the inverse transform reconstructing f_0. The reconstructed f is output in fort.9 – it is essentially identical to f_0

[NB inverse transformations go by just requesting a final level less than the initial level.

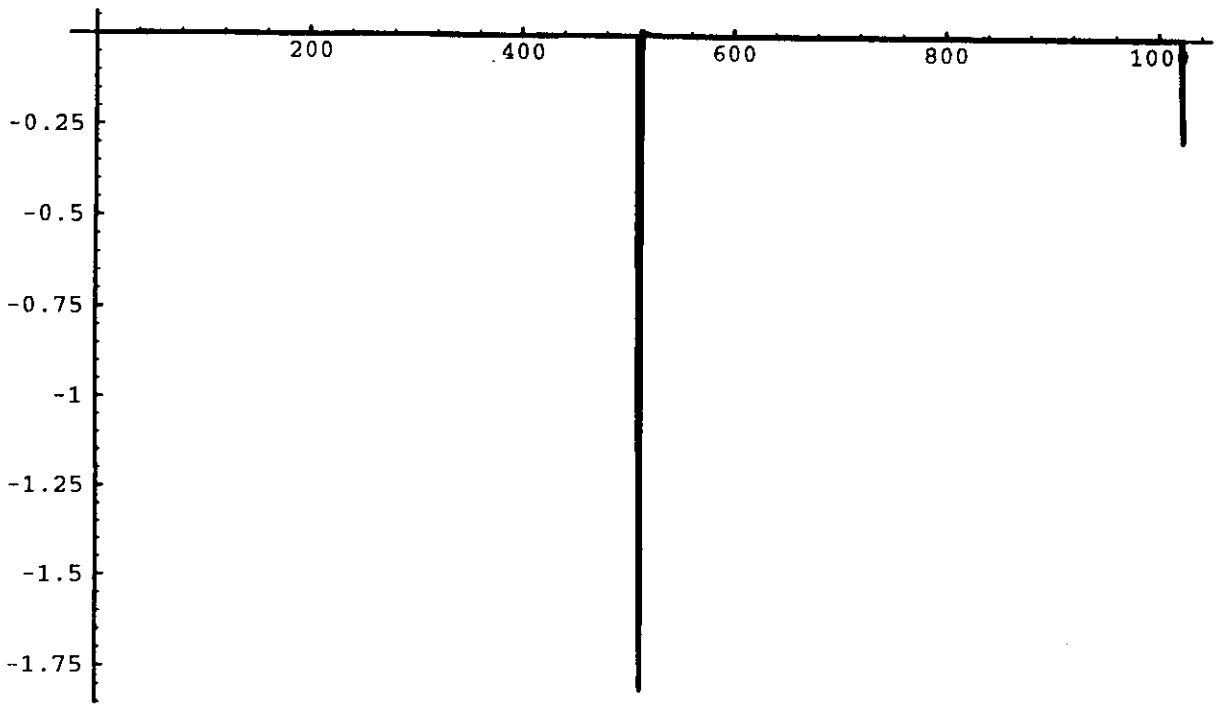
Eg the full inverse is initial = m final = 0]



