Wavelet approaches to synchrony (wsyn) package vignette

Lawrence Sheppard, Jonathan Walter, Thomas Anderson, Lei Zhao, Daniel Reuman

The `wsyn` package provides wavelet-based tools for investigating population synchrony. Population synchrony is the tendency for population densities measured in different locations to be correlated in their fluctuations through time (A. Liebhold, Koenig, and Bjørnstad 2004). The basic dataset that `wsyn` helps analyze is one or more time series of the same variable, measured in different locations at the same times; or two or more variables so measured at the same times and locations. Tools are implemented for describing synchrony and for investigating its causes and consequences. Wavelet approaches to synchrony include Grenfell, Bjørnstad, and Kappey (2001); Viboud et al. (2006); Keitt (2008); Sheppard et al. (2016); Sheppard, Reid, and Reuman (2017); Sheppard et al. (2019); Walter et al. (2017); Anderson et al. (2018). The focus here is on techniques used by Sheppard et al. (2016); Sheppard, Reid, and Reuman (2017); Sheppard et al. (2019); Walter et al. (2017); Anderson et al. (2018). The techniques can also be used for data of the same format representing quantities not related to populations, or, in some cases, multiple measurements from the same location. Some of the functions of this package may also be useful for studying the case of "community synchrony," i.e., co-located populations of different species that fluctuate through time in a positively or negatively or time-lag-correlated way.

1 Preparing the data

A typical dataset for analysis using `wsyn` is an $N \times T$ matrix of numeric values where rows correspond to sampling locations (so the number of sampling locations is N) and columns correspond to evenly spaced times during which sampling was conducted (so the number of times sampling was conducted is T).

1.1 Missing data

Standard implementations of wavelet trasforms require time series consisting of measurements taken at evenly spaced times, with no missing data. Most functions provided in `wsyn` make these same assumptions and throw an error if data are missing. The user is left to decide on and implement a reasonable way of filling missing data. Measures of synchrony can be influenced by data filling techniques that are based on spatial interpolation. We therefore recommend that spatially informed filling procedures not be used. We have previously used the simple approach of replacing missing values in a time series by the median of the non-missing values in the time series (Sheppard et al. 2016). This approach, and other related simple procedures (Sheppard et al. 2016), seem unlikely to artefactually produce significant synchrony, or coherence relationships with other variables, but rely on the percentage of missing data being fairly low and may obscure detection of synchrony or significant coherence relationships if too many data are missing. For applications which differ meaningfully from the prior work for which the tools of this package were developed (e.g., Sheppard et al. (2016); Sheppard, Reid, and Reuman (2017); Sheppard et al. (2019); Walter et al. (2017); Anderson et al. (2018)), different ways of dealing with missing data may be more appropriate.

1.2 De-meaning, detrending, standardizing variance, and normalizing

A function `cleandat` is provided that performs a variety of combinations of data cleaning typically necessary for analyses implemented in `wsyn`, including de-meaning, linear detrending, standardization of time series variance, and Box-Cox transformations to normalize marginal distributions of time series. Most functions in `wsyn` assume at least that means have been removed from time series, and throw an error if this is not the case. Approaches based on Fourier surrogate (section 5) require time series with approximately normal marginals.

2 The wavelet transform

The function `wt` implements the complex Morlet wavelet transform, on which most other `wsyn` functions are based. An S3 class is defined for `wt`, and it inherits from the generic class `tts`. See the help files for the generator functions `wt` and `tts` for slot names and other information about these classes. Both classes have `set` and `get` methods for interacting with the slots, see help files for `tts_methods`, `wt_methods` and `setget_methods`. The `set` methods just throw an error, since generally one should not be changing the individual slots of objects of one of classes in `wsyn` as it breaks the consistency between slots.

Background on the wavelet transform is available from many sources, including Addison (2002), and we do not recapitulate it. We instead describe the wavelet transform operationally, and demonstrate the implementation of the wavelet transform in `wsyn` using examples from Fig. S2 of Sheppard et al. (2019). Given a time series $x(t)$, $t = 1, \ldots, T$, the wavelet transform $W_\sigma(t)$ of $x(t)$ is a complex-valued function of time, $t = 1, \ldots, T$, and timescale, σ. The magntiude $|W_\sigma(t)|$ is an estimate of the strength of the oscillations in $x(t)$ at time t occurring at timescale σ. The complex phase of $W_\sigma(t)$ gives the phase of these oscilaltions.

To demonstrate the wavelet transform, start by generating some data. Start with a sine wave of amplitude 1 and period 15 that operates for $t = 1, \ldots, 100$ but then disappears.

```
time1<-1:100
time2<-101:200
times<-c(time1,time2)

ts1p1<-sin(2*pi*time1/15)
ts1p2<-0*time2
ts1<-c(ts1p1,ts1p2)

ts<-ts1
```

Then add a sine wave of amplitude 1 and period 8 that operates for $t = 101, \ldots, 200$ but before that is absent.

```
ts2p1<-0*time1
ts2p2<-sin(2*pi*time2/8)
ts2<-c(ts2p1,ts2p2)

ts<-ts+ts2
```

Then add normally distributed white noise of mean 0 and standard deviation 0.5.

```
ts3<-rnorm(200,mean=0,sd=0.5)
ts<-ts+ts3
```

Now apply the wavelet transform, obtaining an object of class `wt`. Default parameter values for `scale.min`, `scale.max.input`, `sigma` and `f0` are usually good enough for initial data exploration.

```
library(wsyn)
```

```
##
## Attaching package: 'wsyn'
```

```
## The following object is masked from 'package:stats':
##
##     power
```

```
ts<-cleandat(ts,times,clev=1)
wtres<-wt(ts$cdat,times)
class(wtres)
```

```
## [1] "wt"   "tts"  "list"
```

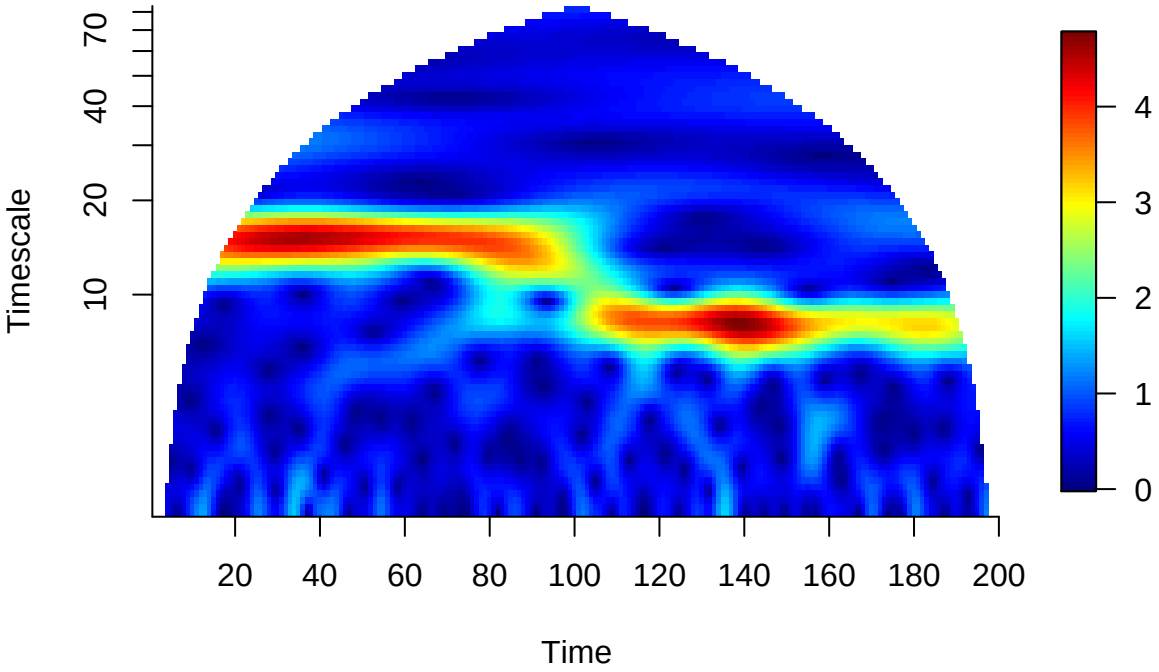
```
names(wtres)
```

```
## [1] "values"     "times"      "wtopt"      "timescales" "dat"
```

Methods `get_times`, `get_timescales`, `get_values`, `get_wtopt`, and `get_dat` extract the slots. Set methods also exist, but these just throw an error since setting individual slots of a `wt` object will break the relationship between the slots.

There is a `plotmag` method for the `tts` class that plots the magnitude of the transform against time and timescale.

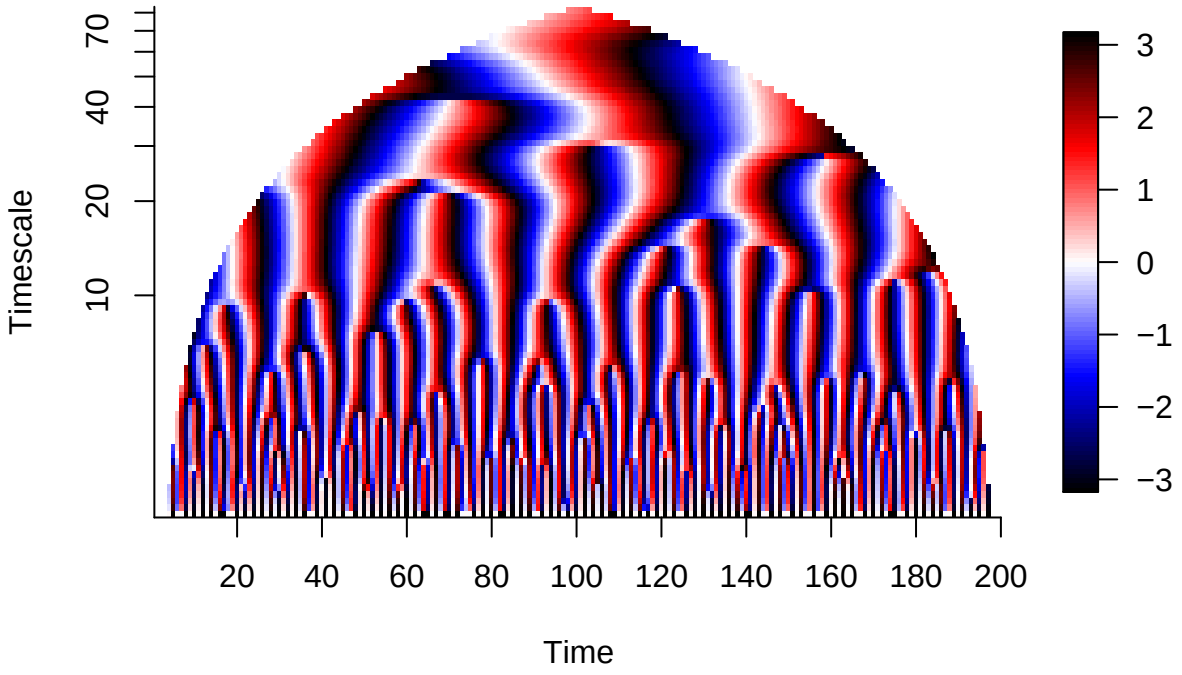
```
plotmag(wtres)
```



We can see the oscillations at timescale 15 for the first hundred time steps, and the oscilaltions at timescale 8 for the last 100 time steps, as expected. Because the wavelet transform is based on convolution of a wavelet function with the time series, times and timescales for which the overlap of the wavelet with the time series is insufficient are unreliable and are omitted. This affects times closer to the edges of the time series, and is the reason for the "rocketship nose cone" shape of wavelet plots. More values are omitted for longer timescales because long-timescale wavelets overhang the end of the time series further in the convolution operation. All plots based on wavelet transforms have the same property.

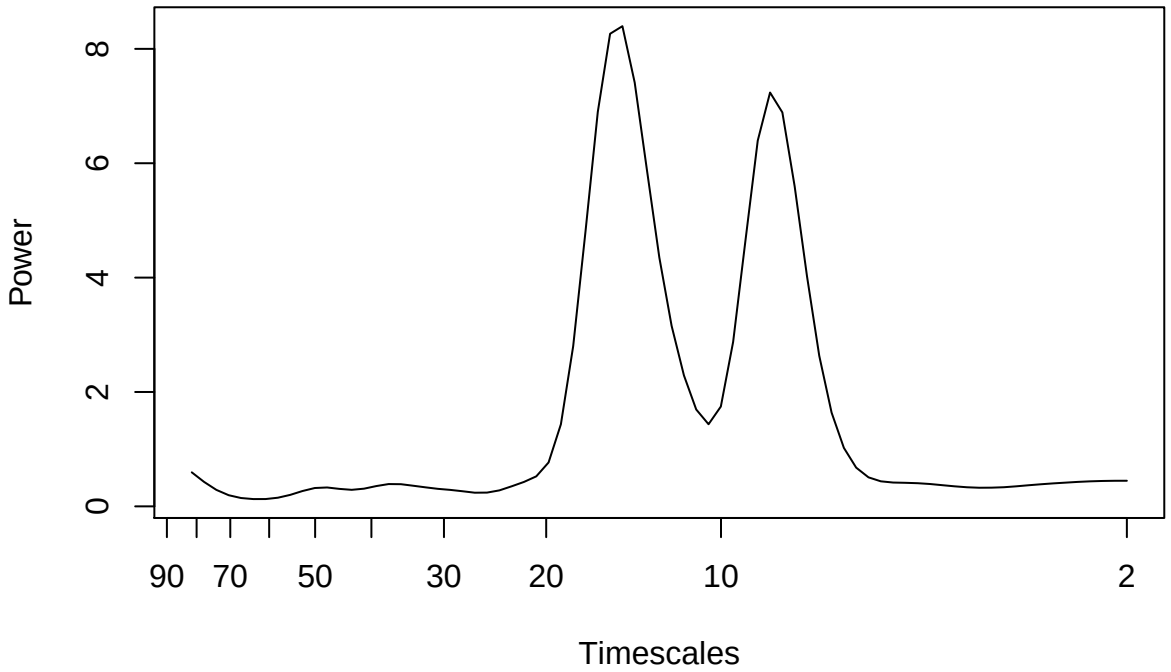
There is also a `plotphase` method for the `tts` class that plots the phase of the transform against time and timescale.

```
plotphase(wtres)
```



One can compute the power.

```
h<-power(wtres)
plot(log(1/h$timescales),h$power,type='l',lty="solid",xaxt="n",
     xlab="Timescales",ylab="Power")
xlocs<-c(min(h$timescales),pretty(h$timescales,n=8))
graphics::axis(side=1,at=log(1/xlocs),labels=xlocs)
```



There are also `print` and `summary` methods for the `wt` class and `tts` class. See the help files for `tts_methods` and `wt_methods`.

```
print(wtres)
```

