A wavelet analysis tutorial that even a psychologist could understand

Thomas Gladwin (note in 2012: this is from a pre-PhD student-Thomas of many years ago, who was possibly quite drunk when he wrote this. That's my excuse anyway and I'm sticking with it.)

Preface

Wavelet analysis is a nice new technique applicable to EEG data that makes two things possible: an even larger amount of worthless psychophysiological posturing (but with prettier pictures) and a far deeper insight into the workings of the human brain. The second possibility requires the combination with other theories and techniques that map onto the information provided by wavelet analysis in some productive way. My guess is that wavelet analysis will need to be related to the behaviour of integrate-and-fire neural networks to really help neuroscience forward - advanced measurement techniques just sit around looking depressed on weak, ad hoc theories.

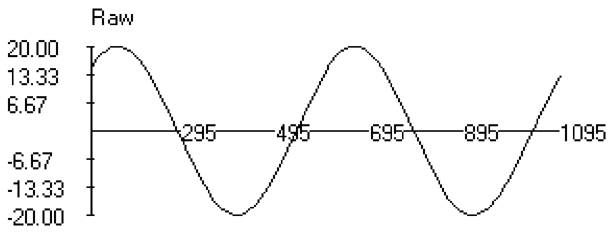
The reason I think it was worthwhile to write this tutorial (actually, it's just a set of sheets I added some text to) is that when I wanted to learn about wavelets it took a long time to gather the necessary information because so much background knowledge was assumed. For me, that wasn't the case at all - the prerequisites for the prerequisites weren't there - and it was hard to find any references explaining the meaning of all the formulae.

I've tried to include, not a set of necessary mathematical tutorials, but hopefully enough information for anyone interested enough to be able to look up the details and for anyone disinterested enough to sail through the text and feel good about his or her astounding level of insight.

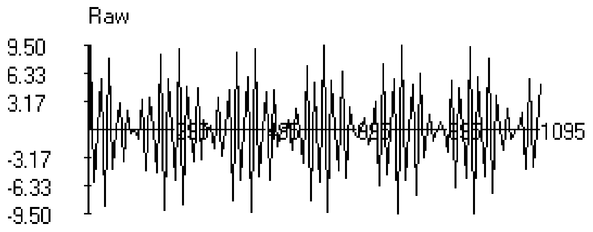
Have fun with the tutorial, and please provide any feedback you feel might be helpful in improving it.

The addition and decomposition of signals

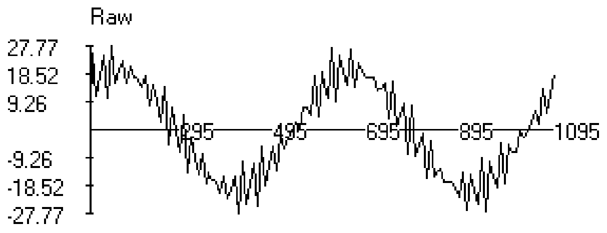
EEG signals may be decomposed into simple component (co-)sine-based signals oscillating at various frequencies (see below for a short introduction to oscillating signals). This principle is most easily understood by reversing the decomposition procedure. In the example below, the very simple cosine Y1(t) and the slightly more complicated cosine Y2(t) are added to produce the composite function Y3(t).



$Y1(t) = A * \cos(2 * \pi * 10 * t)$



$Y2(t) = A(t) * \cos(2 * \pi * 20 * t)$



$Y3(t) = Y1(t) + Y2(t)$

The goal of wavelet analysis is to work backwards from a composite signal to the simple component signals. Wavelet analyses are performed a frequency at a time; in the above example, wavelet analysis of the 20 Hz content of Y3(t) would have provided the characteristics of Y2(t). The goal of this tutorial is to explain what characteristics wavelet analysis reveals about component signals and how this is achieved.

The characteristics of oscillatory signals

The generic cosine function contains three parameters, A, O and P:

$Y(t) = A * \cos(O*t + P)$

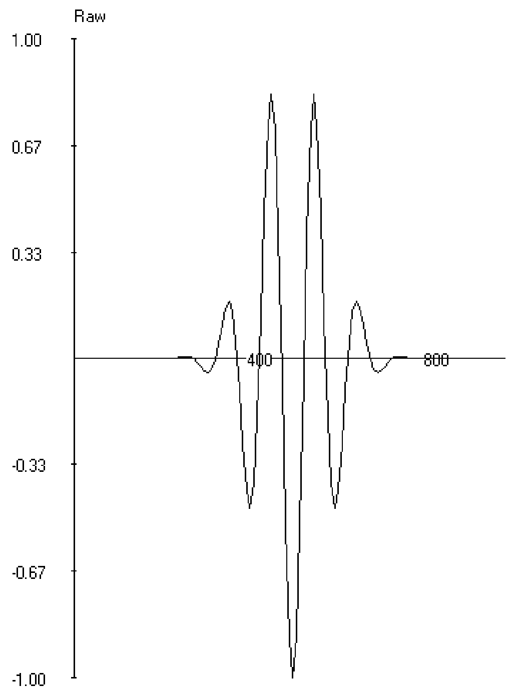
The cosine at the heart of the expression is an oscillatory function. Oscillatory functions fluctuate around a central value (usually zero) like a pendulum swings across its lowest point. The time it takes for the pendulum to swing from one extreme position to the other and back again is called the period. The period is directly related to the frequency which is simple (1 / period). If the period is given in seconds, the frequency is in Hertz (Hz), or oscillations-per-second.