Fort 10 - original Function



Part. II - Smoothed one


200
400
600
800
1000
-0.25
-0.5
-0.75
-1
-1.25
-1.5
-1.75

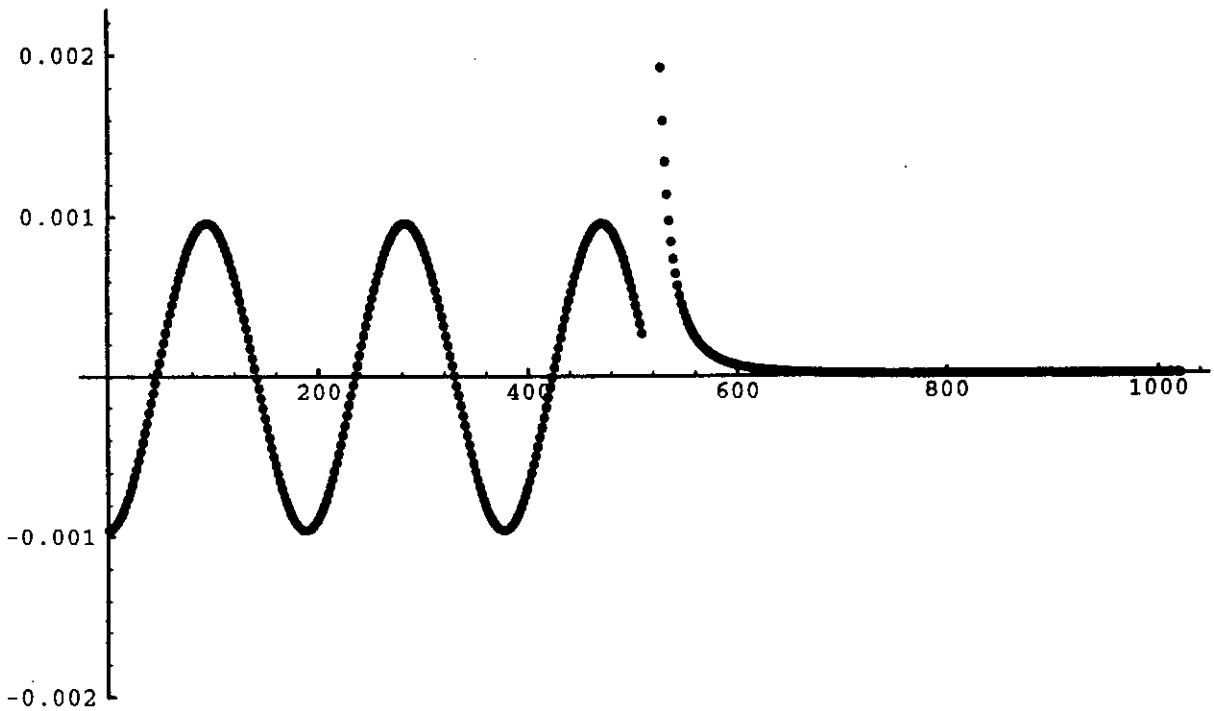
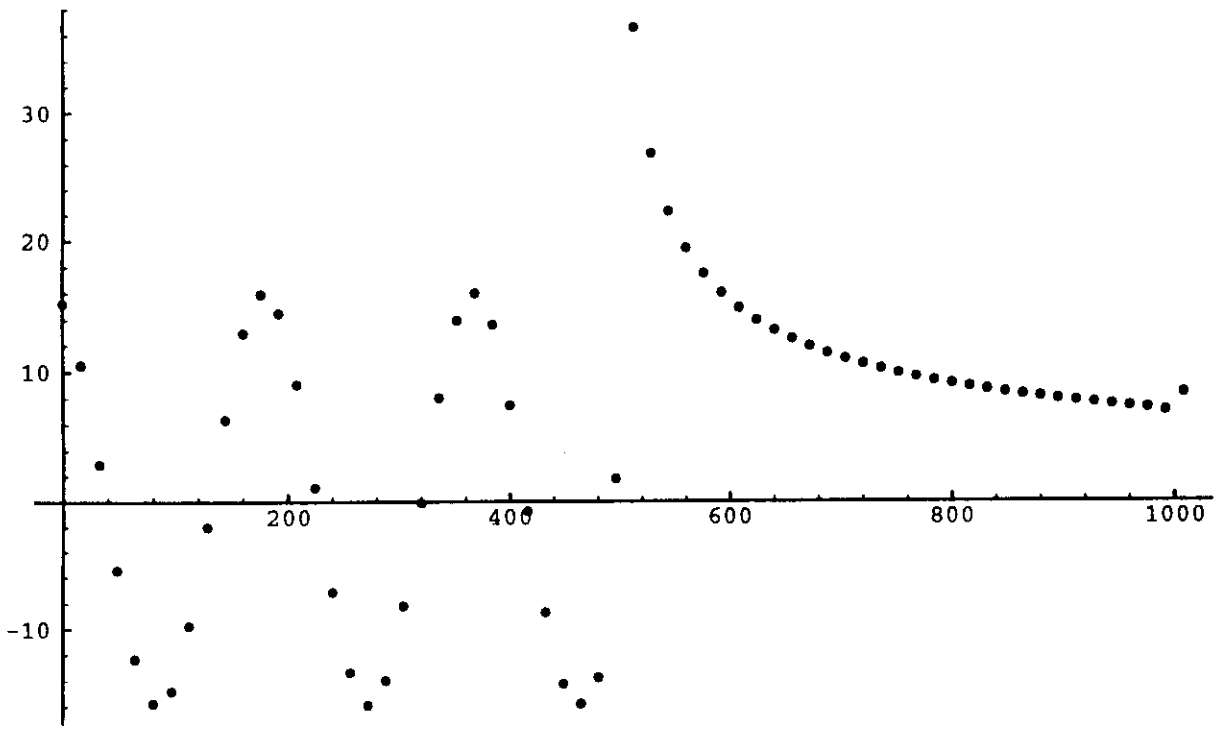


Fig. 5.1 Wavelet from code



Function scaled 4 times

Fig. 14

$1024/2^4 = 64$ data points

This directory contains example wavelet programs
and sample outputs.

The subroutine 'wavelet.f90' is a Daubechies wavelet
transform.

It is called by program 'driver.f90'

To compile and run just do:

```
f90 driver wavelet
a.out
```

At the screen you see the sum rule check, that the
sum of the initial data = the f(0) coefficient of the last transform
level.

The files fort.10, fort.11,... are produced.
fort.10 contains the original function
fort.11 contains the function scaled once
fort.12 contains the functin scaled twice,...
Note that each file has half the data points of the previous one,
and the last file has one data point.

The wavelet coefficients at each level of scaling are written
in fort.51, fort.52 etc
fort.51 contains the level 1 wavelet coefficients
fort.52 contains level 2
and so on ...

Finally the program runs the inverse wavelet transform
to reconstruct f from the scaled function and wavelet data.
The reconstructed f is written out in fort.9,
you can check it is identical to the original function in
fort.10, except from very small rounding errors.

Possible exercises for the student include:

1. Modify the function f(i) and examine the
   scaled function and wavelet coefficients.
   Smooth functions will have small wavelet coefficients,
   while functions with discontinuities or singularities
   will have large wavelet coefficients near the singularity.

2. Modify the Daubechies wavelet to the Haar wavelet. To
   do this just change some commented lines in 'wavelet.f90'.
   Compare the Daubechies and Haar coefficients for various
   functions.

3. Generate the orthonormal 'wavelet' basis functions.
   E.g. take an initial vector f(0)=1 f(i)=0 otherwise.
   Run the INVERSE transform, initial=m final=0.
   The resulting function is the 'scaling functions'
   or 'father wavelet'.
   Now try f(n/2)=1 f(i)=0 otherwise.
   Again run the inverse transform inital=m final=0,
   now your function is the 'mother wavelet'.
   Notice that father and mother are continuous functions
   but not very smooth. Try the same for the Haar wavelet
   are they continous?

4. Try the inverse transform for various initial
   unit vectors, f(i)=1 for some i, f zero otherwise.
   Each time you will get a scaled version of the mother
   wavelet in various positions.
   Try guessing the scale and position from the i value.