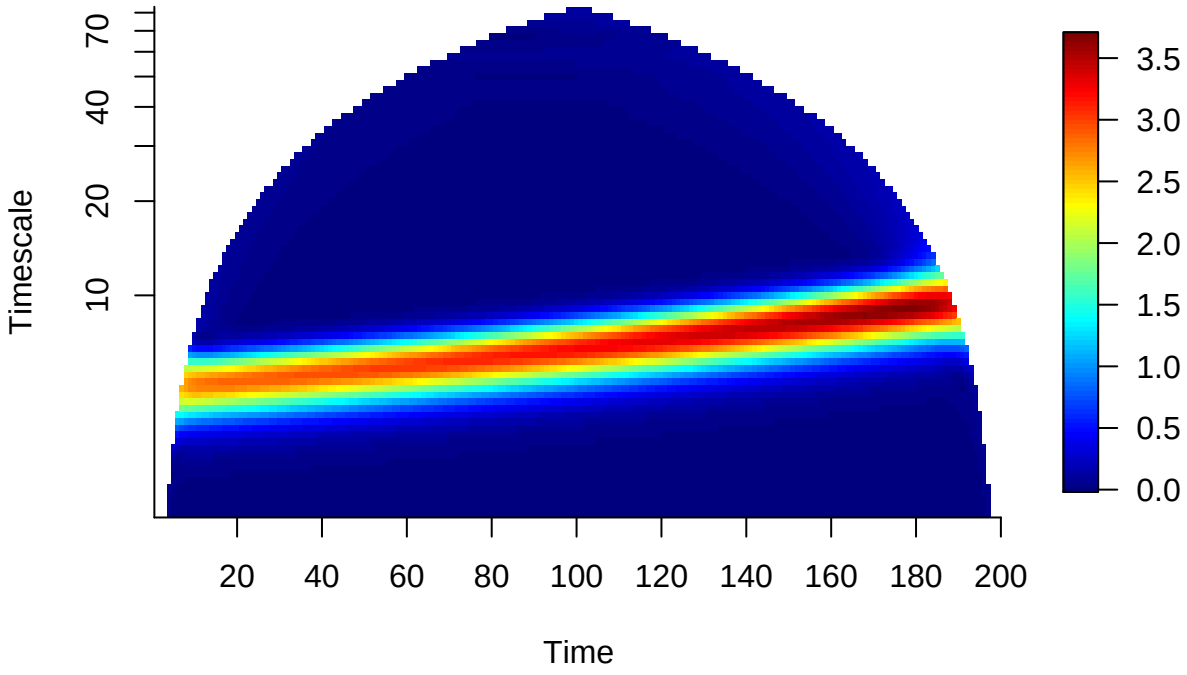
```
## wt object:
## times, a length 200 numeric vector:
## 1 2 3 4 5 ... 196 197 198 199 200
## timescales, a length 77 numeric vector:
## 2 2.1 2.205 2.31525 2.4310125 ... 67.0902683058082 70.4447817210986 73.9670208071536 77.6653718475113 81.5486
## values, a 200 by 77 matrix, upper left to four digits is:
##            [,1]            [,2]            [,3]            [,4]            [,5]
## [1,]         NA              NA              NA              NA              NA
## [2,]         NA              NA              NA              NA              NA
## [3,]         NA              NA              NA              NA              NA
## [4,]  0.4958+0i  0.4847-0.0672i  0.4664-0.1269i  0.4404-0.1634i  0.3969-0.1697i
## [5,] -0.3485+0i -0.3408+0.1189i -0.3339+0.2249i -0.3255+0.2877i -0.3041+0.2917i
## wtopt: scale.min=2; scale.max.input=NULL; sigma=1.05; f0=1
```

```
summary(wtres)
```

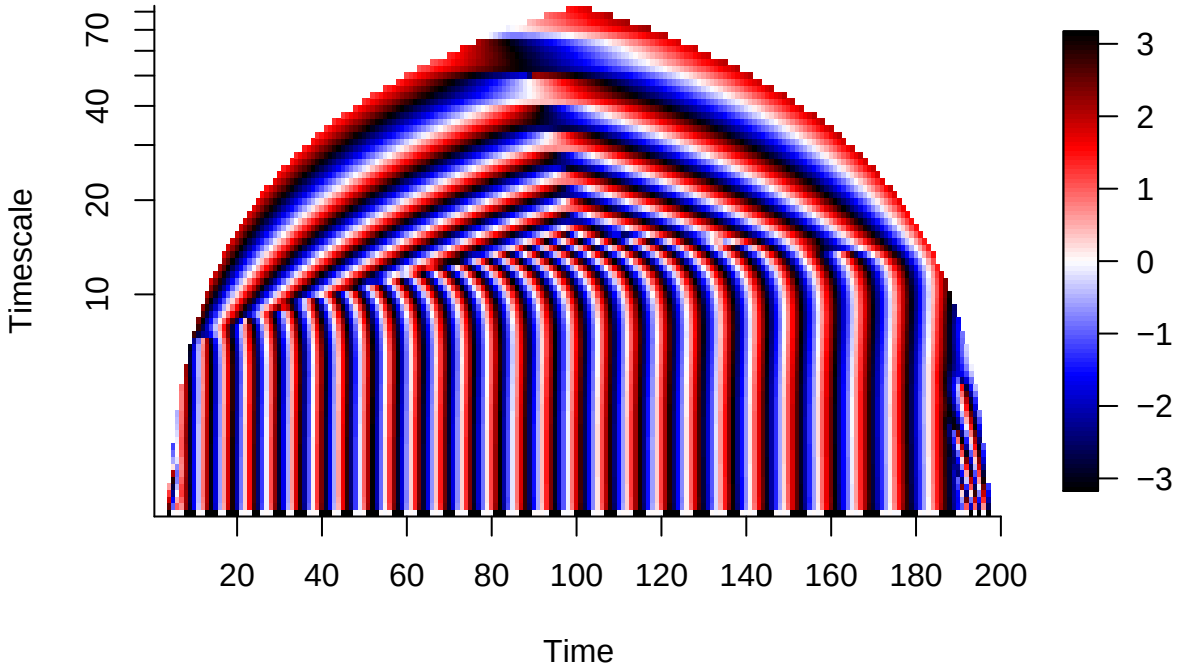
```
## class: wt
## times_start: 1
## times_end: 200
## times_increment: 1
## timescale_start: 2
## timescale_end: 81.54864
## timescale_length: 77
## scale.min: 2
## scale.max.input: NULL
## sigma: 1.05
## f0: 1
```

Now we give a second example, also from Fig. S2 of Sheppard et al. (2019). The frequency of oscillation of the data geenerated below changes gradually from 0.2 cycles per year (timescale 5 years) to 0.1 cycles per year (timescale 10 years).

```
timeinc<-1 #one sample per year
startfreq<-0.2 #cycles per year
endfreq<-0.1 #cycles per year
times<-1:200
f<-seq(from=startfreq,length.out=length(times),to=endfreq) #frequency for each sample
phaseinc<-2*pi*cumsum(f*timeinc)
t.series<-sin(phaseinc)
t.series<-cleandat(t.series,times,1)$cdat
res<-wt(t.series, times)
plotmag(res)
```




```
plotphase(res)
```



The `times` argument to `wt` (and to several other functions, see below) is tightly constrained. It must be a numeric vector of unit-spaced times (`diff(times)` is a vector of 1s) with no missing entries. It must be the same length as the data and correspond to the timing of measurement of the data. For the most common use cases, the unit spacing of times will be natural in some time unit, i.e., sampling is typically conducted with frequency once per time unit for some natural time unit (e.g., once per year, month, day, week, fortnight). In those cases, `timescales` in output and on plots will have units cycles per time unit for the time unit of sampling. Applications with sampling time step not equal to 1 in some natural unit of time can view the `times` vector as a vector of time steps, rather than times, *per se*, and `timescales` will be in units of cycles per time step.

The arguments `scale.min`, `scale.max.input`, `sigma`, and `f0`, which are arguments to `wt` and several other functions, are for

constructing the timescales used for wavelet analysis. The argument `scale.min` is the shortest timescale, and must be 2 or greater. Starting from `scale.min`, each timescale is `sigma` times the previous one, up to the first timescale that equals or surpasses `scale.max.input`. The scalloping of wavelet transforms places additional, independently implemented constraints on the largest timescale examined so choosing larger `scale.max.input` will only result in longer timescales up to the limits imposed by scalloping. The argument `f0` is the ratio of the period of fluctuation to the width of the envelope. Higher values of `f0` imply higher frequency resolution (i.e., a wavelet component includes information from a narrower range of Fourier components) but lower temporal resolution (the component includes information from a wider range of times). Resolution should be chosen appropriate to the characteristics of the data (Addison 2002).

3 Time- and timescale-specific measures of synchrony

The function `wpmf` implements the wavelet phasor mean field and the function `wmf` implements the wavelet mean field. These are techniques for depicting the time and timescale dependence of synchrony, and are introduced in this section. S3 classes are defined for `wmf` and `wpmf`, both of which inherit from the generic class `tts`. See the help files for the generator functions for these classes (`wmf`, `wpmf` and `tts`, respectively) for slot names and other information about the classes. There are again `set` and `get` methods for these classes, and `print` and `summary` methods. See help files for `wmf_methods` and `wpmf_methods`.

3.1 The wavelet phasor mean field

The wavelet phasor mean field (the `wpmf` function in `wsyn`) depicts the time and timescale dependence of phase synchrony of a collection of time series. If $x_n(t)$, $n = 1, \ldots, N$, $t = 1, \ldots, T$ are time series of the same variable measured in N locations at the same times, and if $W_{n,\sigma}(t)$ is the wavelet transform of $x_n(t)$ and $w_{n,\sigma}(t) = \frac{W_{n,\sigma}(t)}{|W_{n,\sigma}(t)|}$ has only the information about the complex phases of the transform (unit-magnitude complex numbers such as these are called *phasors*), then the wavelet phasor mean field is

$$\frac{1}{N}\sum_{n=1}^{N} w_{n,\sigma}(t). \tag{1}$$

For combinations of t and σ for which oscillations at time t and timescale σ in the time series $x_n(t)$ have the same phase (they are phase synchronized), the phasors $w_{n,\sigma}(t)$ will all point in similar directions in the complex plane, and their sum will be a large-magnitude complex number. For combinations of t and σ for which oscillations at time t and timescale σ in the time series $x_n(t)$ have unrelated phases (they are not phase synchronized), the phasors $w_{n,\sigma}(t)$ will all point in random, unrelated directions in the complex plane, and their sum will be a small-magnitude complex number. Therefore plotting the magnitude of the wavelet phasor mean field against time and timescale quantifies the time and timescale dependence of phase synchrony in the $x_n(t)$. The wavelet phasor mean field, being a mean of phasors, always has magnitude between 0 and 1.

We provide an example based on supplementary figure 1 of Sheppard et al. (2016). A related technique, the wavelet mean field (section 3.2), was used there, but the wavelet phasor mean field also applies and is demonstrated here. We construct data consisting of time series measured for 100 time steps in each of 11 locations. The time series have three components. The first component is a sine wave of amplitude 1 and period 10 years for the first half of the time series, and is a sine wave of amplitude 1 and period 5 for the second half of the time series. This same signal is present in all 11 time series and creates the synchrony among them.

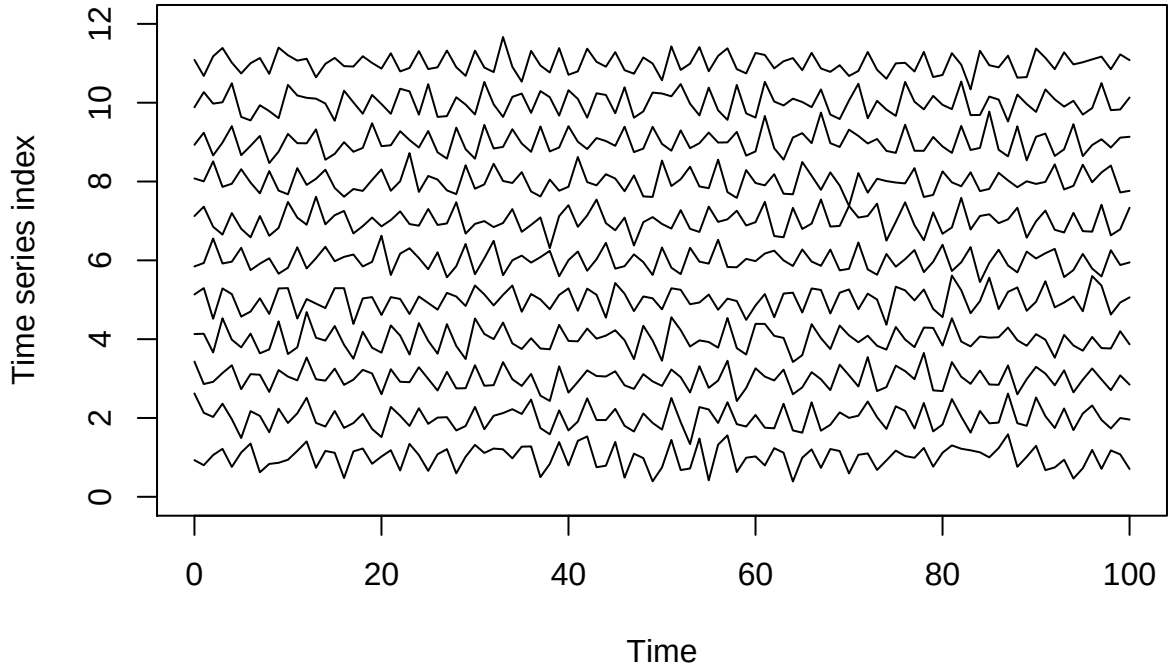
```
times1<-0:50
times2<-51:100
times<-c(times1,times2)
ts1<-c(sin(2*pi*times1/10),sin(2*pi*times2/5))+1.1
```

The second component is a sine wave of amplitude 1 and period 3 years that is randomly and independently phase shifted in each of the 11 time series. The third component is white noise, independently generated for each time series.

```
dat<-matrix(NA,11,length(times))
for (counter in 1:dim(dat)[1])
{
  ts2<-3*sin(2*pi*times/3+2*pi*runif(1))+3.1
  ts3<-rnorm(length(times),0,1.5)
  dat[counter,]<-ts1+ts2+ts3
}
dat<-cleandat(dat,times,1)$cdat
```