A, the amplitude of the cosine, is analogous to the maximum deviation of such a pendulum. A determines the vertical scaling of Y(t): how large the function’s deviations from zero are. O is the angle speed and determines the horizontal scaling. A larger O causes the function to run through its oscillation more quickly; it makes the pendulum swing faster. O is therefore directly related to the period and frequency of a signal. P, the initial phase, simply shifts the function over time. If P is zero, cos(O*t) is obviously equal to cos(O*t + P). If P is equal to half a period, cos(O*t + P) at time t returns the function value associated with the cosine at time t + a half period.

An important distinction between oscillatory signals is whether they are stationary or non-stationary. The three parameters of stationary signals are constant whereas those of non-stationary signals are themselves functions of time. Wavelet analysis of a composite signal at a specified frequency provides a vector (a vector is a list of numbers where each number is associated with an identifying index-number) of amplitudes and phases for the component functions.

Wavelets

The basis of wavelet analysis is the wavelet function. Wavelets are a type of oscillatory signals that are localized in time, unlike, for example, sines and cosines. A well-known example of a wavelet is the Morlet wavelet:

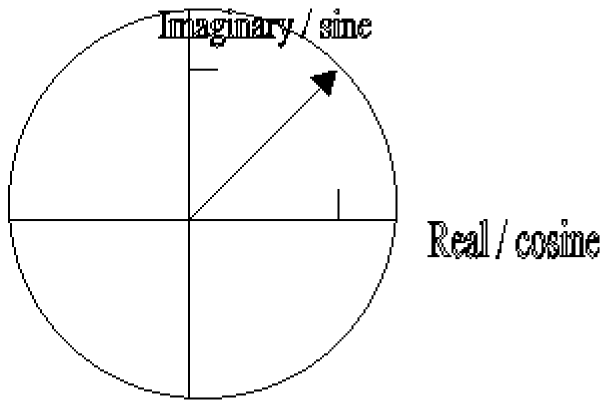


$$W(t) = \exp(-a(t - b)^2) * \cos (\omega(t - b))$$

The wavelet is centered on time b because the amplitude $A(t) = \exp(-a(t - b)^2)$ of the signal becomes smaller at longer distances from the central time ($A(t)$ is never larger than when $(t - b)^2$ is zero).

Real and imaginary axes

Wavelets contain a real and imaginary part. The real part is the function described above, the product of a Gaussian curve and a cosine. The imaginary part is the product of a Guassian curve and a sine. The idea of real and imaginary parts of oscillatory signals can be illustrated via the unit circle representation of a cosine.



The projections onto the real axis are the function values of the cosine.

The angle of the vector is the phase of the signal. The passage of time is represented by the clockwise rotation of the vector. Recall that vectors are a list of numbers. In this case the list has two elements: the real / cosine and imaginary / sine values. The arrow-representation of such a vector is drawn from the (0, 0) point to the (real value, imaginary value) point. The projections onto the axes are thus simply the function values.

The length of the vector projecting into real and imaginary space is the amplitude parameter. Note, for example, that A * cos(0) = A and A * sin(0.5 * pi) = A. As a general proof using Pythagoras' theorem:

$A^2 = (A * \cos(\alpha))^2 + (A * \sin(\alpha))^2$,

> That is, the squared diagonal (the length of the vector which we claim to be A) is equal to the squared sides (the elements of the vector). The right-hand side can be rewritten as follows:

$A^2 * (\cos(\alpha)^2 + \sin(\alpha)^2)$,

> And the term $(\cos(\alpha)^2 + \sin(\alpha)^2)$ is always equal to one (applying Pythagoras' theorem to the unit circle). So we end up with $A^2 = A^2$, which is true, proving that the length of the vector projecting into "cosine / sine" space is the amplitude parameter.

The successive function values of a cosine cos(x) over ascending x are represented by the projections onto the real axis associated with the clockwise rotation of the vector.

The reason the imaginary part is interesting will become clear later on. The rationale of wavelet analysis as presented in this tutorial can be understood by remembering only that the imaginary part of oscillatory signals behaves like the same signal except for a time-shift. The term "imaginary" can then be understood to refer to the fact that all you actually see when you plot a cosine are the values on the real axis; the values on the imaginary axis only come into play when you invoke the rotating-vector representation, for example because that representation makes wavelet analysis easy to explain.

However, there is a better reason that that, explained in the following section. The imaginary axis is actually called so because the values on that axis are so-called imaginary numbers.