Enjoy!

```
! driver for subroutine wavelet

program driver

   implicit none
   integer, parameter :: m = 10 , n = 2**m                ! length of data array
   integer, parameter :: r8 = selected_real_kind(9,99)
   real(r8)           :: f(0:n-1)

   integer            :: initial,final

   integer            :: i,l,d
   real(r8)           :: total


! define any function you like here ------------------------------

   do i=0,n-1
     if (i < n/2)  f(i) = cos( real(i)/30.)              ! smooth bit
     if (i >= n/2)  f(i) = 10./sqrt(10+i-n/2.)     ! singularity close to n/2
   end do

   total=sum(f)                                        ! also sum the data

!-------------------------------------------------------------------------

    write (10,'(i4,e14.6)') (i,f(i),i=0,n-1)
                           ! put the data in a file for plotting later

!-------------------------------------------------------------------------
! transform level by level, storing the 'coarsened' functions  in fort.11,
! for.12 and so on.
! store the wavelet coefficients in fort.51,fort.52 etc

   do initial=0,m-1
      final=initial+1

      call wavelet(n,f,initial,final)

      l=final
      d=2**(l-1)
      write (10+l,'(i4,e14.6)')  (i,f(i),i=0,n-2*d,2*d)
                                                         ! scaled function
      write (50+l,'(i4,e14.6)')  (i,f(i),i=d,n-d,2*d)
                                                         ! wavelet coefficients
   end do


!-------------------------------------------------------------------------


! now check the sum rule:

   print '('' initial sum='',e14.6)',total
   print '('' at level'',i4,'' f(0)='',e14.6)',final,f(0)


! now invert the whole transform

   initial=m
   final=0
   call wavelet(n,f,initial,final)


   write (9,'(i4,e14.6)') (i,f(i),i=0,n-1)
                                  ! write it out to fort.9

                                  ! plot it to check that it is identical
```



```
                                  !to the original apart from rounding error


end program driver
```

```
!-------------------------------------------------------------------------
!
!   wavelet.f90   a FORTRAN 90 subroutine to implement the
!   discrete wavelet transform using the Daubechies-2 wavelets.
!
!   written by James F. Annett for the College on Computational Physics
!   ICTP, Trieste, May-June 1995.
!-------------------------------------------------------------------------
!
!   The subroutine applies either the forwrd of backward
!   discrete wavelet transform of a one dimensional array of
!   data points. It iteratively maps the data from one scale,
!   or resoloution level, to another.
!
!
!   Forward transform:
!
!      The subroutine acts on a one dimensional data array f(0:n-1)
!      For simplicity, n is restricted to powers of 2.
!
!
!      The wavelet transform maps the data array from initial resoloution
!      level 'initial' to final resoloution level 'final'.
!
!      If f() is an untransformed real-space vector, then the inital level
!      must be set to  0
!      If f(0) is the output from a previous call to 'wavelet' then the
!      initial level  should be set at the previous final resolution level.
!      The final_level is set to the desired resolution level of the output.
!
!      In normal use one would set, initial=0 and final=m, to fully
!      convert a data vector of n=2^m elements into its wavelet components.
!      The reverse transformation reconstructs f from the wavelets
!      and is achieved by setting the resolutions : inital=m, final=0.
!      Use intermediate values final to examine the paritally transformed
!      data at any desired resolution.
!
!
!      Forward transfomations:
!
!                  inital < final
!
!      The subroutine recursively applies the transformation
!           f(i)  =  u1 f(i) + u2 f(i+d) + u3 f(i+2d) + u4 f(i+3d)
!      where d=2^(l-1) and i = 0, 2d, 4d, 6d,...
!      This coarse grains  the data from level l to level l+1.
!      (N.B. the data array is considered wrapped around with periodic
!      boundary conditions so f(i) == f( mod(i,n)) in the above expressions.)
!
!      The 'wavelet' or difference coefficients are:
!           h(i+d)  =  v1 f(i) + v2 f(i+d) + v3 f(i+2d) + v4 f(i+3d)
!      Since the transform goes on 'in place' in vector f
!      h(i+d) is stored in position i+d in the array, where again
!      d=2^(l-1) is the spacing of level l, and i = 0, 2d, 4d, 6d,...
!
!      For example, if an array f of length 2^m is transformed from
!      initial level 0 to final level m
!      the array f has the elements
!
! array
! index:   0     1     2     3     4     5     6     7     8     9     ...
! level 1:       h(1)        h(3)        h(5)        h(7)        h(9)  ...
! level 2:             h(2)                    h(6)                    ...
! level 3:                   h(4)                                      ...
! level 4:                                                 h(8)        ...
!                     ......
! level m: f(0)
!
!  (I.e. the level l wavelet coefficients are the array section
!               f(d:n-d:2d)  where  d=2^(l-1)
```



```
!   FORTRAN 90 type syntax)
!
!
!
!  The transformation is normalised so that
!       f(0) at level m  = sum_i  f(i) at level 0
!  i.e. the final coarse grained function value is just the sum of the original data.
!
!
!
!  Reverse transform:
!
!  Call wavelet with
!                 final <intitial
!
!  The subroutine then inverts  the above mapping.  It reconstructs the full
!  resolution of f  at level 'final' using the  data for f on the coarser
!  scales 'inital'  and the 'wavelet' differences h.
!
!
!
!-------------------------------------------------------------------------


subroutine wavelet(n,f,initial,final)
  implicit none
  integer,  intent(in)    :: n                                ! number of data points
  integer,  parameter     :: r8 = selected_real_kind(9,99)    ! double precision
  real(r8), intent(inout) :: f(0:n-1)                         ! data points
  integer,  intent(in)    :: initial,final                    ! resolution levels


!-------------------------------------------------------------------------
! local variables:

  integer                 :: d,l,m,nl
  real(r8), dimension(:), allocatable   :: fcopy

!-------------------------------------------------------------------------
! wavelet coefficients:

  real(r8),parameter      :: u1 = 0.68301270189221932338d0, &  ! (1+sqrt(3))/4
                             u2 = 1.18301270189221932338d0, &  ! (3+sqrt(3))/4
                             u3 = 1.d0-u1,                  &
                             u4 = 1.d0-u2,                  &
                             v1 = -u4,                      &
                             v2 =  u3,                      &
                             v3 = -u2,                      &
                             v4 =  u1

                             ! the u's are  the magic Daubechies scaling function phi2
                             ! the v's the Daubechies wavelet psi2

     !---------------------------------------------------------
     ! for the Haar wavelet just replace the above with
     ! u1=1 ; u2=1 ; u3 = 0 ; u4 = 0
     ! v1=1 ; v2=-1; v3 = 0 ; v4 = 0
     !---------------------------------------------------------


!-------------------------------------------------------------
!  Begin:


!  First check n is a power of 2 equal to 2^m

     m=nint(log(real(n))/log(2.))
     if (n != 2**m ) stop ' Error in subroutine wavelet: n must be a power of 2'
```

```
! Next check that the initial and final levels make sense

     check:  if (initial <= m .and.  final <= m .and.
                      initial >= 0 .and.  final >= 0 ) then



!-------------------------------------------------------------------------
!  Now we can begin the main calculation....



! FORWARD TRANSFORM ----------------------------------
!    (final  > initial)

                          !  first  allocate working vector
      if (final > initial) &
         &   allocate ( fcopy(0:n/2**initial+1) )

      transform:    do l=initial,final-1

                    d=2**l                          ! d is the 'stride' at this level
                    nl=n/d                          ! nl is the number of data points
                                                    ! at this level

                    fcopy(0:nl-1) = f(0:n-d:d)      ! copy data to work array
                    fcopy(nl)     = f(0)            ! wrapping periodically
                    fcopy(nl+1)   = f(d)


                    ! the scaled or coarsened f

                    f(0:n-2*d:2*d)  = u1 * fcopy(0:nl-2:2) +  &
                                      u2 * fcopy(1:nl-1:2) +  &
                                      u3 * fcopy(2:nl+0:2) +  &
                                      u4 * fcopy(3:nl+1:2)

                    ! the wavelet coefficients  or differences, h

                    f(d:n-d:2*d)  = v1 * fcopy(0:nl-2:2) +  &
                                    v2 * fcopy(1:nl-1:2) +  &
                                    v3 * fcopy(2:nl+0:2) +  &
                                    v4 * fcopy(3:nl+1:2)

                    ! the above two lines are equivalent to matrix multiplication
                    !
                    !     (u1 u2 u3 u4                          )
                    !     (v1 v2 v3 v4                          )
                    !     (      u1 u2 u3 u4                    )
                    !     (      v1 v2 v3 v4                    )
                    ! f = (            u1 u2 u3 u4              ) * fcopy
                    !     (            ....                     )
                    !     (                        v1 v2 v3 v4)
                    !     (u3 u4                        u1 u2)
                    !     (v3 v4                        v1 v2)
                    ! with stride d


                    end do transform

   !---------------- done transform ! ----------------------------------


   ! INVERSE TRANSFORM ----------------------------------
   !  (final  < initial)
```



```
                          !  first  allocate working vector
      if (final < initial) &
           allocate ( fcopy(-2:n/2**final-1) )

      inverse:    do l=initial,final+1,-1

                    d=2**(l-1)                      ! d is the 'stride' at this level
                    nl=n/d                          ! nl is the number of data points
                                                    ! at this level

                    fcopy(0:nl-1) = f(0:n-d:d)      ! copy data to work array
                    fcopy(-1) = f(n-d)              ! wrapping periodically
                    fcopy(-2) = f(n-2*d)


                    f(0:n-2*d:2*d)  = u3 * fcopy(-2:nl-4:2) +  &
                                      v3 * fcopy(-1:nl-3:2) +  &
                                      u1 * fcopy( 0:nl-2:2) +  &
                                      v1 * fcopy( 1:nl-1:2)

                    f(d:n-d:2*d)  =   u4 * fcopy(-2:nl-4:2) +  &
                                      v4 * fcopy(-1:nl-3:2) +  &
                                      u2 * fcopy(0:nl-2:2)  +  &
                                      v2 * fcopy(1:nl-1:2)

                    ! the above two lines are equivalent to matrix multiplication
                    !
                    !     (u1 v1                          u3 v3)
                    !     (u2 v2                          u4 v4)
                    !     (u3 v3 u1 v1                       )
                    !     (u4 v4 u2 v2                       )
                    !     (      u3 v3 u1 v1                 )
                    ! f=  (      u4 v4 u2 v2                 ) * fcopy
                    !     (            ......                )
                    !     (                        u2 v2     )
                    !     (                        u3 v3 u1 v1)
                    !     (                        u4 v4 u2 v2)
                    ! with stride d.
                    ! This is the transpose matrix of the forward transform.

                    ! We must scale by 0.5 to normalize the inverse

                      f(0:n-d:d)  = 0.5d0*f(0:n-d:d)

                   end do inverse

          !-------- done inverse transform!--------------------



!-----------------------------------------------------------------
! Tidy up

      deallocate (fcopy)

end if  check

end subroutine wavelet
```