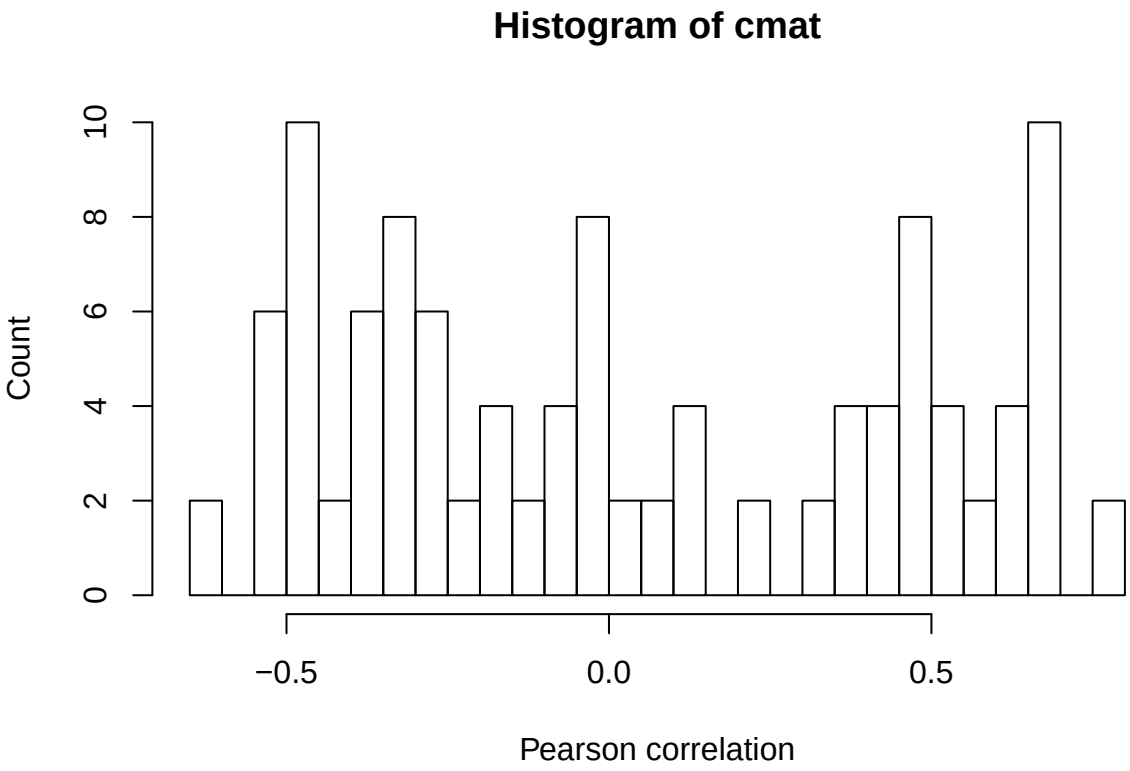
The second and third components do not generate synchrony, and obscure the synchrony of the first component. As a result, synchrony cannot be readily detected by visually examining the time series:

```
plot(times,dat[1,]/10+1,type='l',xlab="Time",ylab="Time series index",ylim=c(0,12))
for (counter in 2:dim(dat)[1])
{
  lines(times,dat[counter,]/10+counter)
}
```



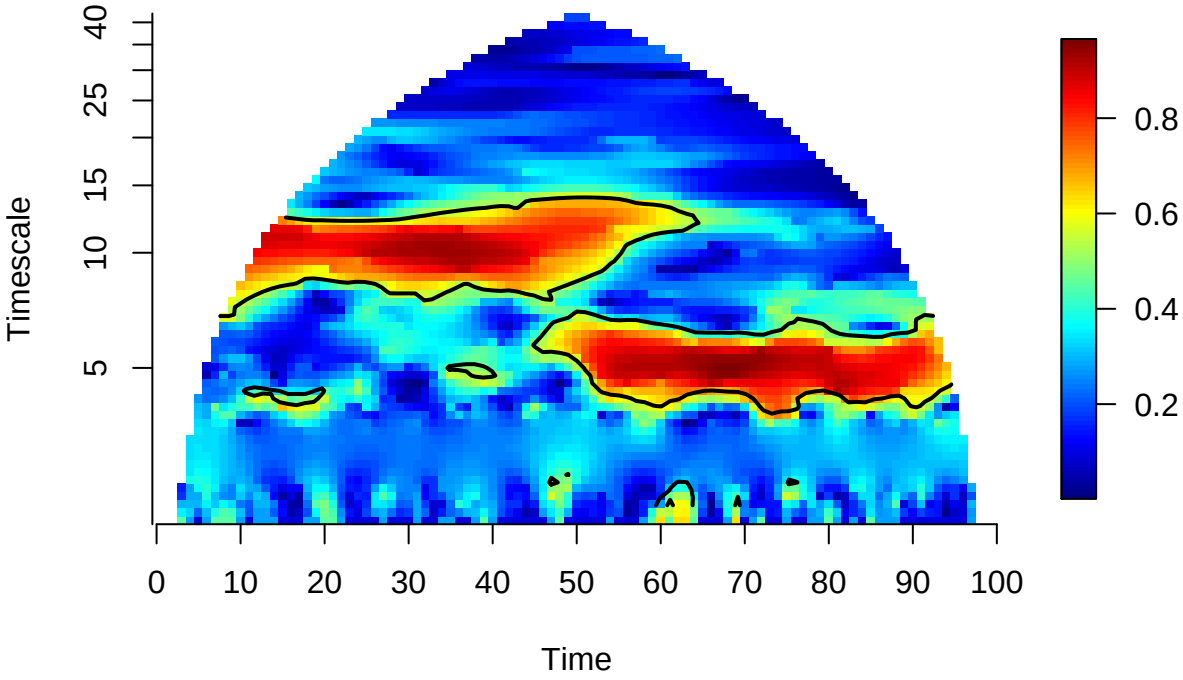
Nor can synchrony be readily detected by examining the 55 pairwise correlation coefficients between the time series, which are widely distributed and include many values above and below 0:

```
cmat<-cor(t(dat))
diag(cmat)<-NA
cmat<-as.vector(cmat)
cmat<-cmat[!is.na(cmat)]
hist(cmat,30,xlab="Pearson correlation",ylab="Count")
```



But the wavelet phasor mean field sensitively reveals the synchrony and its time and timescale structure:

```
res<-wpmf(dat,times,sigmethod="quick")
plotmag(res)
```



The `wpmf` function implements assessment of the statistical significance of phase synchrony in three ways, one of which is demonstrated by the contour lines on the above plot (which give a 95% confidence level by default - the level can be changed with the `sigthresh` argument to the `plotmag` method for the `wpmf` class). The method of significance testing the wavelet phasor

mean field plot is controlled with the `sigmethod` argument to `wpmf`, which can be `quick` (the default), `fft` or `aaft`. The `quick` method compares the mean field magnitude value for each time and timescale separately to a distribution of magnitudes of sums of N random, independent phasors.

Each time/timescale pair is compared independently to the distribution, and the multiple testing problem is not accounted for, so some time/timescales pairs will come out as showing "significant" phase synchrony by chance, i.e., false-positive detections of phase synchrony can occur. For instance, the small islands of significant synchrony at timescale approximately 5 and times about 15 and 40 on the above plot are false positives.

Signficance is based on stochastic generation of magnitudes of sums of random phasors, so significance contours will differ slightly on repeat runs. Increasing the number of randomizations (argument `nrand` to `wpmf`) reduces this variation.

The `quick` method can be inaccurate for very short timescales. The two alternative methods, `fft` and `aaft`, mitigate this problem but are substantially slower. The `fft` and `aaft` methods are based on surrogate datasets (section 5) so are discussed in section 5.4.

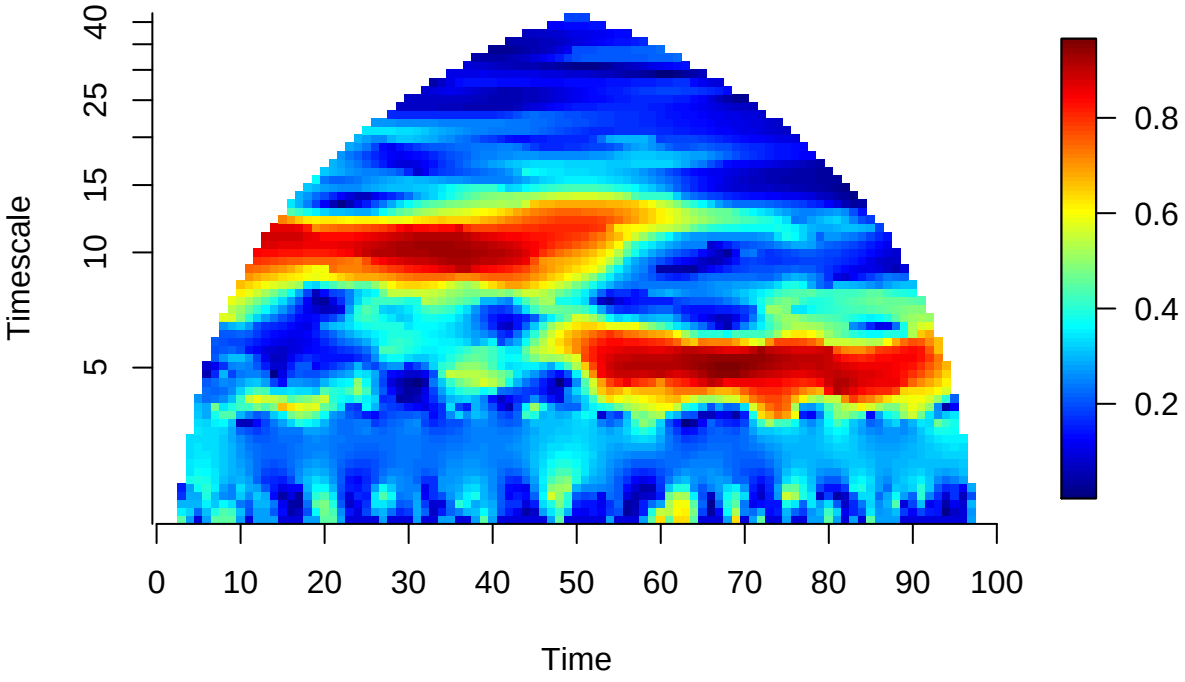
The `power` and `plotphase` methods also work on `wpmf` objects.

Examples of the wavelet phasor mean field technique applied to real datasets are in, for instance, Sheppard et al. (2019) (their Fig. S1) and Anderson et al. (2018) (their Fig. 4).

3.2 The wavelet mean field

The wavelet mean field (the `wmf` function in `wsyn`) depicts the time and timescale dependence of synchrony of a collection of time series $x_n(t)$ for $n = 1, \ldots, N$ and $t = 1, \ldots, T$, taking into account both phase synchrony and associations through time of magnitudes of oscillations in different time series at a given timescale. See Sheppard et al. (2016) for a precise mathematical definition. The plot is similar in format to a wavelet phasor mean field plot, but without significance contours:

```
res<-wpmf(dat,times)
plotmag(res)
```



The wavelet mean field is a more useful technique than the wavelet phasor mean field insofar as it accounts for associations of magnitudes of oscillation, in addition to phase synchrony, but it is less useful insofar as significance contours are not available. The wavelet mean field also has some mathematical advantages, described in Sheppard et al. (2016) and Sheppard et al. (2019).

The "wavelet Moran theorem" and other theorems described in those references use the wavelet mean field, not the wavelet phasor mean field, and can be used to help attribute synchrony to particular causes. Thus the two techniques are often best used together: significance of phase synchrony can be identified with the wavelet phasor mean field, and then once significance is identified, synchrony can be described and studied using the wavelet mean field.

The `power` and `plotphase` methods also work on `wmf` objects.

Examples of the wavelet mean field applied to real data are in Sheppard et al. (2016) and Sheppard et al. (2019).

4 Coherence

Coherence is explained by Sheppard et al. (2016) and Sheppard, Reid, and Reuman (2017), among others. We summarize here some of the explanations given in those references. Let $x_{1,n}(t)$ and $x_{2,n}(t)$ for $n = 1, \ldots, N$ and $t = 1, \ldots, T$ be two variables measured at the same N locations and T times. Let $W_{i,n,\sigma}(t)$ $(i = 1, 2)$ be the corresponding wavelet transforms. The coherence of $x_{1,n}(t)$ and $x_{2,n}(t)$ is the magnitude of a quantity we denote $\Pi_\sigma^{(12)}$, which is in turn the mean of $w_{1,n,\sigma}(t)\overline{w_{2,n,\sigma}(t)}$ over all time-location pairs for which this product of normalized wavelet transforms is still defined after wavelet scalloping is performed. The overline is complex conjugation. This product is a function of timescale. The $w_{1,n,\sigma}(t)$ are wavelet transforms normalized in one of a few different ways (see below). Because $w_{1,n,\sigma}(t)\overline{w_{2,n,\sigma}(t)}$ is a complex number with phase equal to the phase difference between the two wavelet components, the mean, $\Pi_\sigma^{(12)}$, of this quantity over times and locations has large magnitude if the phase difference between the transforms is consistent over time and across sampling locations. Coherences essentially measure the strength of association between the variables in a timescale-specific way that is also not confounded by lagged or phase-shifted associations. Coherences and related quantities are analyzed in `wsyn` using the function `coh` and its corresponding S3 class, `coh`, and the S3 methods that go with the class. Methods comprise `set` and `get` and `print` and `summary` methods (see the help file for `coh_methods` for these basic methods), as well as `bandtest`, and `plotmag`, `plotrank`, and `plotphase`.

One possible normalization is phase normalization, $w_{i,n,\sigma}(t) = \frac{W_{i,n,\sigma}(t)}{|W_{i,n,\sigma}(t)|}$. Set the `norm` argument of `coh` to "`phase`" to use this normalization. The coherence with this normalization is often called the *phase coherence*, or the *spatial phase coherence* if $N > 1$ (Sheppard, Reid, and Reuman 2017). Phase coherence measures the extent to which the two variables have consistent phase differences over time and across locations, as a function of timescale. The normalization descibed in the "Wavelet mean field" section of the Methods of Sheppard et al. (2016) gives the version of the coherence that was there called the *wavelet coherence*, or the *spatial wavelet coherence* if $N > 1$. Set the `norm` argument of `coh` to "`powall`" to use this normalization. If `norm` is "`powind`", then $w_{i,n,\sigma}(t)$ is obtained by dividing $W_{i,n,\sigma}(t)$ by the square root of the average of $W_{i,n,\sigma}(t)\overline{W_{i,n,\sigma}(t)}$ over the times for which it is defined; this is done separately for each i and n. The final option for `norm` available to users of `coh` is "`none`", i.e., raw wavelet transforms are used. For any value of `norm` except "`phase`", the normalized wavelet components $w_{i,n,\sigma}(t)$ can also have varying magnitudes, and in that case the coherence reflects not only consistencies in phase between the two variables over time and across locations, but is also further increased if there are correlations in the amplitudes of the fluctuations.

We demonstrate coherence and its implementation in `wsyn` via some simulated data. This demonstration reporoduces an example given in supplementary figure 5 of Sheppard et al. (2016). Data consist of an environmental-driver variable, `x`, and a driven biological variable, `y`, between which we compute coherence. Both were measured at 11 locations. The environmental variable `x` was constructed as the sum of: 1) a single common signal of amplitude 1 and period 10 years, present at all 11 locations; 2) a single common signal of amplitude 5 and period 3 years, also present at all 11 locations; 3) white noise of mean 0 and standard deviation 1.5, independently generated for all 11 locations.

```
times<-(-3:100)
ts1<-sin(2*pi*times/10)
ts2<-5*sin(2*pi*times/3)
x<-matrix(NA,11,length(times)) #the driver (environmental) variable
for (counter in 1:11)
{
  x[counter,]=ts1+ts2+rnorm(length(times),mean=0,sd=1.5)
}
```

Population time series $y_n(t)$ were the moving average, over three time steps, of the $x_n(t)$, plus white noise: $y_n(t) =$

$\left(\sum_{k=0}^{2} x_n(t-k)\right)/3 + \epsilon_n(t)$. Here the $\epsilon_n(t)$ are independent, normally distributed random variables of mean 0 and standard deviation 3.

```
times<-0:100
y<-matrix(NA,11,length(times)) #the driven (biological) variable
for (counter1 in 1:11)
{
  for (counter2 in 1:101)
  {
    y[counter1,counter2]<-mean(x[counter1,counter2:(counter2+2)])
  }
}
y<-y+matrix(rnorm(length(times)*11,mean=0,sd=3),11,length(times))
x<-x[,4:104]
x<-cleandat(x,times,1)$cdat
y<-cleandat(y,times,1)$cdat
```

The function `cleandat` with `clev=1` (mean removal only) was used. Mean removal is sufficient cleaning for these artificially generated data.