Imaginary numbers

There are various kinds of numbers: positive integers (e.g. 1, 32, …), negative values (e.g. -1, -32, …), rational numbers (e.g. ½) and irrational numbers (pi, √2), for example. All these kinds of numbers are classed together as real numbers: they obey the fundamental laws of arithmetic. These laws, listed below, determine how numbers behave when mathematical operators are applied to them:

The **c**ommutative and **a**ssociative laws of **a**ddition and **m**ultiplication.

$a + b = b + a$ ‘**ca**’ $(a + b) + c = a + (b + c)$ ‘**aa**’

$a * b = b * a$ ‘**cm**’ $(a * b) * c = a * (b * c)$ ‘**am**’

The distributive law of multiplication and division over addition and subtraction

e.g. $(a + b) * c = (a * c) + (b * c)$

The powering laws below follow from the fundamental laws.

$a^n * a^m = a^{n+m}$

$a^n / a^m = a^{n-m}$

$(a^n)^m = a^{n*m}$

One might wonder why numbers should have to obey the fundamental laws of arithmetic, regardless of whether the objects we usually measure with those numbers do tend to be so subservient. In fact, not all numbers do. Examples of non-real numbers are the imaginary and complex numbers.

Consider the number *j* defined as

$j = \sqrt{(-1)}$

This number doesn’t follow the Rules. Consider squaring *j*:

$j^2 = j * j$

If *j* is real, let’s refer to this hypothetical real *j* as *j~*, it is either negative or non-negative (zero or positive). At the moment, we don’t know which of the two *j~* is, but that doesn’t matter. If *j~* is non-negative then

$j\sim * j\sim >= 0$

whereas if $j\sim$ is negative then

$j\sim * j\sim = -1 * |j\sim| * -1 * |j\sim| = -1 * -1 * |j\sim| * |j\sim| = 1 * |j\sim| * |j\sim| >= 0$

So the Rules imply that squaring any real number $j\sim$ always results in a positive result. However, squaring $j = \sqrt{}-1$ results, by definition and formally from the third powering rule $(a^n)^m = a^{n*m}$, in a negative number, namely -1. The number *j* thus disobeys the Rules and disqualifies itself from the membership of the real numbers.

Imaginary numbers are numbers that are defined in such a way that they do not obey the fundamental rules of arithmetic, *j* being the prime example. *j* is actually the only imaginary number we need, as any imaginary number can be written as j * R, where R is a real number. For instance, $\sqrt{}-9 = \sqrt{}-1 * 3 = j * 3$.

While *j* is an imaginary number, j^2 is not; j^2 is simply the negative number –1, so there's no problem in working with that. Squaring *j* thus brings it back into the domain of real numbers. So, even though *j* is imaginary, it seems that it is not therefore ineffable. In contrast, imaginary numbers are no more mystical than real numbers and no less able to describe quite natural relations. The following example of this was taken from Isaac Asimov.

Imagine you're standing on the zero point of a line on which all the positive and negative numbers are represented and are looking out over the positive numbers, focussing, say, on 42. Multiplying by –1 has the effect of your turning around, 180 degrees, and now you're looking at –42. Multiplying by –1 again has you, as would be expected, completely turned around so you're looking at 42 again.

From your regained starting point, now multiply by *j*. Asimov's analogy was that you're now turned around a quarter instead of a half circle. So multiplying by *j* again would turn you onwards to half a circle away from your starting position, and this is indeed the case: $j * j = -1$. So this very simple analogy of what imaginary numbers do seems to work.

The analogy can be further strengthened by multiplying by *j*, then by –1, and then by *j* again, which is equivalent to multiplying by –1 twice, doing nothing, or multiplying by *j* four times. So the mysterious *j* actually has a quite natural relation to the real numbers, namely a perpendicular one; that is, if, as above, + and – have a reversed relation of 180 degrees, *j* has a relation of 90 degrees.

Now we've seen that numbers such as *j*, or numbers that are multiples of *j*, are imaginary and perpendicular to the real numbers, our new family of number can easily be expanded to include complex numbers. Complex numbers are simply the sum of a real and an imaginary number:

Complex-value = Real-value + *j* * Imaginary-value

A number that is the sum of two real numbers would fall somewhere on the line that contains the real numbers; likewise, a sum of imaginary numbers would fall on the imaginary axis. In contrast, a complex number can fall anywhere on the plane defined by the real and imaginary axis.

A complex number is really a vector of two numbers, pointing to somewhere on the real – imaginary plane from the origin. Such a vector is defined by its elements, but also by the length of the vector and its angle. The elements of a complex number can be written using the cosine and sine of the triangle defined by the vector's length and angle:

R = length * cos angle

I = length * sin angle

This leads to the polar form of a complex number:

Complex-value

= R + *j* * I

= length * cos angle + *j* * length * sin angle

= length * (cos angle + *j* * sin angle)

Now the relation between the cosine function and its associated imaginary sine turns out to be very simple. If the real values R(x) of a function f(x) follow a cosine, then:

R(x)

= length * cos angle

= cos x

So length = 1 and angle = x. From this and the perpendicularity of the imaginary axis which led to the expression for the imaginary value I:

I = length * sin angle

length = 1

angle = x

I(x) = sin x

So we see that a real cosine function does indeed have, in a space defined by the axis of real numbers and the perpendicular imaginary axis, a vector representation that projects a sine function onto the imaginary axis.