Lecture 3

3.1 Wavelets

We are now in a position to derive the wavelets themselves!

For the Daubechies wavelets this is a two step process.

First we do it for the discrete wavelet transform.

Next we must take a continuum limit to get functions $\psi(x)$ defined for all x.

3.2 First the discrete case

we have a 1-1 linear mapping from our N data points $f_0(0), \ldots f_0(N-1)$ to the "smoothed" function values

f_0

$\swarrow \quad \searrow$

$f_1 \quad h_1$

$\downarrow \quad \searrow$

$f_2 \quad h_2$

$\downarrow \quad \searrow$

$f_3 \quad h_3$

going down to some level L

we have level 1 wavelet coefficients $h_1(1), h_1(3), h_1(5), \ldots$

level 2 wavelet " $h_2(2), h_2(6), \ldots$

3 wavelets " $h_3(4), \ldots$

level L wavelets $h_L(d), h_L(3d), \ldots \quad d = 2^{L-1}$

and the remaining level L function values $f_L(0), f_L(2d), \ldots$

As already discussed

$$N = N_1 + N_2 + N_3 + \ldots N_{2^L} + N_{2^L}$$

$$(h_1)\ (h_2)\ (h_3)\ \ldots\ (h_L)\ \ (f_L)$$

so the set of values $h_1, h_2, h_3, \ldots h_L$ and f_L contain all the information of the original function.