The relationship between the environmental and biological variables cannot readily be detected using ordinary correlation methods - correlations through time between the $x_n(t)$ and $y_n(t)$ range widely on both sides of 0:

```
allcors<-c()
for (counter in 1:dim(x)[1])
{
  allcors[counter]<-cor(x[counter,],y[counter,])
}
allcors
```

```
##  [1]  0.04718970  0.02647060  0.05421114 -0.00676947 -0.22863024  0.10363673
##  [7] -0.03022360  0.01025200  0.07389408  0.20168476  0.11456214
```

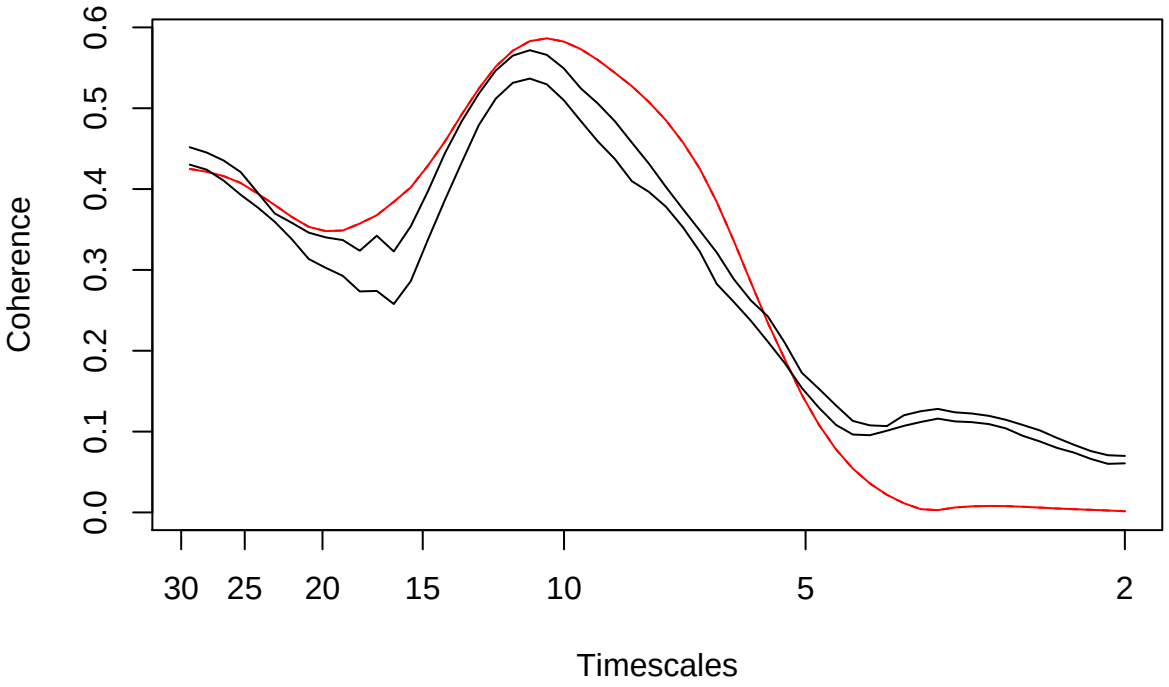
However, the function `coh` can be used to compute the coherence between `x` and `y`:

```
res<-coh(dat1=x,dat2=y,times=times,norm="powall",
         sigmethod="fftsurrog1",nrand=100,
         f0=0.5,scale.max.input=28)
```

The normalization to be used is specified via `norm`, as described above. The `powall` option corresponds to the (spatial) wavelet coherence of Sheppard et al. (2016). There are several alternative methods for testing the significance of coherence, and the method used is controlled by the `sigmethod` argument. All significance methods are based on "surrogate datasets". These are datasets that have been randomized in an appropriate way. In addition to computing coherence of data, coherence is also computed in the same way for `nrand` surrogate datasets that represent the null hypothesis of no relationship between `dat1` and `dat2` while retaining other statistical features of the data. See section 5 for details of surrogates and significance testing, and allowed values of `sigmethod`. Larger values of `nrand` produce more accurate significance results that are less variable on repeat runs, but also require more computational time. Using `nrand` at least 1000 or 10000 for final runs is recommended. This can take some time for values of `sigmethod` other than `fast` (see section 5), so `100` was used above for demonstration purposes only. The arguments `f0` and `scale.max.input` control details of the wavelet transform (see section 2) and are set here to agree with values used by Sheppard et al. (2016).

Coherence can be plotted using `plotmag`:

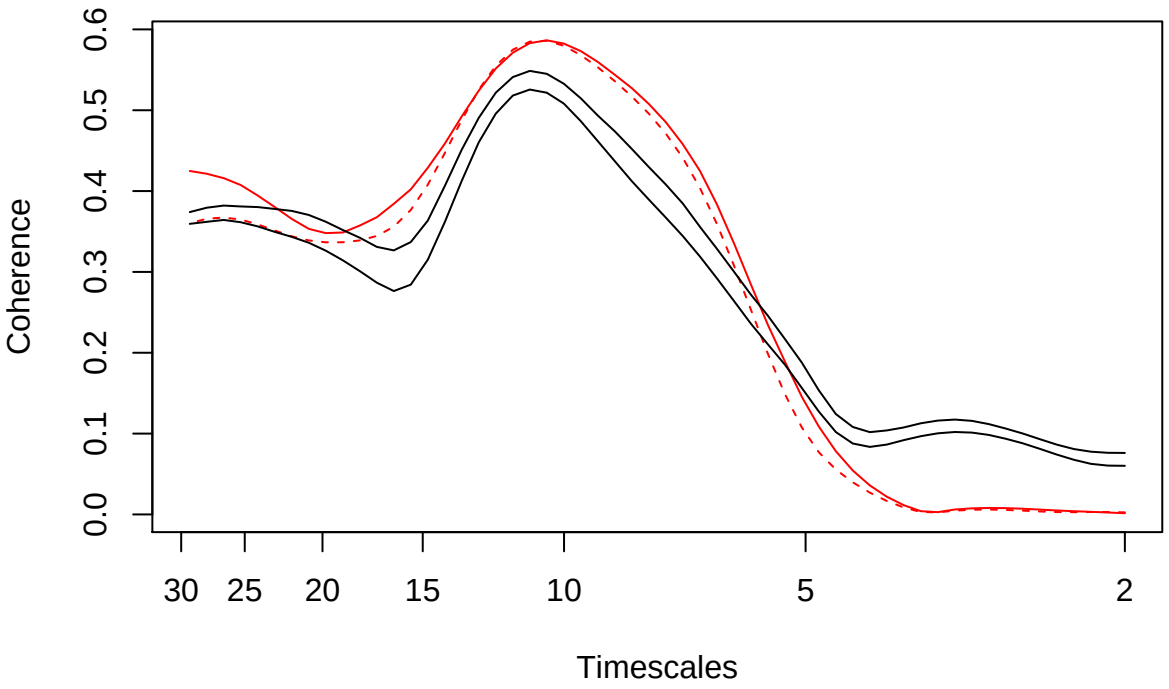
```
plotmag(res)
```



The red line here is the coherence. The black lines are 95th and 99th (these are the default values for `sigthresh`) quantiles of coherences of surrogate datasets. Coherence is significant (the red line is above the black lines) for timescales close to 10 years, but not significant for timescales close to 3 years, as expected since three values of x are averaged to produce one value of y, so the 3-year periodicity in x is averaged away and does not pass to y. Thus coherence reveals a (timescale-specific) relationship between x and y that correlation methods did not reveal.

Typically preferable (Sheppard, Reid, and Reuman 2017) is the `fast` option to `sigmethod`, because far more surrogates can be used in the same computational time:

```
res<-coh(dat1=x,dat2=y,times=times,norm="powall",
         sigmethod="fast",nrand=10000,
         f0=0.5,scale.max.input=28)
plotmag(res)
```



For the `fast` algorithm (section 5) the modulus of `res$signif$coher` (plotted above as the dashed red line) can be compared to quantiles of the modulus of `res$signif$scoher` (the black lines are 95th and 99th quantiles) in the usual way to make statements about significance of coherence, but the modulus of `res$signif$coher` is only approximately equal to the standard coherence, which is the modulus of `res$coher` (and which is plotted above as the solid red line). Thus one should use the dashed red line above, and `res$signif$coher`, when making conclusions about the significance of coherence, and the solid red line, and `res$coher`, when using the actual value of the coherence. Typically the two red lines are quite similar. For values of `sigmethod` other than `fast`, they are equal.

The significance indicated on the above plots is done on a timescale-by-timescale basis, and type-I errors (false positives) are not taken into account. Neither do the individual timescales correspond to independent tests. A method of aggregating signficance across a timescale band was described by Sheppard et al. (2016), and is implemented in `bandtest`:

```
res<-bandtest(res,c(8,12))
```

A call to `bandtest` computes a p-value for the aggregate significance of coherence across the specified band (8 to 12 year timescales, in this case), and also computes the average phase of $\Pi_\sigma^{(12)}$ across the band. This information is added as a new row to the `bandp` slot of the `coh` object (which was previously `NA` in this case).

```
get_bandp(res)
```

```
##   ts_low_bd ts_hi_bd     p_val   mn_phs
## 1         8       12 9.999e-05 1.033593
```

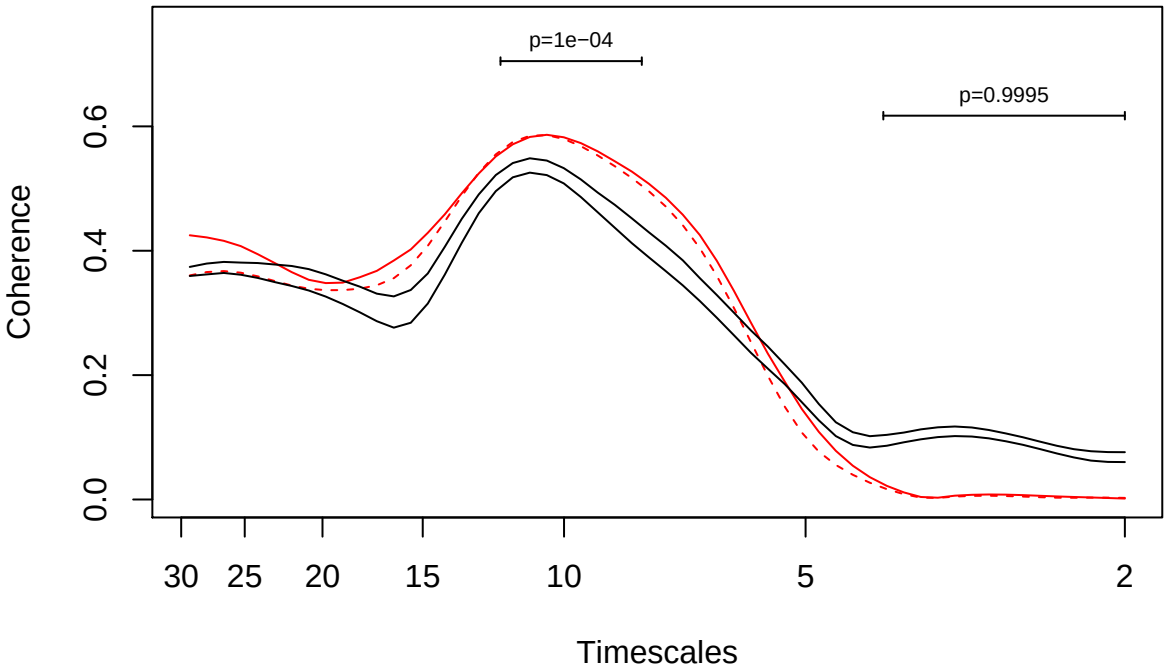
Doing another timescale band adds another row to `bandp`:

```
res<-bandtest(res,c(2,4))
get_bandp(res)
```

```
##   ts_low_bd ts_hi_bd     p_val    mn_phs
## 1         8       12 9.999e-05  1.033593
## 2         2        4 9.995e-01 -1.679863
```

The aggregate p-values are now displayed on the plot:

```
plotmag(res)
```

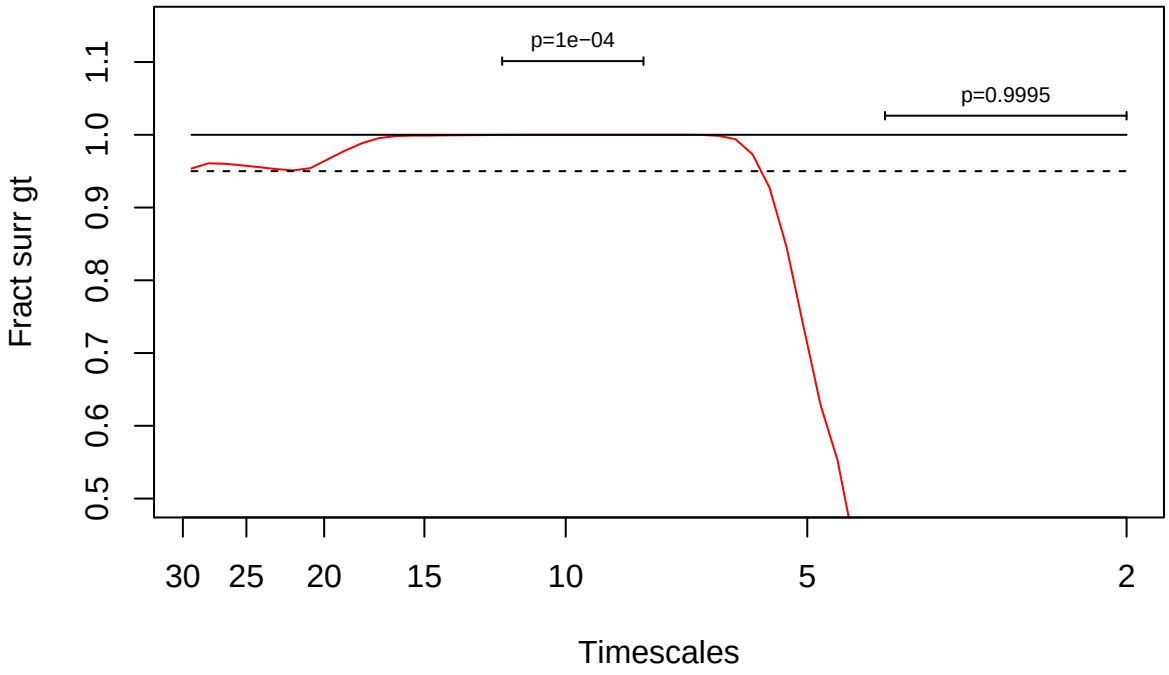


These results show the variables x and y are highly significantly coherent across the timescale band 8 to 12 years, but are not signficantly coherent across the band 2 to 4 years, as expected from the way the data were generated and by comparing the red and black lines.