Inner products

The inner product of two vectors is a measure of similarity, as illustrated by the following examples. See a text on matrix algebra for more details (e.g. on notation). The similarity-interpretation of inner products will be used later in the tutorial, where the comparison of the similarities of a signal with the real versus the imaginary parts of a wavelet will be shown to be the basic mechanism of wavelet analysis.

(-1 0 1 0) * (-1 0 1 0)' = 2

(-1 0 1 0) * (0 1 0 -1)' = 0

(-1 0 1 0) * (1 0 -1 0)' = -2

Note that, whilst the vectors are given above in vector notation, they could also be plotted as functions.

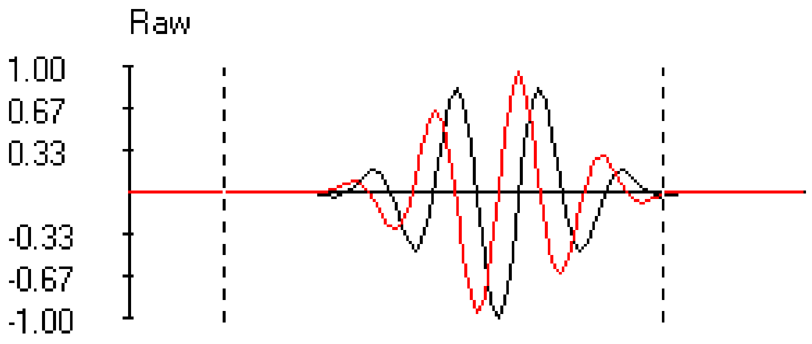
The estimation of instantaneous amplitude and phase by convolution

Wavelet analysis is based on the convolution of a wavelet of a given frequency with a signal. Convolution is, for the purposes of this tutorial, analogous to the calculation of the inner product described above, and is therefore a measure of similarity. The difference is that convolution is calculated for a continuous function instead of for the discrete elements of a vector (the same discrete – continuous relationship applies to variability and power). The convolution is performed for both the real and the imaginary parts of the wavelet. Because signal components will only be similar if they are of roughly the same frequency as the wavelet, the analysis automatically selects relevant signal components.

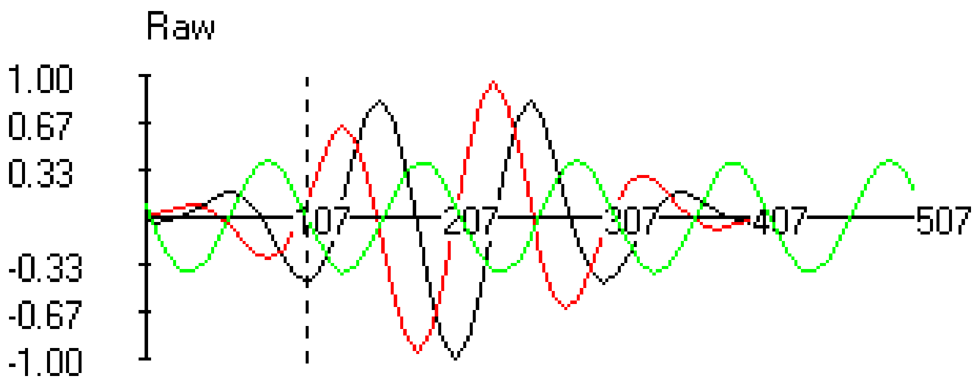
The instantaneous amplitude (A(t) in y(t) = A(t) * cos (x) for a given t, or instant in time) is proportional to the length of the vector pointing to the values of the real and imaginary convolutions. Note that the reason both the real and imaginary axes are necessary is that the real convolution in isolation would oscillate along with the signal. The combined convolution therefore abstracts the amplitude parameter from the actual signal values.

The phase can be defined as the angle of the vector with the real axis. A phase of zero means that the signal is in phase with a cosine beginning on the center of the wavelet. A phase-difference of zero between two signals means that they both overlap a wavelet with the same, arbitrary starting-phase.

The following figure is an example of the real (black) and imaginary (red) parts of a wavelet:



This wavelet is now superimposed on a signal (green) to be analyzed at the wavelet's frequency:



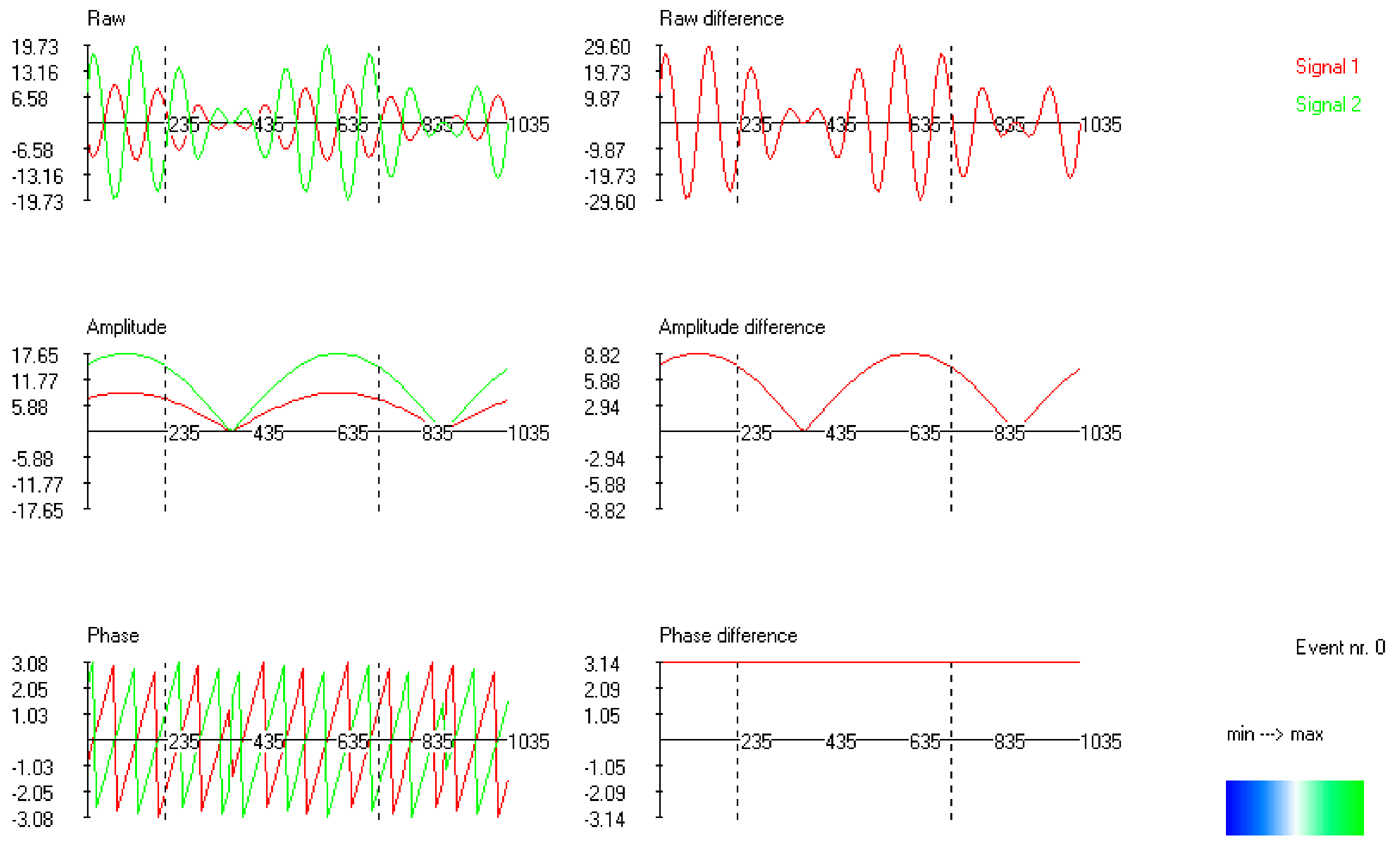
The convolution of the imaginary part with the signal is strongly negative (all contributing values are based on a minus-by-plus combination), whereas the real convolution is zero: there is no systematic relation between the positive and negative values of the signal with those of the wavelet.

It can be inferred from this combination of similarities that the phase of the signal is (0.25 + 0.5) * period (the quarter period due because a sine can be written as a phase-shifted cosine, the half because the signal and wavelet are in anti-phase) and that the amplitude is proportional to the imaginary convolution.

If the signal had been shifted to be more similar to the real part of the wavelet, the imaginary convolution would have correspondingly decreased; visualized on the unit circle, the arrow would have turned to follow the shift in the signal. Only simple goniometry is needed to extract the length and angle of the vector from the two orthogonal (that is, angled at 90 degrees and therefore maximally unsimilar themselves) similarity dimensions.

Example of the results of wavelet analysis at two frequencies

In the figure below, the three rows show the raw signal, the instantaneous amplitude and the instantaenous phase. The right column shows the respective differences between the signals.



Phase-locking measures are based on the instantaneous phase differences shown here in the lower-right plot.

Localization in time and frequency

Wavelets cannot be precisely localized in time or frequency. A wavelet cannot be precisely localized in frequency because it is not a pure (co-) sine. A wavelet cannot be precisely localized in time because a certain time-region of a function is necessary to determine similarity / to compute convolutions.

The localization in time and frequency is (for the Morlet wavelet) a Gaussian function around a central time and frequency, not a single time point (as in the raw signal) or frequency (as in Fourier analysis). The localization can therefore be described in terms of mean and standard deviation in sec and Hz (sigma_t and sigma_f) respectively.

The Gaussian localization curves indicate the influence of frequencies and time points at a distance from the mean; the broader the curve, the more influence distant points in time and frequency have and the higher the level of uncertainty of what part of signal is contributing to the convolution used to determine similarity.

The influence of distant points determines the resolution of the analysis; for example, the broader sigma_t is, the more time points of the signal together contribute to computations concerning one time point of the analysis.