But our transform is also linear, so we can write:-

$$\vec{F}_0 = \sum \vec{\psi}_{1m} h_1(m) + \sum \vec{\psi}_{2m} h_2(m) + \sum \vec{\psi}_{3m} h_3(m) + \ldots + \sum \vec{\psi}_{Lm} h_L(m) + \sum \vec{\phi}_{Lm} f_L(m)$$

regarding the set of N f_0 values as a vector.
[N.B L can be any level less than the maximum (M) ($N = 2^M$)]

We thus have a set of N basis vectors $\vec{\psi}_{1m}, \vec{\psi}_{2m}, \ldots$ $\vec{\psi}_{Lm}$ and $\vec{\phi}_{Lm}$

You can examine these basis vectors quite easily using wavelet.f90. Simply make a vector of data such that it is all zero except for one point which is unity. Now run the inverse transform to level 0. Your data array will be filled with a basis vector. Which one depends on your initial point and level.

To be specific, take $N = 1024 = 2^{10}$
and the initial level $L = 8$.

At this level we have 4 values $f_8(0), f_8(2d), f_8(4d), f_8(6d)$

where $d = 2^{L-1} = 128$

Set the whole array to zero, except element $2d = 256$
which is set to 1.0.
Call wavelet(n,f,8,0) and examine f.

You will see the basis vector $\vec{\phi}_{8,256}$

It should look like this

IF you set one of the other f_8 values to unity
you would see the same, except translated by 256 units,
and possibly wrapped around periodically.

If you had chosen level $L=7$ not 8,
(say set $f_7(256)$ to 1.0, call wavelet(n,f,7,0))
you would see the same thing only half as wide!

This function is the Daubechies "scaling function"
or father wavelet $\phi(x)$ sampled on a discrete set of points.

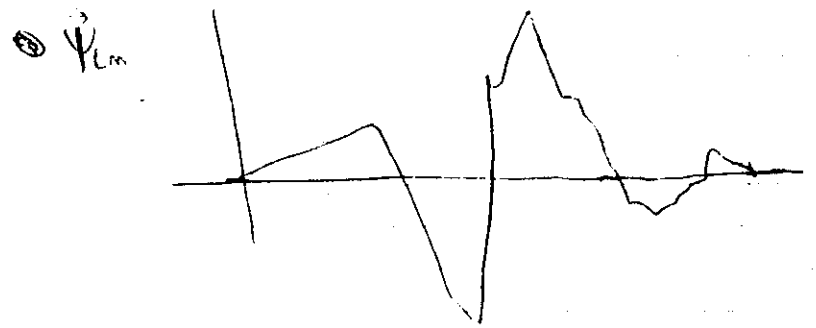
$\phi(x)$ is technically not a wavelet, but any wavelet
expansion also has such functions.

[The father is not a wavelet himself, but generates daughters!]

Now lets look at the ψ_{lm} basis vectors:

again take $N = 1024 = 2^{10}$.
Lets set $l = 8$, so we have wavelet coefficients $h_1(d), h_2(2d), \ldots$ $d = 2^{l-1}$
again there are four $h_8(128)$ $h_8(384)$, $h_8(640)$, $h_8(896)$
set one of these, say $h_8(384)$ [position 384 in array] to 1.0
Call wavelet(n,f,8,0).

Now your data will be:



This is the "mother wavelet", (or one of her daughters - daughters are identical to the mother except in size!)

Again changing from $h_8(384)$ to $h_8(640)$ just translates the same wavelet by 256;
and changing from level 8 to 7 (say $h_7(64) = 1.0$) makes a half-size wavelet.

[You can make them from level 7 or 10 if you like, but they will look different because they wrap around the periodic boundaries]

[If make them from level 1 or 2 and you will see just discrete points]

3.3 Orthonormality

To recap, our wavelet transform has analysed our initial vector $\vec{f}_0$ into a sum of wavelet coefficients

$$\vec{f} = \sum_m \vec{\psi}_1(m)\, h_1(m)$$
$$+ \sum_m \vec{\psi}_2(m)\, h_2(m)$$
$$+ \sum_m \vec{\psi}_3(m)\, h_3(m)$$
$$\vdots$$
$$+ \sum_m \vec{\psi}_L(m)\, h_L(m)$$
$$+ \sum_m \vec{\phi}_L(m)\, f_L(m)$$

The $\vec{\psi}$ and $\vec{\phi}$ are thus a complete (framed in finite sizes) set of basis vectors.