Band-aggregated p-values are produced essentially by averaging the rank in surrogates of the empirical coherence across timescales. The same procedure is then applied to each surrogate, ranking it with respect to the other surrogates and taking the mean across timescales. Comparing the empirical mean rank to the dsitribution of surrogate mean ranks gives a p-value (Sheppard et al. 2016; Sheppard, Reid, and Reuman 2017; Sheppard et al. 2019).

One can also display a plot of the ranks of `Mod(res$signif$coher)` in the distribution of `Mod(res$signif$scoher)` values at each timescale:

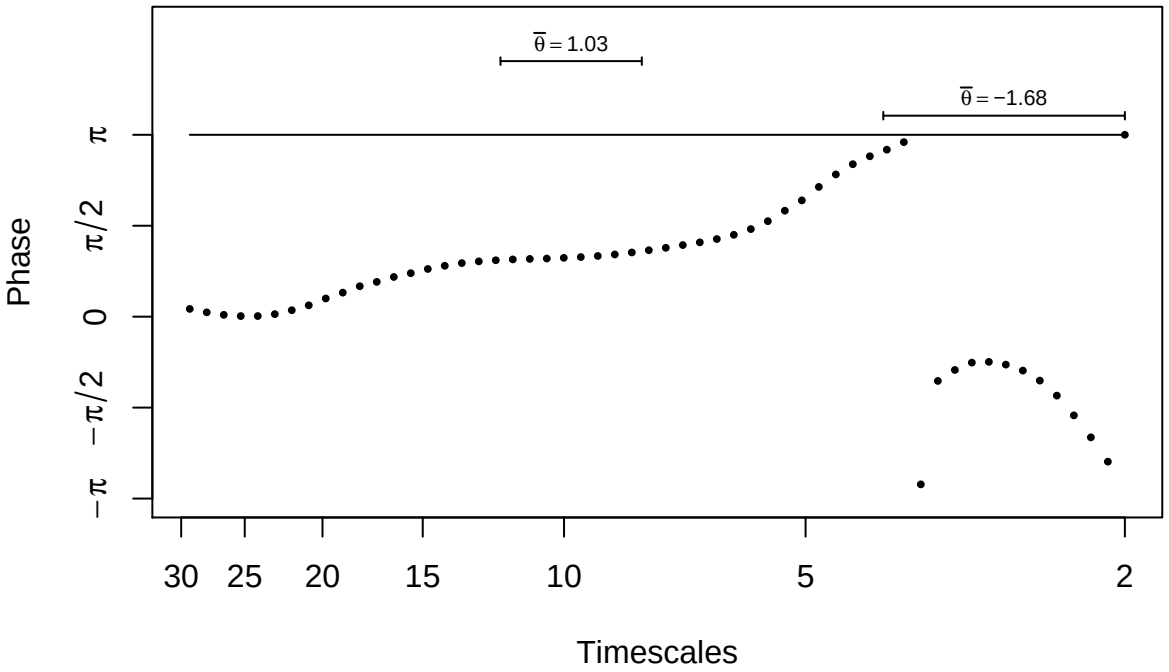
```
plotrank(res)
```



The vertical axis label `Fract surr gt` stands for the fraction of surrogate coherences that the coherence of the data is greater than at the given timescale, so values are between 0 and 1 and large values indicate significance. Whenever values exceed the argument `sigthresh` (which takes the default value 0.95 for the above call to `plotrank`), the coherence is nominally significant. The value(s) of `sigthresh` are displayed as dashed horizontal line(s) on the plot. This is nominal significance because of the multiple-testing problem. As can be seen, p-values stored in `bandp` are also displayed on the plot, and these values aggregate across timescales appropriately and alleviate the multiple-testing problem.

Average phases of $\Pi_\sigma^{(12)}$ across the timescale bands of interest were also computed by `bandtest` and stored in the `bandp` slot, in units of radians. These are average phases for the `coher` slot of a `coh` object. Phases of $\Pi_\sigma^{(12)}$ can be plotted against timescale and average phases in `bandp` displayed using the `plotphase` function:

```
plotphase(res)
```



Average phases for timescale bands across which coherence is not significant (e.g., the 2 to 4 year band in the above plot) are random and meaningless. Average phases for bands across which coherence is significant (e.g., the 8 to 12 band in the plot) can give valuable information about the nature of the relationship between the variables (Sheppard et al. 2019).

5 Surrogates

Some of the text of this section was adapted, with minor modifications only, from our earlier work (Sheppard et al. 2016; Sheppard et al. 2019; Anderson et al. 2018).

The level of coherence consistent with the null hypothesis that there is no relationship between two variables depends on the spatial and temporal autocorrelation of the data. For instance, two variables that fluctuate regularly at the same frequency and are both highly spatially synchronous will have a phase difference that is highly consistent over time and space, and therefore will have high spatial wavelet coherence, even if they are not related. Two irregular oscillators with low spatial synchrony are less likely to show consistent phase differences over time and space if they are unrelated. We test coherences for significance using resampling schemes based on surrogate datasets that randomize away phase relationship between variables while retaining, to the extent possible, the spatial and temporal autocorrelation properties and the marginal distributions of the time series. We use the widely applied Fourier surrogate and amplitude adjusted Fourier surrogate methods (Prichard and Theiler 1994; Schreiber and Schmitz 2000), implemented in the `surrog` function in `wsyn` and summarized below. Surrogates are also used for applications other than measures of coherence (see, e.g., section 5.4).

5.1 Fourier surrogates

Details are presented elsewhere (Prichard and Theiler 1994; Schreiber and Schmitz 2000). We summarize here. A Fourier surrogate of a time series $x(t)$ is obtained by the following steps:

- Compute the fast Fourier transform of $x(t)$, here called $X(\tau)$ for the timescale τ
- Randomize the phases of the transform by multiplying $X(\tau)$ by a random, uniformly distributed unit-magnitude complex number; do this independently for each τ
- Inverse transform, giving the surrogate time series

This procedure can be done using `surrog` with `surrtype="fft"`. Because only the phases of the Fourier transform are randomized, not the magnitudes, autocorrelation properties of the surrogate time series are the same as those of $x(t)$.

Fourier surrogates of N time series $x_n(t)$ measured at locations $n = 1, \ldots, N$ and times $t = 1, \ldots, T$ are obtained by the following steps:

- Compute the fast Fourier transform of $x_n(t)$ for each n, here called $X_n(\tau)$
- Randomize the phases of the transforms by multiplying $X_n(\tau)$ by a random, uniformly distributed unit-magnitude complex number. Do this independently for each τ, but different random multipliers can optionally be used for each n if desired, or the same phase multiplier can be used for all n, for a given τ (these are called "synchrony preserving surrogates" - see below)
- Inverse transform, giving the surrogate time series

This procedure can be done using `surrog` with `surrtype=fft` and with `syncpres=TRUE` (for synchrony-preserving surrogates) or with `syncpres=FALSE` (for independent surrogates). Autocorrelation properties of individual time series are preserved, as for the $N = 1$ case covered above. If synchrony-preserving surrogates are used, all cross-correlation properties between time series are also preserved, because cross spectra are unchanged by the joint phase randomization. Therefore synchrony is preserved.

Fourier surrogates tend to have normal marginal distributions (Schreiber and Schmitz 2000). Therefore, to ensure fair comparisons between statistical descriptors (such as coherences) of real and surrogate datasets, Fourier surrogates should only be applied to time series that themselves have approximately normal marginals. The Box-Cox transformations implemented in `cleandat` can help normalize data prior to analysis. If data are difficult to normalize, or as an alternative, the amplitude-adjusted Fourier transform surrogates method of the next section can be used instead.

5.2 Amplitude-adjusted Fourier surrogates

Amplitude-adjusted Fourier surrogates are described elsewhere (Schreiber and Schmitz 2000). Either synchrony preserving (`syncpres=TRUE`) of independent (`syncpres=FALSE`) amplitude-adjusted Fourier (AAFT) surrogates can be obtained from `surrog` using `surrtype="aaft"`. AAFT surrogates can be applied to non-normal data, and return time series with exactly the same marginal distributions as the original time series. AAFT surrogates have approximately the same power spectral (and cross-spectral, in the case of `syncpres=TRUE`) properties as the original data.

5.3 Fast coherence

The fast coherence algorithm implemented in `coh` (option `sigmethod="fast"`) implements Fourier surrogates only, and only applies for `norm` equal to `none`, `powall`, or `powind`. It is described in detail elsewhere (Sheppard, Reid, and Reuman 2017).

5.4 Alternatives to the "quick" method of assessing significance of wavelet phasor mean field values

When `sigmethod` is `fft` in a call to `wpmf`, the empirical wavelet phasor mean field is compared to wavelet phasor mean fields of Fourier surrogate datasets. The `signif` slot of the output is a list with first element `"fft"`, second element equal to `nrand`, and third element the fraction of surrogate-based wavelet phasor mean field magnitudes that the empirical wavelet phasor mean field magnitude is greater than (a times by timescales matrix). For `sigmethod` equal to `aaft`, AAFT surrogates are used instead. Non-synchrony-preserving surrogates are used.

6 Wavelet linear models and their uses for understanding synchrony

Linear models on wavelet transforms were introduced by Sheppard et al. (2019), where they were used for understanding the causes of synchrony. We demonstrate the implementation in `wsyn` of the tools developed by Sheppard et al. (2019), without giving a complete description of the concepts or mathematics behind those tools. Such a description is in Sheppard et al. (2019).

6.1 Model construction tools

First create a diver variable composed of an oscillation of period 12 years and an oscillation of period 3 years, and normally distributed white noise of mean 0 and standard deviation 1.5.

```
lts<-12
sts<-3
mats<-3
times<-seq(from=-mats,to=100)
ts1<-sin(2*pi*times/lts)
ts2<-sin(2*pi*times/sts)
numlocs<-10
d1<-matrix(NA,numlocs,length(times)) #the first driver
for (counter in 1:numlocs)
{
  d1[counter,]<-ts1+ts2+rnorm(length(times),mean=0,sd=1.5)
}
```

Next create a second driver, again composed of an oscillation of period 12 years and an oscillation of period 3 years, and normally distributed white noise of mean 0 and standard deviation 1.5.

```
ts1<-sin(2*pi*times/lts)
ts2<-sin(2*pi*times/sts)
d2<-matrix(NA,numlocs,length(times)) #the second driver
for (counter in 1:numlocs)
{
  d2[counter,]<-ts1+ts2+rnorm(length(times),mean=0,sd=1.5)
}
```

Next create an irrelevant environmental variable. With real data, of course, one will not necessarily know in advance whether an environmental variable is irrelevant to a population system. But, for the purpose of demonstrating the methods, we are playing the dual role of data creator and analyst.

```
dirrel<-matrix(NA,numlocs,length(times)) #the irrelevant env var
for (counter in 1:numlocs)
{
  dirrel[counter,]<-rnorm(length(times),mean=0,sd=1.5)
}
```

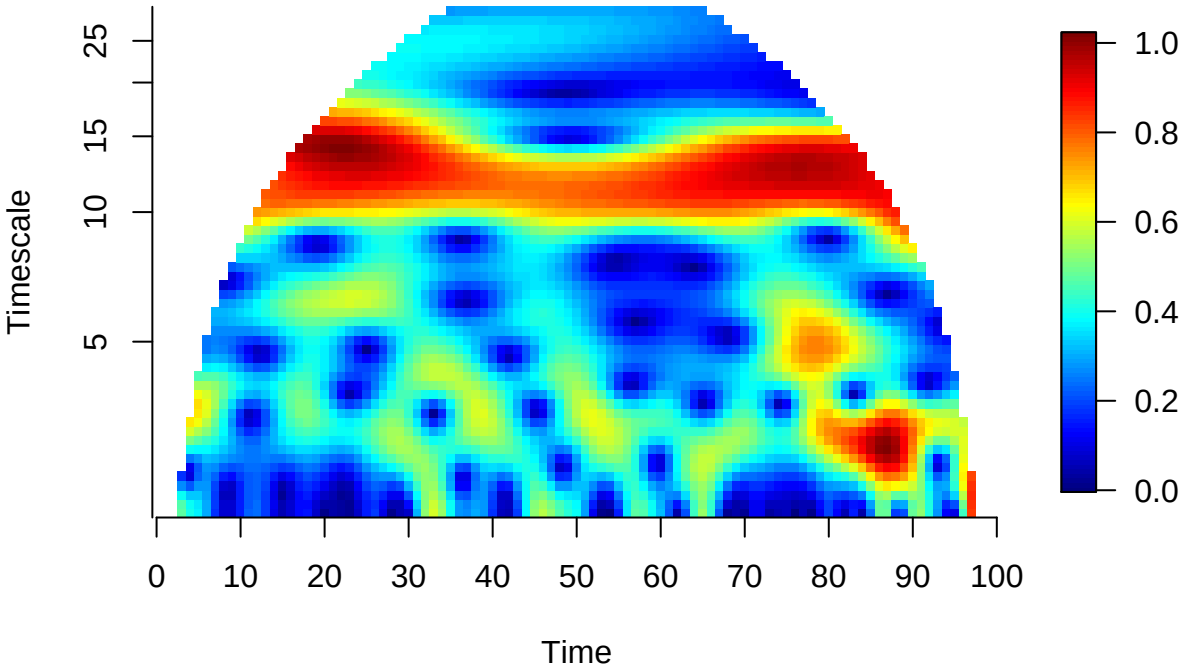
The population in each location is a combination of the two drivers, plus local variability. Driver 1 is averaged over 3 time steps in its influence on the populations, so only the period-12 variability in driver 1 influences the populations.

```
pops<-matrix(NA,numlocs,length(times)) #the populations
for (counter in (mats+1):length(times))
{
  aff1<-apply(FUN=mean,X=d1[,(counter-mats):(counter-1)],MARGIN=1)
  aff2<-d2[,counter-1]
  pops[,counter]<-aff1+aff2+rnorm(numlocs,mean=0,sd=3)
}
pops<-pops[,times>=0]
d1<-d1[,times>=0]
d2<-d2[,times>=0]
dirrel<-dirrel[,times>=0]
times<-times[times>=0]
```

