Signal Processing
With Wavelets

JAMES MONK

Niels Bohr Institute, University of Copenhagen.
Reminder of the Fourier Transform

\[ g(\omega) = \frac{1}{\sqrt{2\pi}} \int_{-\infty}^{\infty} f(t)e^{-i\omega t} \, dt \]

๏ Tells you the frequency components in a signal

๏ One method of encoding a signal (e.g. a piece of music): take the Fourier transform, keep only those contributions in the frequency domain that are large - this is a (bad!) lossy compression technique.

๏ Remember the uncertainty principle - you need an infinite number of Fourier terms in order to make a sharp spike.

๏ Put another way, if you have a spike in your data with a width approaching zero, you start getting a very large number of populated frequencies.

๏ The Fourier basis functions, sine and cosine, have infinite extent.

๏ The Fourier transform does not tell you when (or where) in your data a particular frequency is occurring. It just tells you what contribution a given frequency makes.

๏ The result of all this is ringing. Although using a Fourier basis can be a good way of encoding some signals, in some situations you get artefacts due to the finite number of terms.
Example of the Square Wave

- Using the first ~20 Fourier components of a step function.
- Note the wiggles - ringing
To get around these limitations, people tried modifying the Fourier basis functions by a moveable Gaussian window.

$$g(\omega, t) = \frac{1}{\sqrt{2\pi}} \int_{-\infty}^{\infty} f(\tau) e^{-i\omega \tau} e^{-\frac{(t-\tau)^2}{\sigma^2}} d\tau$$

Note how the transformed function now depends on frequency and time.

But the width of the Gaussian window is fixed.
Enter Wavelets

Mathematical tool developed in the 1980s and 90s

Grew out of short-time Fourier Transforms, i.e. windowed by a Gaussian (Morlet & Grossman, 1980)

Modern, discrete and orthogonal wavelet basis developed in large part by Ingrid Debauchies (~1988)

Many applications in a wide range of subjects. Deep relevance to the way the natural world appears to work.
Continuous Wavelet Transform

\[ W(S, t) = \frac{1}{\sqrt{S}} \int_{-\infty}^{\infty} f(\tau) \psi \left( \frac{t - \tau}{S} \right) d\tau \]

Yields time-frequency information on a signal
Discrete Wavelet Transform

Coefficient

\[ C_{mn} = \sqrt{\frac{2^{(m-M)}}{S_0}} \int \tilde{\psi} \left( 2^{(m-M)} \frac{\phi}{S_0} - n \right) P(\phi) d\phi \]

Arbitrary scale (limit of resolution is a good choice)

Wavelet function

Index \( m \) identifies the physical scale of the coefficient (c.f. wavelength for Fourier)

Index \( n \) identifies the location (translation) of the contribution

Wavelet coefficients have both scale, and translation (FT has only scale)

\[ \psi_{mn}(\phi) = \sqrt{2^m} \psi \left( 2^m \phi - n \right) \]

The wavelet bases are re-scalings and translations of a (scale-less) mother wavelet
The Haar wavelet is the simplest wavelet, consisting of a step function that takes the difference between adjacent points. After taking the difference, the two points are **averaged**, and the output is a re-scaled version of the signal.

Wavelet Coefficients

Re-apply the wavelet to the re-scaled signal
Imagine a short signal 8 samples long

Take the difference and sum of neighbouring cells

Keeping half of the sums and half of the differences preserves all information

Repeat this on the output of the half of the sums we keep.

The Wavelet Coefficients are the differences we keep.
The **high-frequency**, small scale cell-by-cell changes appear on the **right**. There is a uniform change of 1 at this scale.

**Moderate** scale changes over scales of **two** cells are in the **middle** coefficients.

The **total** sum is on the **leftover** coefficient (36).

Note the **high-frequency** component occupies fully **half** of the information!

This is how a **compression** algorithm works.
The Daubechies family of wavelets are usually more useful - encodes high frequency features better.

Derived recursively by inverse transforming \{1,0,0,...N\}

The “well known” Daubechies 4 wavelet

And the re-scaling function
Wavelets are an example of multi-resolution analysis

Your brain processes vision like this - analysing contrast changes over the local background

Already we can see how this is useful in physics to separate fine details from broad structures

(This is the same image, but when seen from afar your brain uses the large-scale structure, up close it prioritises the fine detail)
Use of Wavelets

The cochlea inside your ear is arranged such that it performs a wavelet transform on sound. Attempts to "sonify" LHC data were doing wavelet analysis!

Wavelets can be used as the basis of a compression algorithm, including JPEG 2000.

The Stock market is (allegedly) fractal, and subject to wavelet analysis.

Astronomers use these techniques for image analysis, extraction of fine details like Einstein rings. Wavelets used to decompose the CMBR.
Benoît B.* Mandlebrot: 
“Clouds are not spheres, mountains are not cones, coastlines are not circles, and bark is not smooth, nor does lightning travel in a straight line.”

*The B. stands for “Benoît B. Mandlebrot”

Building up a structure from repeated re-scalings of the same basic shape is very common in nature - fractal structures.

Wavelets can be a good way of understanding and looking for this self-similarity

If you have come across the scale-dependence of e.g. coupling constants in particle physics ($\alpha_s$), then you might see this is a similar idea.
Gaussian Noise example: wavelet_gaussian.py

- Produces a simple Gaussian and then adds noise
- You can vary the amount of noise by increasing or decreasing the “stats” parameter in the code, and also by changing the random seed
- First performs a continuous wavelet transform - note the separation in frequency between the noise and the signal
- Then perform a discrete transform, filter the coefficients to reduce the noise, and reverse the transform to re-obtain the signal
- At its simplest, the threshold sets to zero any coefficient whose value is below the RMS for its “level”
- Wavelet level = all coefficients having the same scaling parameter
- Try and play around with the filtering, see the effect of different thresholds, removing high and low frequency contributions, using different basis functions.
To generate Gaussian-distributed random numbers for the noise, we need the inverse of the ERF function.

The ERF function is the cumulative distribution function of the Gaussian.

The output of ERF is between 0 and 1.

So when you take the inverse ERF of a uniform random number between 0 and 1, you get a Gaussian distribution.
The second example uses the LIGO data available from here: https://losc.ligo.org/events/GW150914/

LIGO provide their own tutorial using Fourier analysis. You can follow it here: https://losc.ligo.org/s/events/GW150914/GW150914_tutorial.html

We will use the 4096 Hz samples from the Livingston and Hanford detectors. Note they are 6.9 ms apart.

Without signal processing, they do not look anything like the famous chirp sound formed when two black holes collide.

Our goal will be to filter using wavelets to see if we can extract the chirp!
LIGO data

![Advanced LIGO strain data near GW150914](image1)

![Advanced LIGO WHITENED strain data near GW150914](image2)