They are also almost orthonormal

$$2^L\ \vec{\psi}_L(m) \cdot \vec{\psi}_L(m') = \delta_{mm'}$$

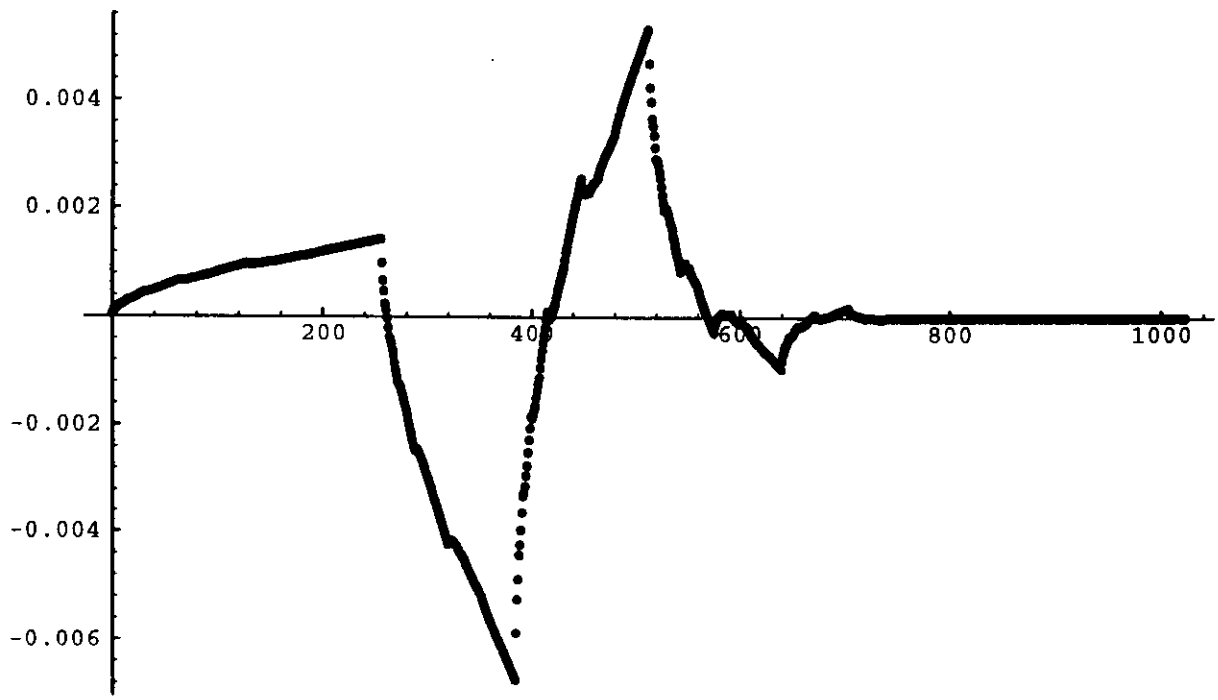
$$\vec{\psi}_L(m) \cdot \vec{\psi}_{L'}(m') = 0 \qquad L \neq L'$$

$$\vec{\psi}_L(m) \cdot \vec{\phi}_{L'}(m') = 0 \qquad \text{any } L, L', m, m'$$

$$2^L\ \vec{\phi}_L(m) \cdot \vec{\phi}_L(m') = \delta_{mm'}$$

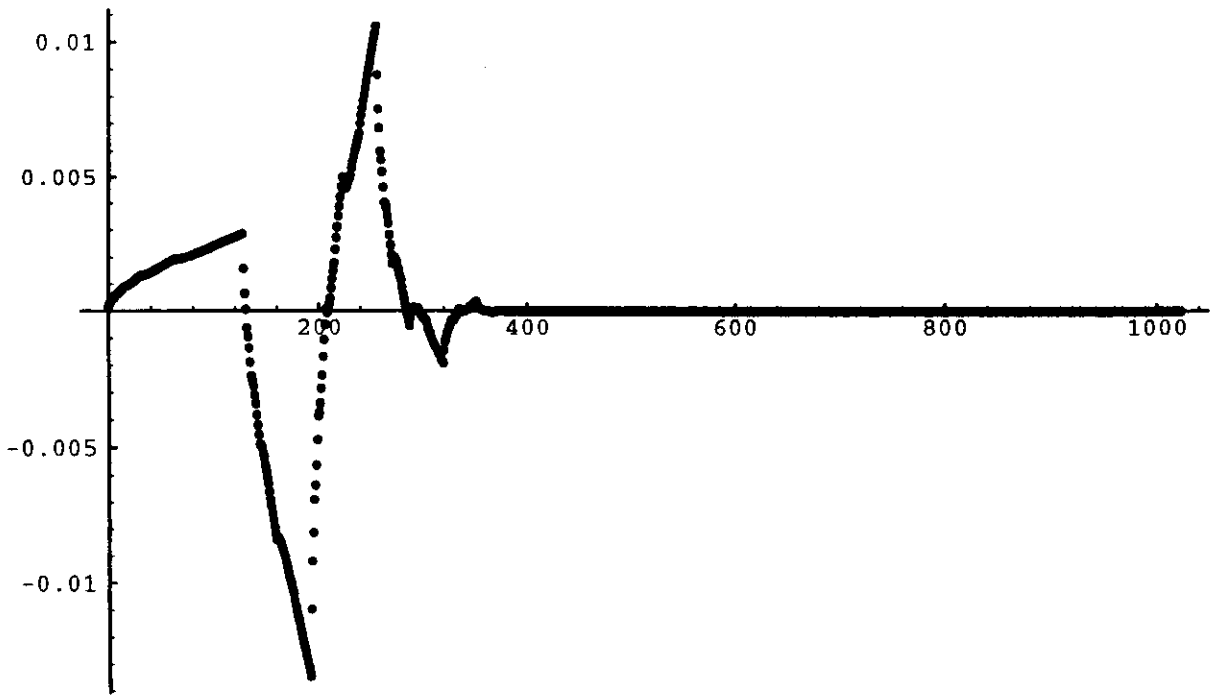
i.e. they are orthogonal under translation, and the wavelets $\vec{\psi}$ are also orthogonal under scaling.

(Note we only ever have one scale of $\vec{\phi}_L$ in the expansion)