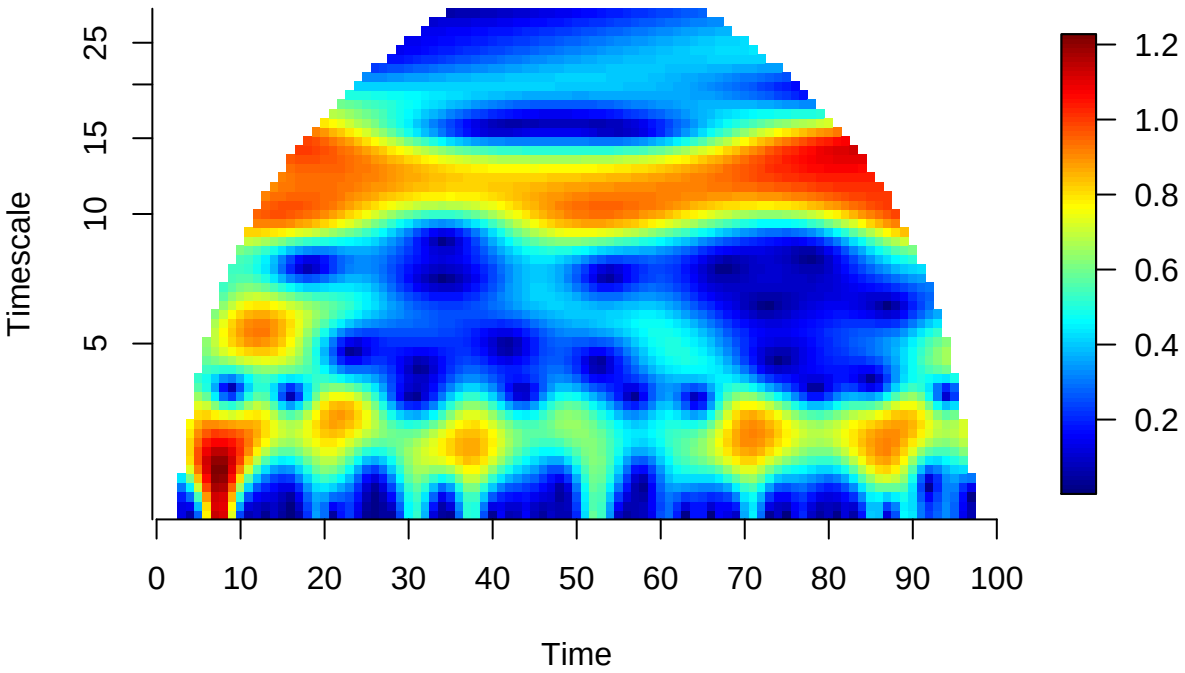
If only the data were available and we were unaware of how they were generated, we may want to infer the causes of synchrony and its timescale-specific patterns in the populations. The wavelet mean fields of `pops`, `d1` and `d2` show some synchrony at

timescales of about 3 and 12 for all three variables.

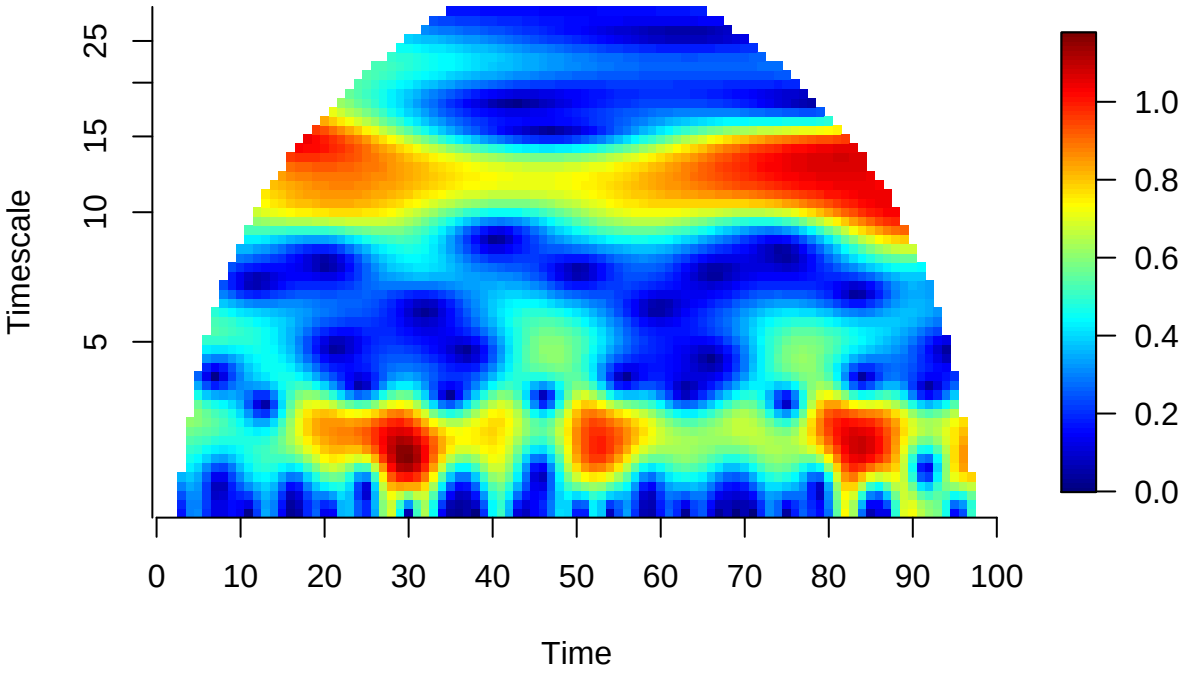
```
dat<-list(pops=pops,d1=d1,d2=d2,dirrel=dirrel)
dat<-lapply(FUN=function(x){cleandat(x,times,1)$cdat},X=dat)
wmfpop<-wmf(dat$pops,times,scale.max.input=28)
plotmag(wmfpop)
```




```
wmfd1<-wmf(dat$d1,times,scale.max.input=28)
plotmag(wmfd1)
```




```
wmfd2<-wmf(dat$d2,times,scale.max.input=28)
plotmag(wmfd2)
```



Thus we cannot know for sure from the wavelet mean fields whether population synchrony at each timescale is due to synchrony in `d1`, `d2`, or both drivers at that timescale. However, we can fit wavelet linear models.

Start by fitting a model with all three predictors. Only the `"powall"` option for `norm` is implemented so far.

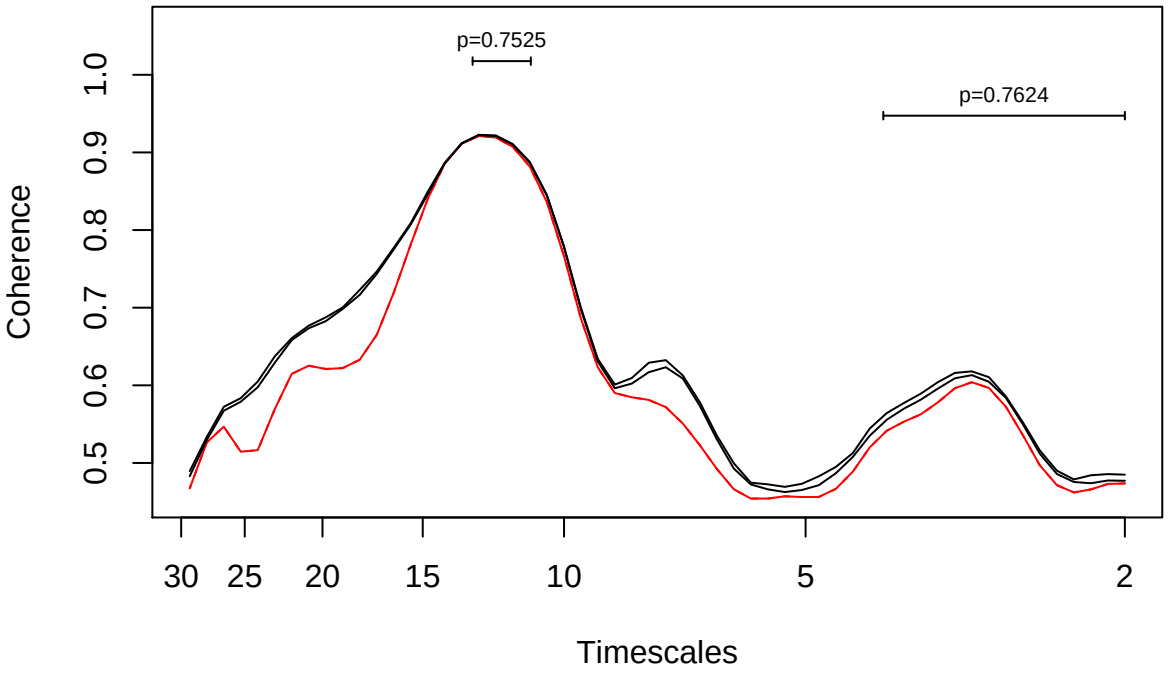
```
wlm_all<-wlm(dat,times,resp=1,pred=2:4,norm="powall",scale.max.input=28)
```

We will carry out analyses for this model at long timescales (11 to 13 years) and short timescales (2 to 4 years) simultaneously. First test whether we can drop each variable.

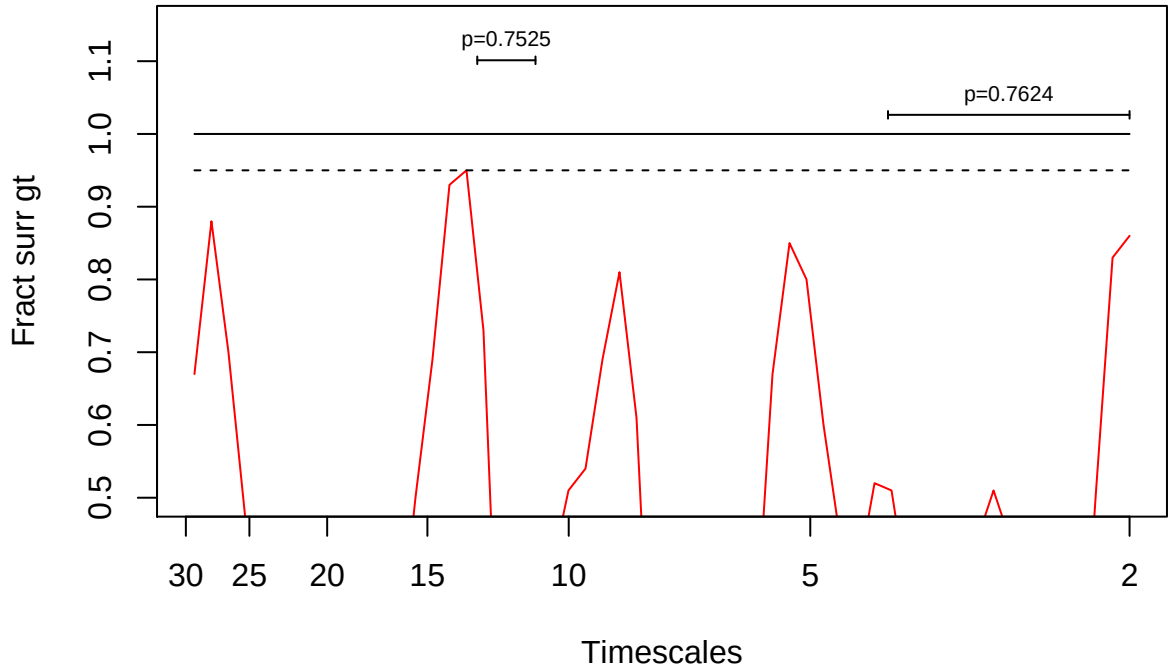
```
wlm_all_dropi<-wlmtest(wlm_all,drop="dirrel",sigmethod="fft",nrand=100)
wlm_all_drop1<-wlmtest(wlm_all,drop="d1",sigmethod="fft",nrand=100)
wlm_all_drop2<-wlmtest(wlm_all,drop="d2",sigmethod="fft",nrand=100)
```

Examine results for dropping `dirrel`, long and short timescales. We find that `dirrel` does not need to be retained in either long- or short-timescale models, as expected given how data were constructed:

```
blong<-c(11,13)
bshort<-c(2,4)
wlm_all_dropi<-bandtest(wlm_all_dropi,band=blong)
wlm_all_dropi<-bandtest(wlm_all_dropi,band=bshort)
plotmag(wlm_all_dropi)
```