Example of the time – frequency uncertainty relationship (from Tallon - Baudry, 1995)

Localization in time and frequency

The relation between sigma_t and sigma_f is constant:

sigma_f * sigma_t == 1 / (2 * π)

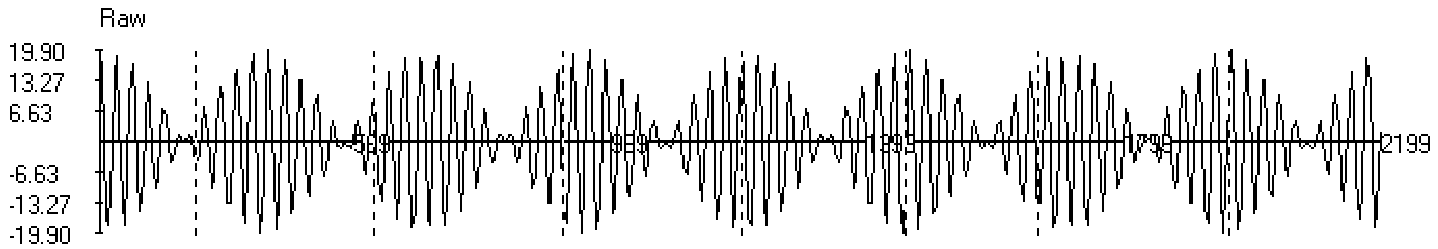
This constant relation implies that the poorer the resolution is in time, the better it is in frequency.

The relation of sigma_f to frequency depends on the so-called wavelet parameter:

sigma_f = f / param

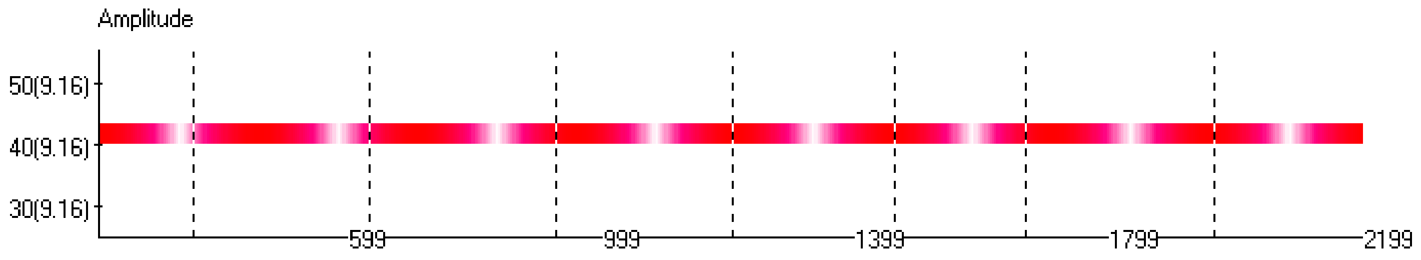
However, the parameter param may be adjusted at the whim of the researcher. If the param is kept constant, however, the analysis can be easily automatized and an interesting contrast with Fourier analysis is exposed. Fourier analysis (a common (co-)sine-based method for decomposing signals that requires the assumption that the signals are stationary) does not adjust the temporal window to different frequencies; in contrast, wavelet analysis does take advantage of the reduced temporal uncertainty that becomes available at higher frequencies. Therefore, wavelet analysis has a better overall time – frequency resolution than short-term Fourier analysis.

Example of the effect of sigma_f

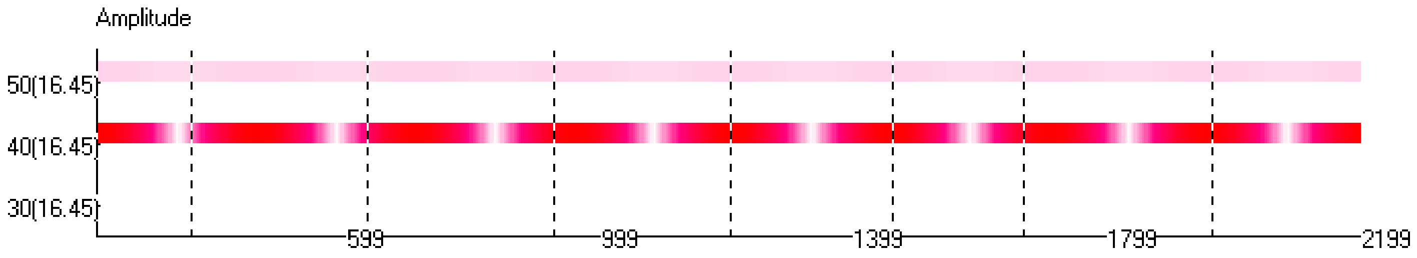


The raw signal plotted below contains a 40 Hz component with amplitude oscillating at 2 Hz. The three lower plots show the results of wavelet analyses, performed for three frequencies, using different wavelet parameters. The sigma_f for 40 Hz is given for each plot (note that the sigma_f will be larger for higher frequencies).