Daubechies $\Psi(x)$, 2^{-8}

actually, $\vec{\psi}_{i,0}$ $i=8$ the discrete version

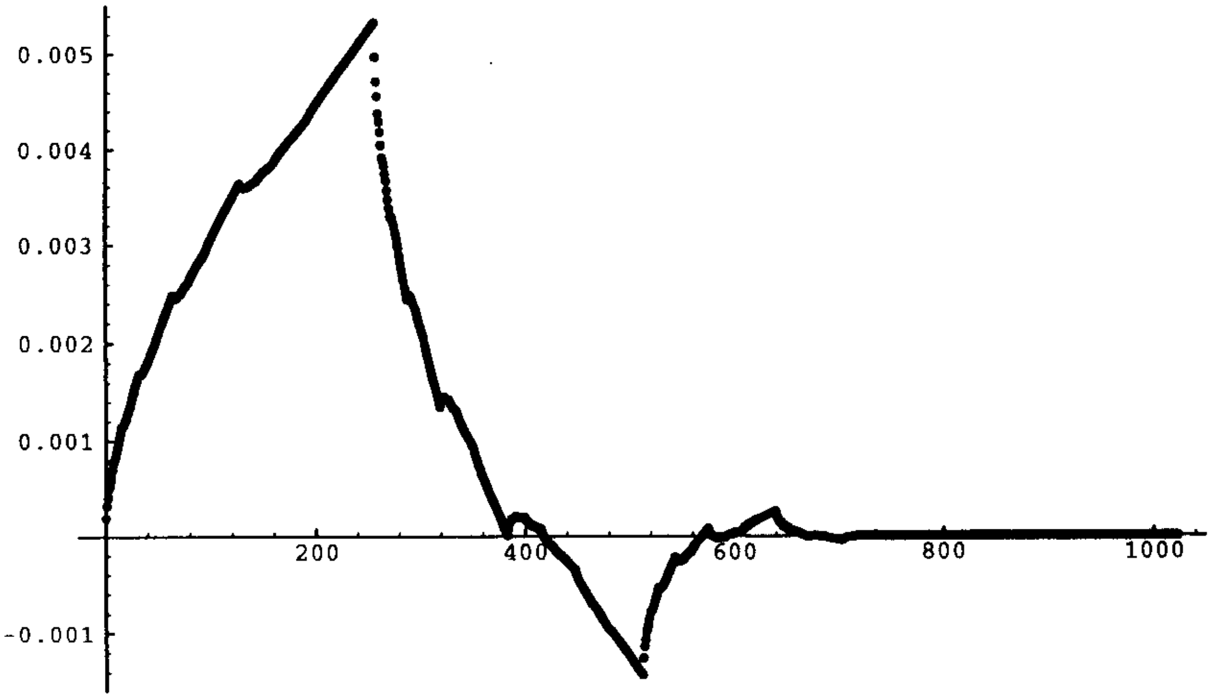


Daubechies $\psi(x)$ scaled once

ie $\psi(2x) \times 2^{-1}$

actually this is the discrete version $\vec{\psi}_{10}$ (=)

a "daughter" of the mother wavelet



Daubechies $\Phi(x) \times 2^{-3}$

scaling function or "father wavelet"

Again on the discrete lattice it is $\vec{\Phi}_{lm}$
$l = 8$

The factor of 2^ℓ arises because the wavelet transform matrix

$$\begin{pmatrix} u_1 & u_2 & u_3 & u_4 & & \\ v_1 & v_2 & v_3 & v_4 & & \\ & & u_1 & u_2 & & \\ & & & \ddots & & \end{pmatrix}$$

was not unitary but $\sqrt{2}$ x a unitary matrix. At ℓ levels and squaring, a 2^ℓ factor is needed.

[just like e^{ikx} is not orthonormal $\frac{1}{\sqrt{N}} e^{ikx}$ are!]

This almost completes our derivation of Daubechies wavelets. We have a set of functions orthogonal on different scales. However it remains to take a continuum limit so for the ψ are only defined on discrete lattices!

3.4 Continuum limit

All of the above manipulations follow quite naturally from properties of finite unitary matrices. It is analogous to discrete Fourier transforms.

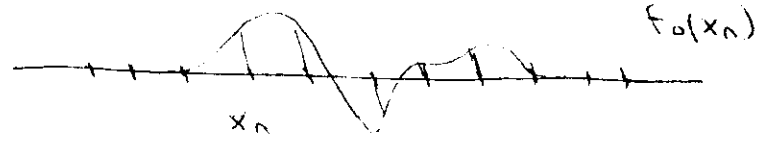
As in the Fourier case, the extension from discrete to continuum must be made very carefully. Details are clearly beyond the scope of these lectures. Daubechies book "Ten Lectures on Wavelets" or original papers (IEEE Trans. Inf. Th. 34, 605 (1988)) present the proofs.

I shall just give a heuristic approach.

First, lets go from a finite sets of points to an infinite number.

Ie we have points $x_n = n$ on the integers say.

We can still do the wavelet transform without any infinite sums provided our function is localised in a finite region.



Up till now we've only gone down in level since there was a natural top and bottom level $0, m$ $(N = 2^m)$ for a finite set of points.

But now there is no limit. We can go up or down!

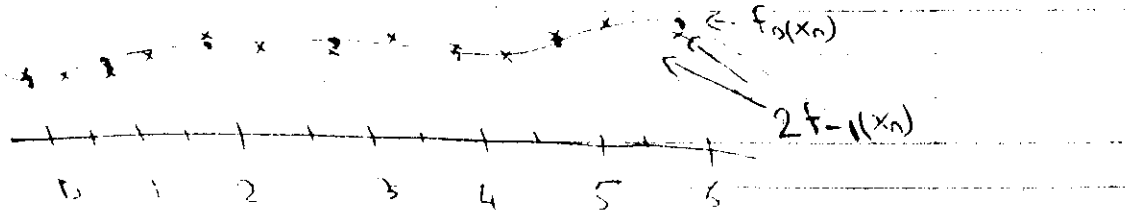
And we can go up or down infinitely many levels!

As before, going down a level replaces $f_0(x_n)$ at integer points with $F_1(x_n)$, $h_1(x_n)$ at points of spacing 2.

What does going up do?

Clearly $F_{-1}(x_n)$ will be defined at the points $\ldots 0, \frac{1}{2}, 1, \frac{3}{2}, \ldots$

We will be filling in points



The inverse transform is

$$\begin{pmatrix} \vdots \\ F_c(-d) \\ F_c(0) \\ F_c(d) \\ F_c(2d) \\ \vdots \end{pmatrix} = \frac{1}{2} \begin{pmatrix} \ddots & & & & & & \\ & u_1 & v_1 & & & & \\ & u_2 & v_2 & & & & \\ & u_3 & v_3 & u_1 & v_1 & & \\ & u_4 & v_4 & u_2 & v_2 & & \\ & & & u_3 & v_3 & u_1 & v_1 \\ & & & u_4 & v_4 & u_2 & v_2 \\ & & & & & u_3 & v_3 \\ & & & & & & \ddots \end{pmatrix} \begin{pmatrix} \vdots \\ h_{c+1}(-d) \\ F_{c+1}(0) \\ h_{c+1}(d) \\ F_{c+1}(2d) \\ \vdots \end{pmatrix}$$

$\uparrow$ index $(-d \;\; 0 \;\; d \;\; 2d$

For an infinite length vector. Here $d = 2^c$

Note that it takes functions on spacing $2d$ to one on spacing d

So we can start with any ~~fn~~ $F_0(x_n)$ on the right side.

where $x_n = n$

and map to $F_{-1}(x_n)$ $x_n = n/2$

map to $F_{-2}(x_n)$ $x_n = n/4$ and so on. ...