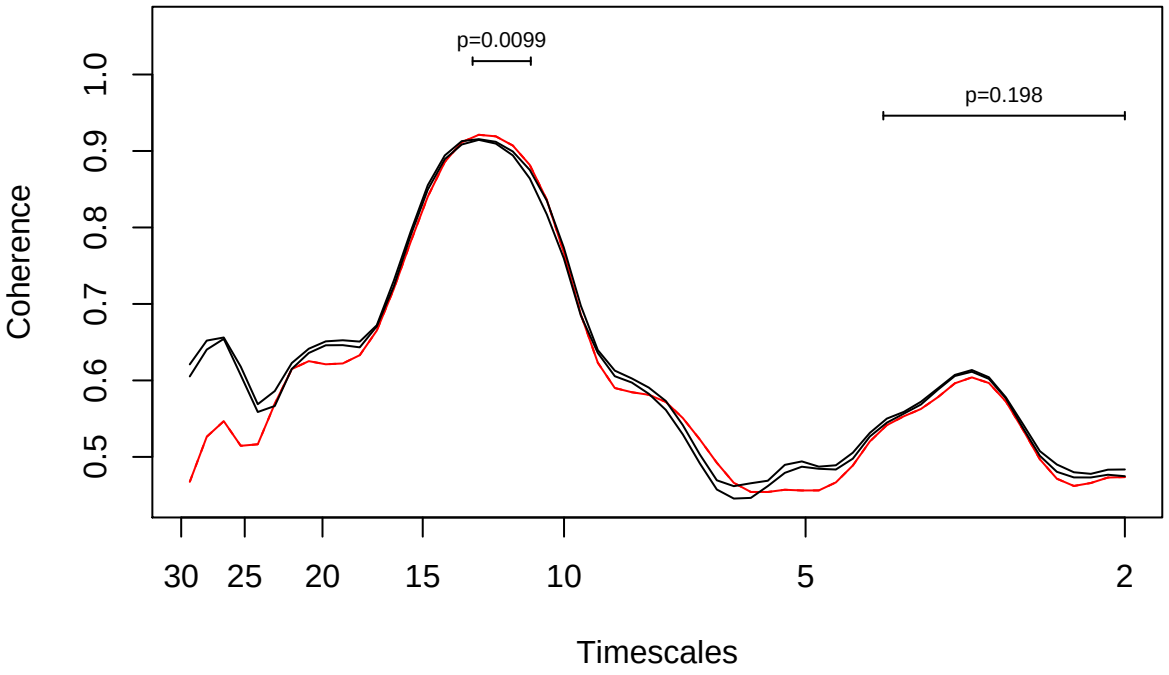



```
plotrank(wlm_all_dropi)
```

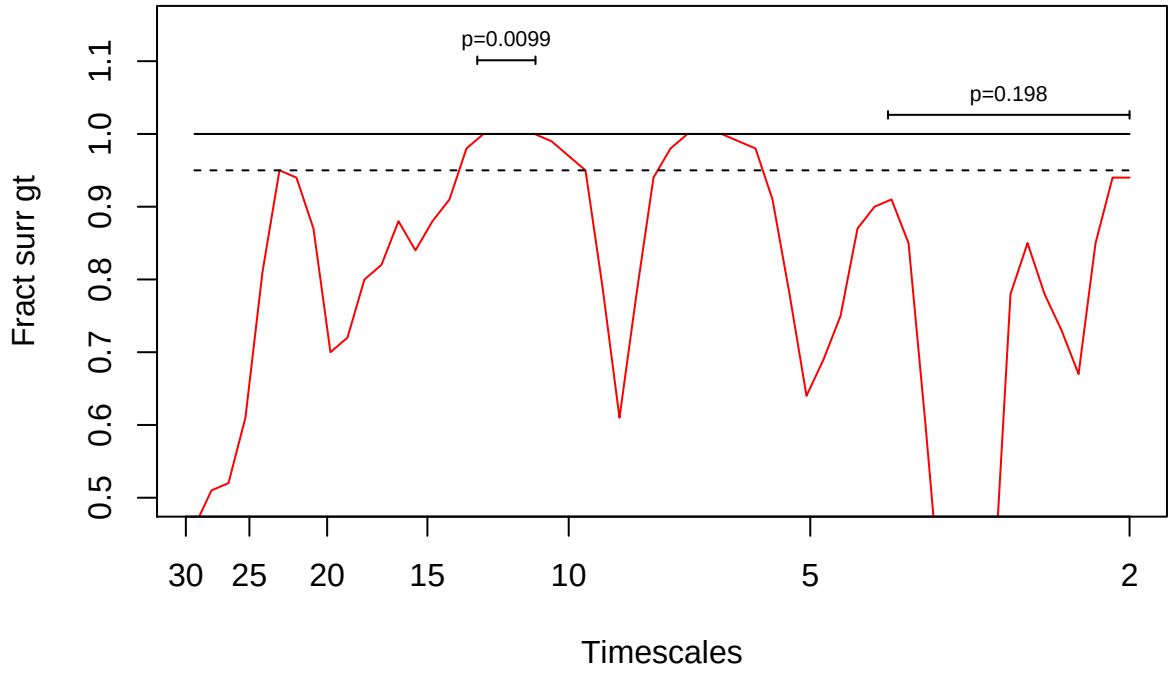


Examine results for dropping `d1`, long and short timescales. We find that `d1` should be retained in a long-timescale model but need not be retained in a short-timescale model, again as expected:

```
wlm_all_drop1<-bandtest(wlm_all_drop1,band=blong)
wlm_all_drop1<-bandtest(wlm_all_drop1,band=bshort)
plotmag(wlm_all_drop1)
```

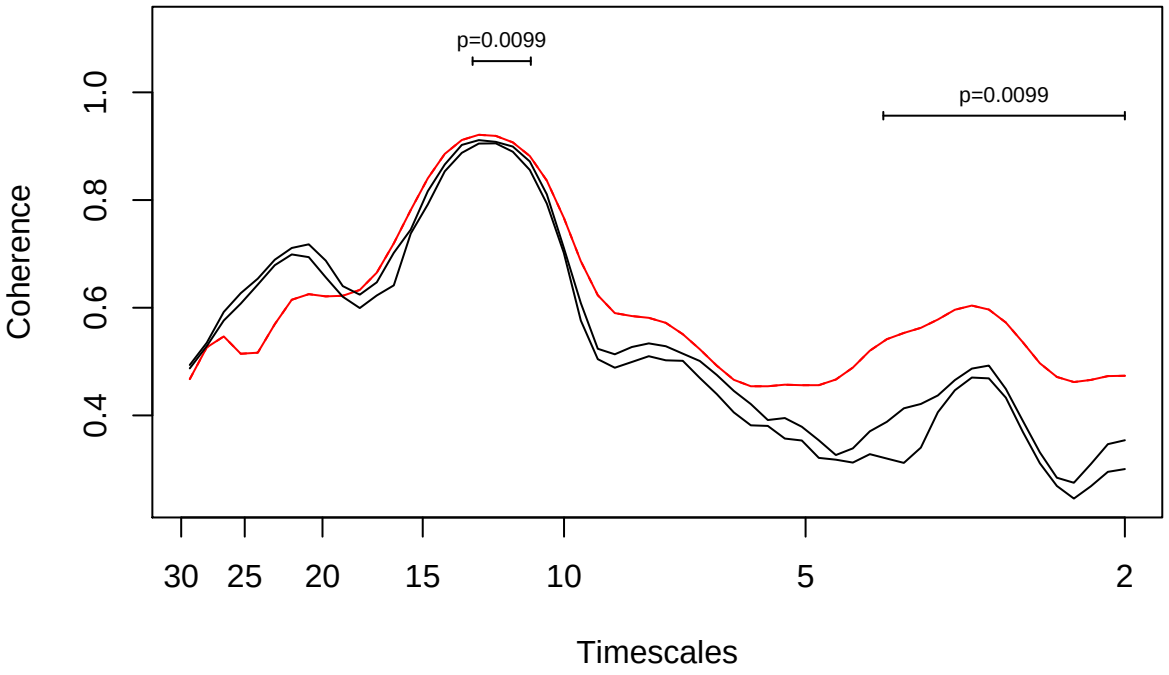



```
plotrank(wlm_all_drop1)
```

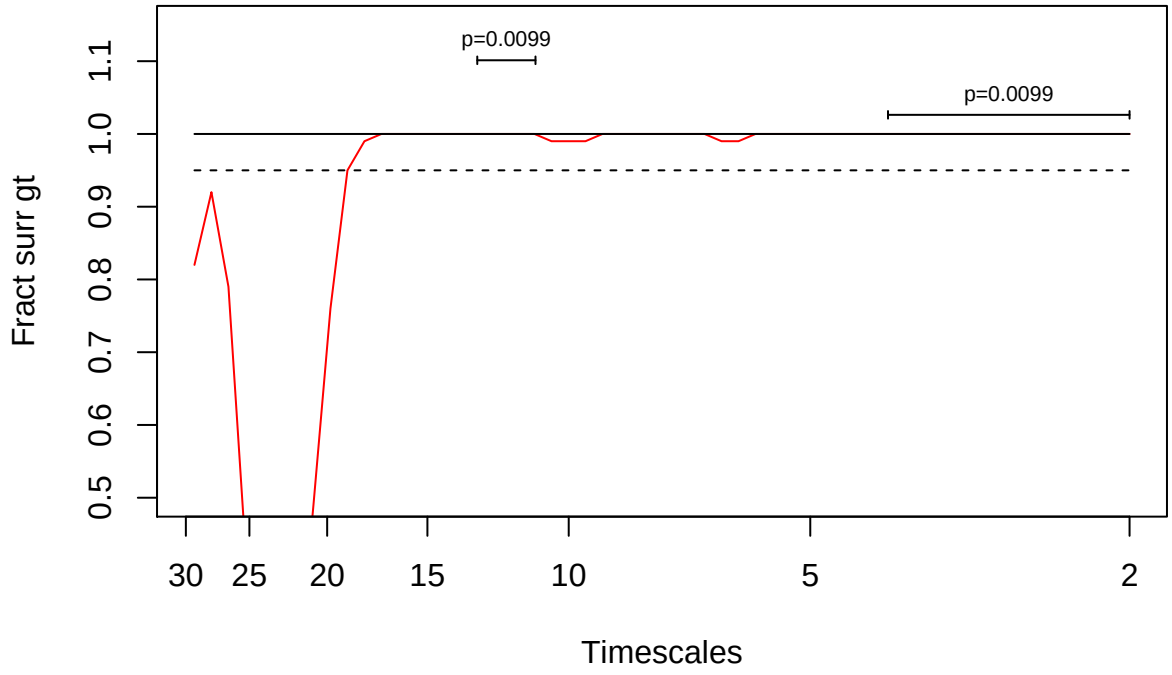


Examine results for dropping `d2`, long and short timescales. We find that `d2` should be retained in both a short-timescale model and in a long-timescale model, again as expected:

```
wlm_all_drop2<-bandtest(wlm_all_drop2,band=blong)
wlm_all_drop2<-bandtest(wlm_all_drop2,band=bshort)
plotmag(wlm_all_drop2)
```




```
plotrank(wlm_all_drop2)
```



Note that only 100 randomizations were used in this example. This is for speed - in a real analysis, at least 1000 randomizations should typically be performed, and preferably at least 10000.

6.2 Amounts of synchrony explained

Now we have constructed models for short timescales ($2-4$ years) and long timescales ($11-13$ years) for the example, finding, as expected, that `d1` is a driver at long timescales only and `d2` is a driver at short and long timescales. How much of the synchrony in the response variable is explained by these drivers for each timescale band?

For short timescales, almost all the synchrony that can be explained is explained by Moran effects of `d2`:

```
se<-syncexpl(wlm_all)
se_short<-se[se$timescales>=bshort[1] & se$timescales<=bshort[2],]
round(100*colMeans(se_short[,c(3:12)])/mean(se_short$sync),4)
```

```
##     syncexpl   crossterms       resids           d1           d2       dirrel
##      61.0564       7.0539      31.8896       0.9406      53.1484       0.2140
## interactions        d1_d2    d1_dirrel    d2_dirrel
##       6.7534       6.8652      -0.1210       0.0092
```

These are percentages of sychrony explained by various factors: `syncexpl` is total synchrony explained by the predictors for which we have data; `crossterms` must be small enough for the rest of the results to be interpretable; `d1`, `d2` and `dirrel` are percentages of sychrony explained by those predictors; `interactions` is percentage of synchrony explained by interactions between predictors (see Sheppard et al. (2019)); and the remaining terms are percentages of synchrony explained by individual interactions.

For long timescales, Moran effects of both drivers are present, as are interactions between these Moran effects:

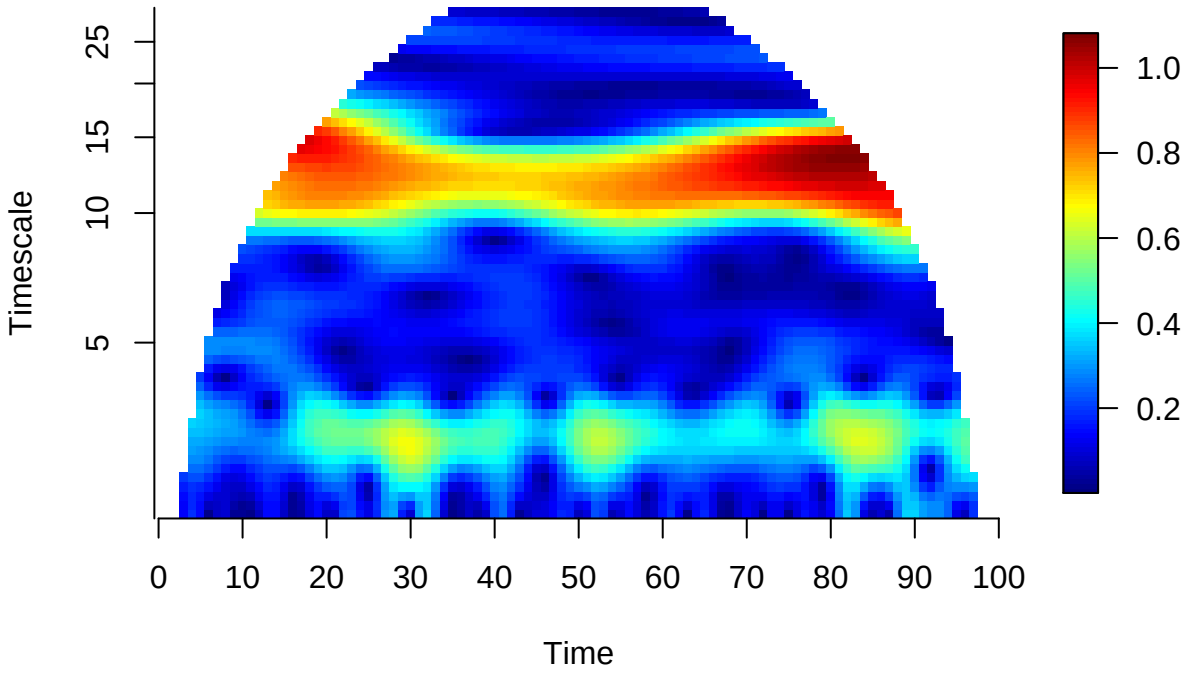
```
se_long<-se[se$timescales>=blong[1] & se$timescales<=blong[2],]
round(100*colMeans(se_long[,c(3:12)])/mean(se_long$sync),4)
```

```
##     syncexpl   crossterms       resids           d1           d2       dirrel
##      94.4137       4.8170       0.7693      25.9920      29.5607       0.0104
## interactions        d1_d2    d1_dirrel    d2_dirrel
##      38.8506      38.5144       0.2230       0.1131
```

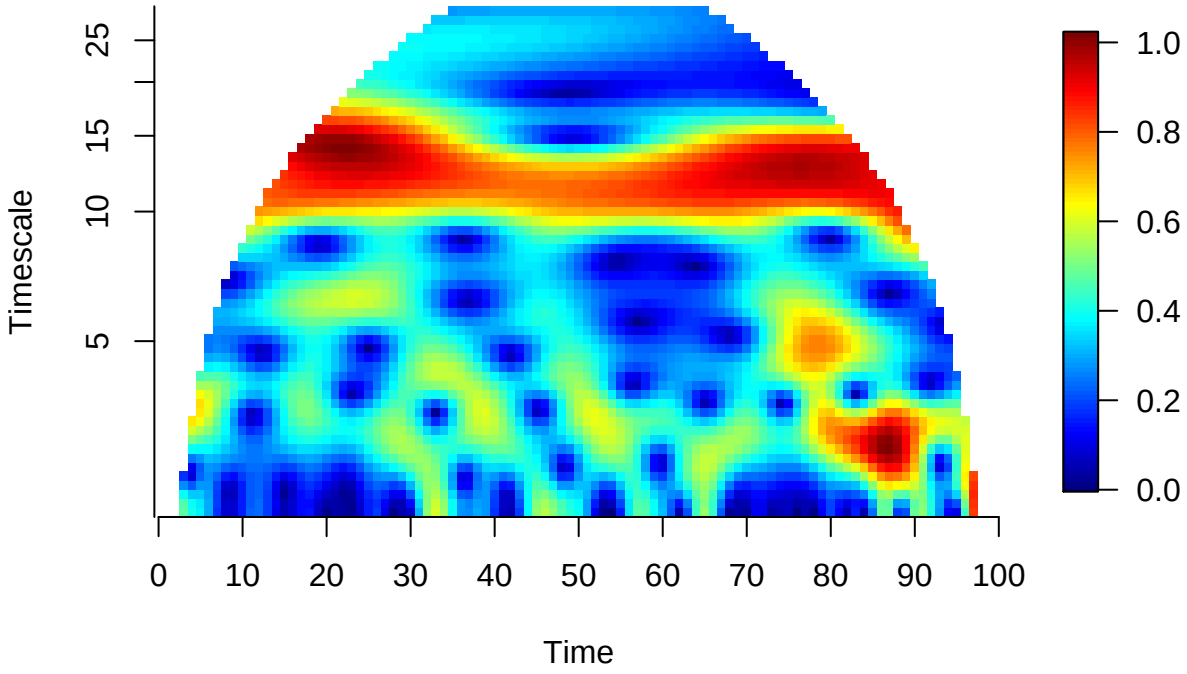
Note that cross terms are fairly small in both these analyses compared to synchrony explained. Results can only be interpreted when this is the case. See Sheppard et al. (2019) for detailed information on cross terms and interacting Moran effects.