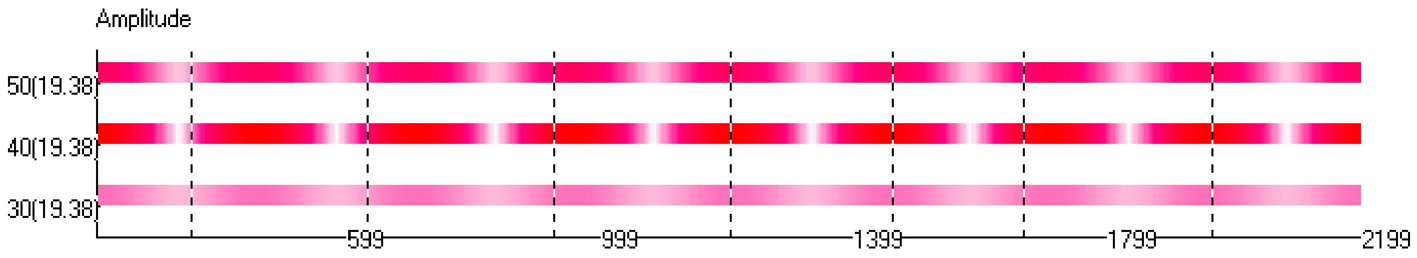
Sigma_f = 2 Hz



Sigma_f = 4 Hz



Sigma_f = 10 Hz



Phase-locking measures: the PLV

The instantaneous phase of a signal can be represented as the angle of a vector of length 1 with the real axis on the unit circle. In these terms, the phase-difference between two signals is the result of vector-subtracting one signal’s phase-vector from the other. In vector subtraction the values of corresponding elements in two vectors are subtracted.

Summation of the difference vector over trials and then taking the length divided by the number of trials provides the phase-locking value (PLV, Lachaux et al., 1995). The PLV is a measure for the consistency of the phase-difference over trials. The PLV can take values of 0 to 1, a value of 1 being achieved if all the phase differences are the same. In that case, vector-adding ("stacking") the difference-vectors leads to a maximal distance between the start of the first vector and the end of the last vector. The more variation between the trials, the more the string of vectors bunches up and the smaller the start - end distance.

The figure below is from Lachaux et al. (1995) and shows two situations, one in which the phase difference is consistent and the PLV high, the other the reverse case.

Phase-locking measures: the QPL

The quasi phase-locking value (QPL) is based on the PLV calculated over time points in a trial instead of over the same time point on different trials. The difference between the two measures is that the PLV will respond to a specific phase difference that is consistent over trials, whereas the QPL responds to the more "higher-order" consistency of the consistency of any phase difference over time within a trial. The procedure for calculating the QPL is as follows:

1. Select a number of adjacent time points as a segment of a trial. We used the samples within +/- 4 * σ t ms around the mean time.

2. Compute the intra-trial PLV over the time points of the segment. The values are weighted by a Gaussian curve around each analyzed time-point for the localization in time. Due to the smoothing effect of the wavelet (adjacent wavelets are largely based on the same set of samples), the intra-trial PLV will show an artifactual synchrony. However, as we are interested in changes over time and the artifactual synchrony is constant over time points this does not present a problem.

3. Repeat 1 and 2 for every trial.

4. Average the intra-trial PLV over all trials for the QPL value.

The QPL thus does not require signals to have the same phase-difference over trials, just to be phase-locked over time within trials.