Obviously we take $h_{c+1} = 0$ in each of these mappings - we are trying to find the fine scale F which corresponds exactly to our coarse scale one.

The key question is, If we start with a function $F_0(x_n)$ defined on the integers, with it having a well defined behavior on smaller and smaller grids - can we define its continuum limit? (Ie an interpolation...).

Also do our basis vectors Φ_{lm}, Ψ_{lm} have well defined continuum limits?

To both of these questions have answers - YES!

First of all the continuum limit of any function :-

Pick any point $x_n = nd$ in f_ℓ

$$f_\ell(x_n) = \frac{1}{2}\left(u_3 f_{\ell+1}(x_n - 2d) + u_1 f_{\ell+1}(x_n)\right)$$

from the vector and since $h_{\ell+1} = 0$.

Now, once the grid gets very fine, and assuming f_ℓ becomes continuous we get

$$f_\ell(x_n) \approx \frac{1}{2}(u_3 + u_1) f_{\ell+1}(x_n)$$

$$= \frac{1}{2} f_{\ell+1}(x_n) \quad \text{since } u_3 + u_1 = 0$$

So apart from a scale F starts to become independent of ℓ as we keep on iterating to finer and finer scales.

Ie. $\lim_{\ell \to -\infty} 2^{\frac{1}{2}\ell} f_\ell(x) \to F(x)$ exists

defining $F(x)$ for any point of the form $n/2^\ell$ (a dense set)

(This quasi proof can be made rigorous property, see Daubechies work)

Now for the basis functions.

Start with $f_0(0) = 1$ $f_0(n) = 0$ $n \neq 0$

Iterate to finer scales $2^\ell f_\ell(x) \to \phi(x)$ defining $\phi(x)$ ✓

For the mother wavelet start with $h_n(\frac{1}{2}) = 1$ all other $f_0, h_n = 0$

and work backwards to $f_{-1}, f_{-2}, \ldots$ again $2^\ell f_\ell(x) \to \psi(x)$ as $\ell \to -\infty$

These limits define $\phi(x), \psi(x)$.

Again we are iteratively defining the functions on a dense, but discrete set of points $n/2^\ell$. Provided ϕ, ψ etc continuous (proof hard) I can get arbitrarily accurate ψ, ϕ at any point by looking at close enough points.

Because of the properties already derived for the discrete basis functions ϕ, ψ, when we make the continuum limit we have the orthonormality relations:

$$\int \phi(x-m)\,\phi(x-n)\,dx = \begin{cases} 0 & n \neq m \\ 1 & n = m \end{cases}$$

$$\int \phi(x-m)\,\psi(x-n)\,dx = 0$$

$$\int \phi(x-m)\,\psi(2^l x - n)\,dx = 0$$

$$\int \psi(x-m)\,\psi(2^l x - n)\,dx = \begin{cases} 0 & \text{if } l \neq 0 \text{ or } n \neq m \\ 1 & \text{if } l = 0 \text{ and } n = m \end{cases}$$

They follow simply by taking the discrete version and letting $\delta x \to 0$ on smaller and smaller grids.

Also $$\int \phi\,\psi(x) = 0$$

$$\int x\,\psi(x) = 0$$

For this ψ we say 'class 1' wavelet.

For higher classes $\int x^p \psi(x) = 0 \qquad 0 \leq p \leq M$

ϕ, ψ vanish outside an interval $(0,3)$ for class 1
larger for class 2 or more

ϕ, ψ are continuous - but not differentiable

Go to higher class to get differentiable functions.

$$F(x) = \sum_{l<L} \psi_{lm}(x)\,(\psi_{lm}, f) + \sum_m \phi_{Lm}(x)\,(\phi_{Lm}, f)$$

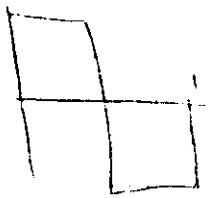
is a wavelet expansion of an arbitrary function $\psi_{lm} = 2^{l/2}\,\psi(2^l x - m)$
$\phi_{lm} = 2^{l/2}\,\phi(2^l x - m)$

3.5 Smoother wavelets

As well as the Daubechies class 1 wavelets we have the Haar system (class 1)

For Haar $\psi_1(x)$

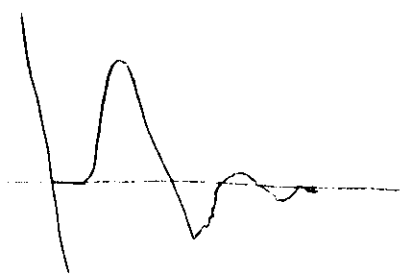
$\psi_1'(x)$



We also have Daubechies class 2, ... wavelets which are smoother

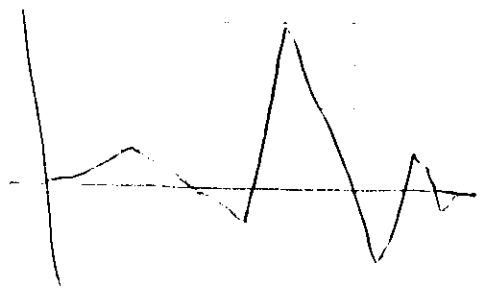
class 2

ϕ



~~class~~

ψ



class 3