C++ code

This is a function taken directly from a program I wrote for signal analysis, SAL. I copied the algorithm from a program written by Ritske de Jong. The code made the wavelet analysis procedure easier for me to understand initially than its "analytical" mathematical representation, so it might be useful for you.

```
void TfSpectro::performWaveletAnalysis(int s)
{
 double wf_samplerate = fSignalPool->samplingRate;
 if (wf_samplerate == 0)
 {
  fMain->inform("Error: zero waveform sample rate.");
  return;
 }

 int nTimesConvRealZero = 0;
 int iF = 0;
 while (iF < nFMids)
 {
  inform("Signal " + IntToStr(s) + ", frequency " + IntToStr(iF));
  double f = fMids[iF];
  double sigmaF = f / param,
         sigmaT = 1.0 / (2.0 * M_PI * sigmaF),
         correctie = 2.0 / (wf_samplerate * sigmaT * sqrt(2.0 * M_PI));
// bereken wavelet waarden op 6*sigmaT_dp dp om tijd-midden
  double sigmaT_dp = (sigmaT * wf_samplerate);
  int breedte = (int)(6.0 * sigmaT_dp);
  double *waveletReal = new double[breedte],
         *waveletIm = new double[breedte];
  double maxWlVal = 0.0;
  for (int wl_dp = 0; wl_dp < breedte; wl_dp++)
  {
   waveletReal[wl_dp] = exp( -0.5 * pow((double)wl_dp / sigmaT_dp, 2.0) ) *
                         cos( 2.0 * M_PI * f * (double)wl_dp / wf_samplerate
);
   waveletIm[wl_dp] =   exp( -0.5 * pow((double)wl_dp / sigmaT_dp, 2.0) ) *
                         sin( 2.0 * M_PI * f * (double)wl_dp / wf_samplerate
);
   if (waveletReal[wl_dp] > maxWlVal)
   {
    maxWlVal = waveletReal[wl_dp];
   }
   if (waveletIm[wl_dp] > maxWlVal)
```

```
    {
     maxWlVal = waveletIm[wl_dp];
    }
   }
// teken wavelet
  int xmax = im_Info->Width,
      ymax = im_Info->Height,
      xmid = xmax / 2,
      ymid = ymax / 2;
  double dx, dy;
  if (breedte > 0)
  {
   dx = (double)xmid / (double)breedte;
  }
  else
  {
   dx = 0;
  }
  dy = (double)(ymid) / maxWlVal;
  im_Info->Canvas->Brush->Color = clWhite;
  im_Info->Canvas->FillRect( Rect(0, 0, xmax, ymax) );
  for (int wl_dp = 1; wl_dp < breedte; wl_dp++)
  {
   im_Info->Canvas->Pen->Color = clBlack;
   im_Info->Canvas->MoveTo( xmid + (int)(dx * (double)wl_dp), ymid - (int)(dy
* waveletReal[wl_dp]) );
   im_Info->Canvas->LineTo( xmid + (int)(dx * (double)(wl_dp - 1)), ymid -
(int)(dy * waveletReal[wl_dp - 1]) );
   im_Info->Canvas->Pen->Color = clRed;
   im_Info->Canvas->MoveTo( (int)(dx * (double)wl_dp), ymid - (int)(dy *
waveletIm[wl_dp]) );
   im_Info->Canvas->LineTo( (int)(dx * (double)(wl_dp - 1)), ymid - (int)(dy
* waveletIm[wl_dp - 1]) );
  }
// bereken convoluties en enveloppes
  int dp = breedte;
  while (dp < fSignalPool->nDatapoints - breedte)
  {
// convoluties
   double convolutionReal = 0,
          convolutionIm = 0;
//   int dpPerPeriod = (wf_samplerate * (1.0 / f));
//   int nPeriods = (int)( (double)dp / (double)dpPerPeriod );
//   double wf_InstPhase = 2.0 * M_PI * ( (double)dp / (double)dpPerPeriod
//    - (double)nPeriods );
   int availableWidth = fFunctions->min(fFunctions->min(breedte, dp),
    fSignalPool->nDatapoints - dp);
   for (int conv_dp = -availableWidth + 1; conv_dp < availableWidth;
conv_dp++)
    {
     convolutionReal += waveletReal[abs(conv_dp)] *
      fSignalPool->signalPool[s][dp + conv_dp];
     if (conv_dp != 0)
     {
      convolutionIm += (conv_dp / abs(conv_dp)) * waveletIm[abs(conv_dp)] *
       fSignalPool->signalPool[s][dp + conv_dp];
     }
    else
    {
     convolutionIm += waveletIm[abs(conv_dp)] *
      fSignalPool->signalPool[s][dp + conv_dp];
    }
   } // conv_dp
// enveloppes
// amplitude
   amplitude[iF][s][dp] = correctie * sqrt( pow(convolutionReal, 2.0) +
    pow(convolutionIm, 2.0) );

//phase

   double period_t = 1.0 / f;
   double dpPerPeriod = wf_samplerate * period_t;
   double wf_InstPhase;
   double dpPastPhaseZero = fmod( dp, dpPerPeriod );
   double fractionOfPeriod = dpPastPhaseZero / dpPerPeriod;
   if (CBWFIP->Checked)
   {
    wf_InstPhase = fractionOfPeriod * 2.0 * M_PI;
   }
   else
   {
    wf_InstPhase = 0;
   }

   if (convolutionReal != 0)
   {
//    phase[iF][s][dp] = atan( convolutionIm / convolutionReal ) -
wf_InstPhase;
    phase[iF][s][dp] = acos( convolutionReal / ((1.0 / correctie) *
amplitude[iF][s][dp]) ) - wf_InstPhase;
    if (convolutionIm >= 0)
    {
     phase[iF][s][dp] *= -1;
    }
   }
   else
   {
    nTimesConvRealZero++;
    if (convolutionIm == 0)
    {
     phase[iF][s][dp] = 0.0 - wf_InstPhase;
    }
    else if ( convolutionIm > 0)
    {
     phase[iF][s][dp] = M_PI / 2.0 - wf_InstPhase;
    }
    else
    {
     phase[iF][s][dp] = 3.0 * M_PI / 2.0 - wf_InstPhase;
    }
   }

   int analyzedDp = dp;
   dp++;
   while (dp < analyzedDp + sigmaT_dp * fractionOfSigmaT)
   {
    if (dp >= fSignalPool->nDatapoints)
    {
     break;
    }
    amplitude[iF][s][dp] = amplitude[iF][s][analyzedDp];
    phase[iF][s][dp] = phase[iF][s][analyzedDp];
    dp++;
   } // while
  } // dp
  delete [] waveletReal;
  delete [] waveletIm;
  iF++;
 }// iF
 getParams();
```

Thomas Edward Gladwin
thomasgladwin@hotmail.com