The pattern of synchrony that would pertain if the only drivers of synchrony were those included in a model can also be produced, and compared to the actual pattern of synchrony (as represented by the wavelet mean field) to help evaluate the model.

```
pres<-predsync(wlm_all)
plotmag(pres)
```

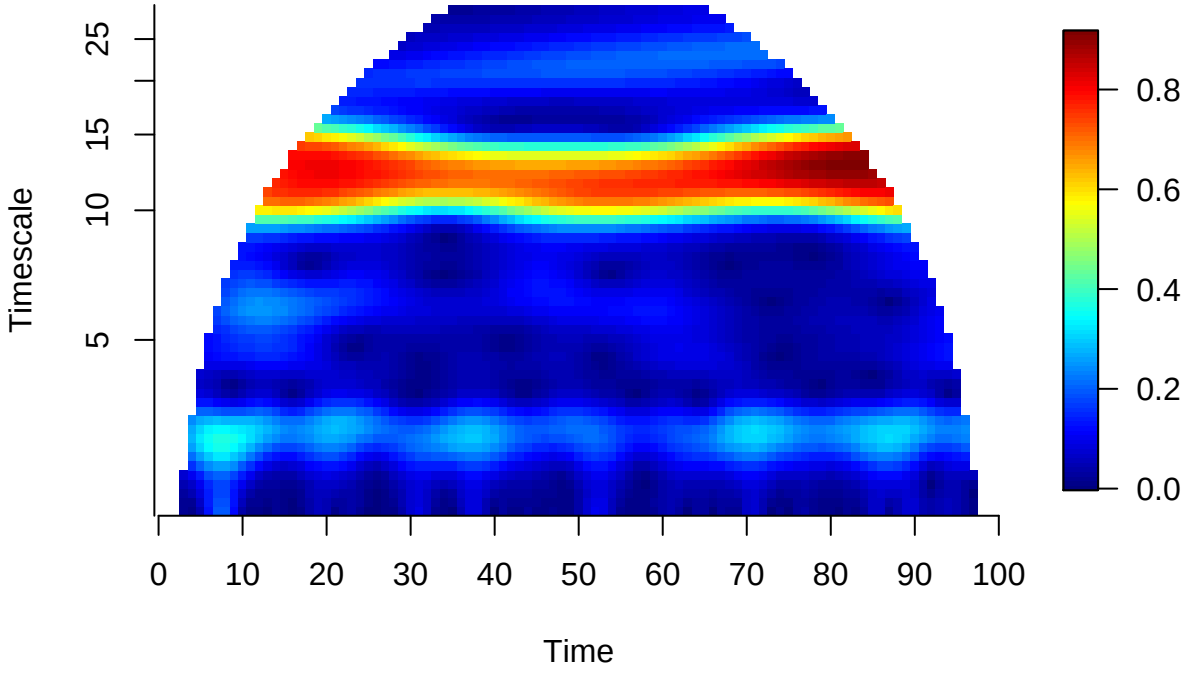


```
plotmag(wmfpop)
```



The similarity is pretty good. Now make the comparison using the model with sole predictor `d1`.

```
wlm_d1<-wlm(dat,times,resp=1,pred=2,norm="powall",scale.max.input=28)
pres<-predsync(wlm_d1)
plotmag(pres)
```



The similarity with the wavelet mean field of the populations is pretty good at long timescales (where the model with sole predictor `d1` was found to be a good model), but not at short timescales.

7 Clustering

Tools are provided in `wsyn` for separating sampling locations into network "clusters" or "modules" or "communities" (these are three alternative names used) consisting of sites that are especially synchronous with each other. Walter et al. (2017) applied this kind of approach to gypsy moth data. Given an $N \times T$ matrix of values corresponding to measurements made in N locations over T times, the approach starts by generating an $N \times N$ synchrony matrix with i, jth entry describing the strength of synchrony between the time series from locations i and j (in one of several ways - see below). This matrix is then passed to an existing clustering algorithm to partition the set of locations.

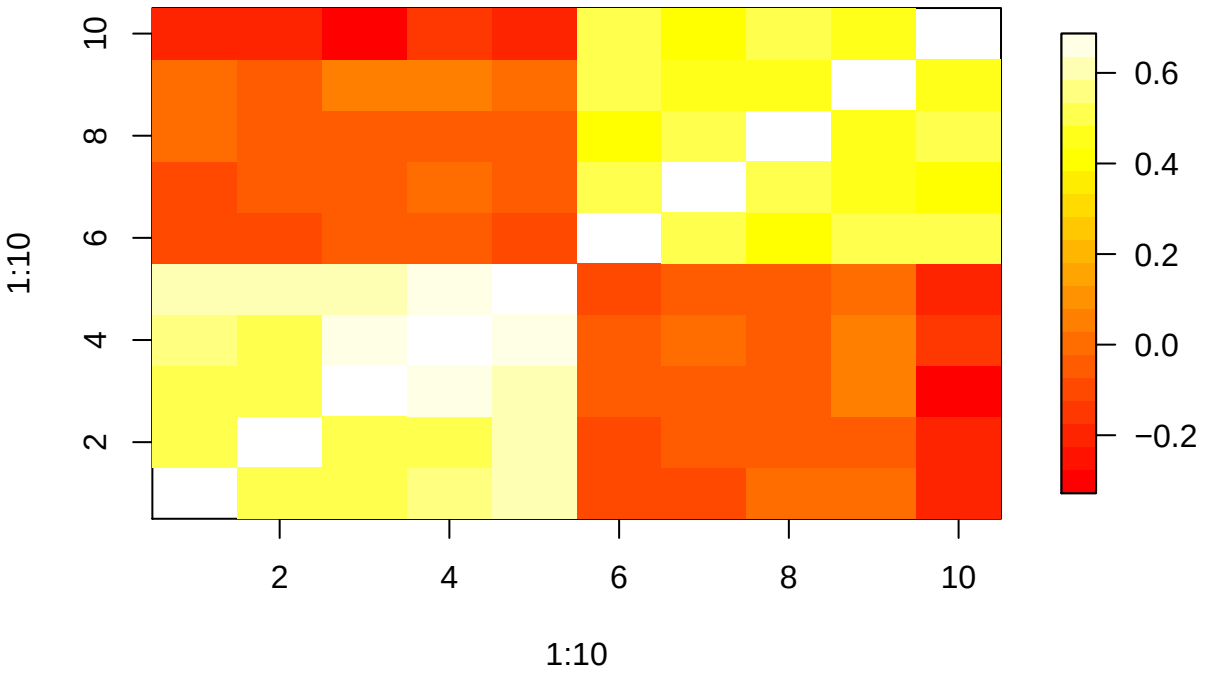
7.1 The synchrony matrix

There are numerous ways to generate a synchrony matrix, and `synmat` provides several alternatives. For an initial demonstration, create some data in two synchronous clusters.

```
N<-5
Tmax<-100
rho<-0.5
sig<-matrix(rho,N,N)
diag(sig)<-1
d<-t(cbind(mvtnorm::rmvnorm(Tmax,mean=rep(0,N),sigma=sig),
           mvtnorm::rmvnorm(Tmax,mean=rep(0,N),sigma=sig)))
d<-cleandat(d,1:Tmax,1)$cdat
```

Then make a synchrony matrix using Pearson correlation.

```
sm<-synmat(d,1:Tmax,method="pearson")
fields::image.plot(1:10,1:10,sm,col=heat.colors(20))
```



The function `synmat` provides many other options, beyond correlation, for different kinds of synchrony matrices. We demonstrate a frequency-specific approach. First create some artificial data.

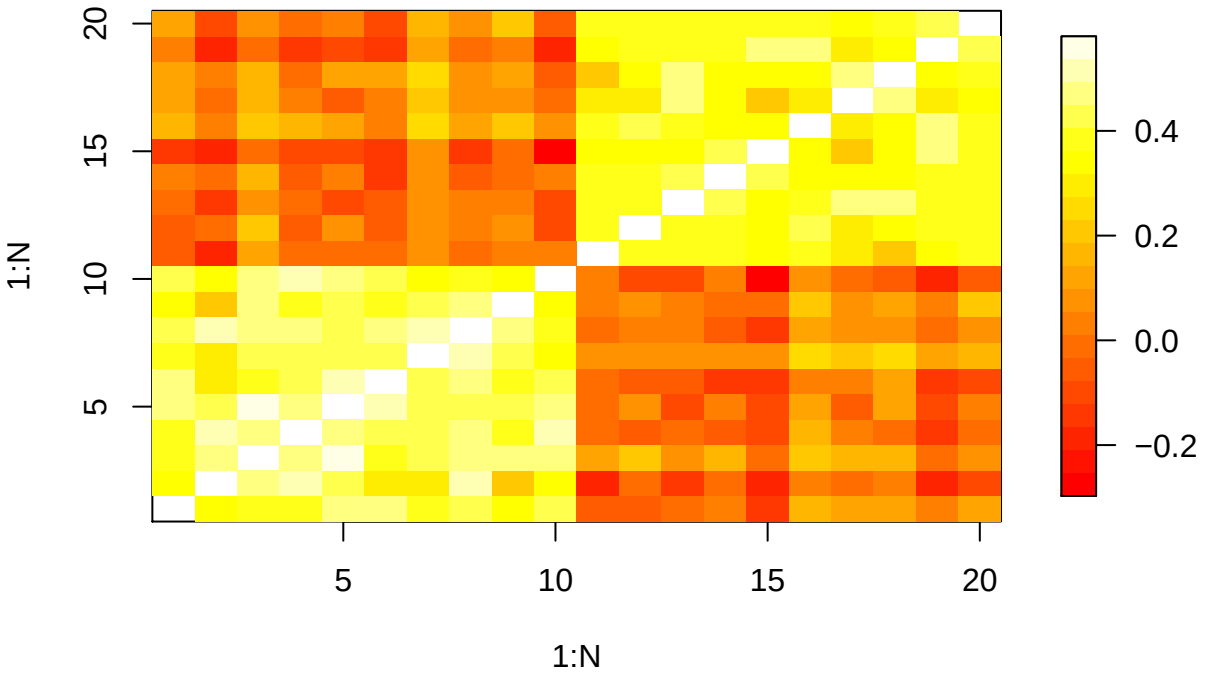
```
N<-20
Tmax<-500
```

```
tim<-1:Tmax

ts1<-sin(2*pi*tim/5)
ts1s<-sin(2*pi*tim/5+pi/2)
ts2<-sin(2*pi*tim/12)
ts2s<-sin(2*pi*tim/12+pi/2)

gp1A<-1:5
gp1B<-6:10
gp2A<-11:15
gp2B<-16:20

d<-matrix(NA,Tmax,N)
d[,c(gp1A,gp1B)]<-ts1
d[,c(gp2A,gp2B)]<-ts1s
d[,c(gp1A,gp2A)]<-d[,c(gp1A,gp2A)]+matrix(ts2,Tmax,N/2)
d[,c(gp1B,gp2B)]<-d[,c(gp1B,gp2B)]+matrix(ts2s,Tmax,N/2)
d<-d+matrix(rnorm(Tmax*N,0,2),Tmax,N)
d<-t(d)

d<-cleandat(d,1:Tmax,1)$cdat
```

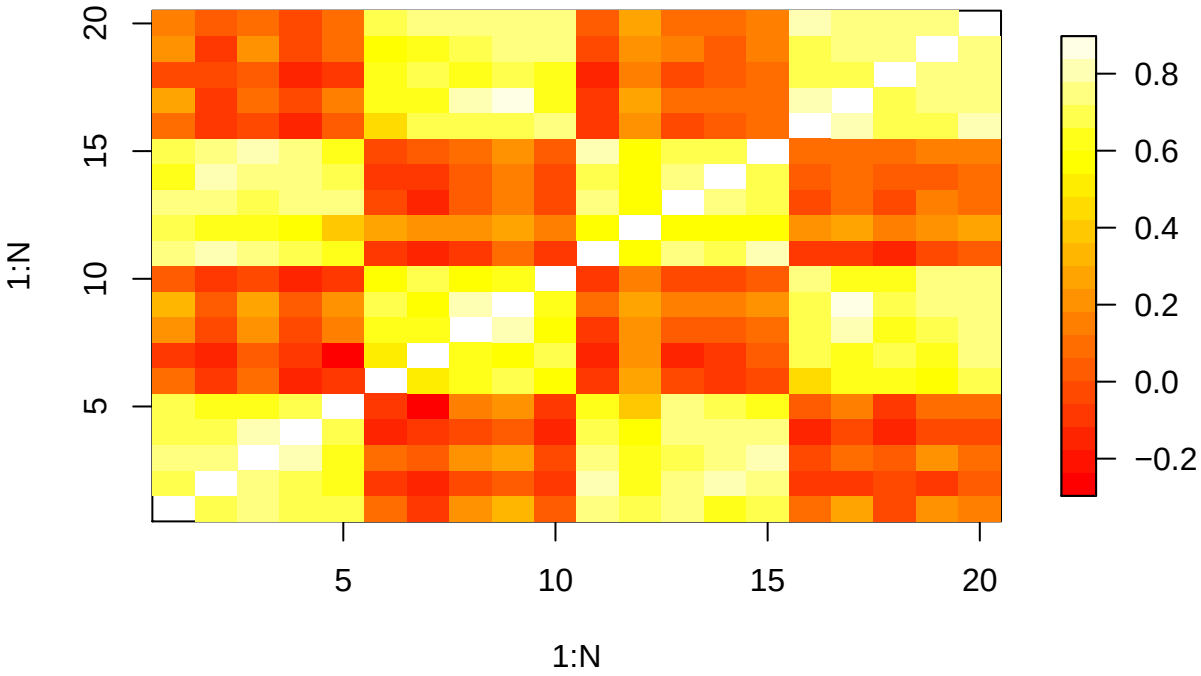
These data have period-5 oscillations which are synchronous within location groups 1 and 2, but are asynchronous between these groups. Superimposed on the period-5 oscillations are period-12 oscillations which are synchronous within location groups A and B, but are asynchronous between these groups. Groups 1 and 2 are locations $1-10$ and $11-20$, respectively. Group A is locations $1-5$ and $11-15$. Group B is locations $6-10$ and $16-20$. So the spatial structure of period-5 oscillations differs from that of period-12 oscillations. Strong local noise is superimposed on top of the periodic oscillations.

We measure synchrony matrices using portions of the the cross-wavelet transform centered on periods 5, and 12 (in separate synchrony matrices), to detect the different structures on different timescales.

```
sm5<-synmat(dat=d,times=1:Tmax,method="ReXWT",tsrange=c(4,6))
fields::image.plot(1:N,1:N,sm5,col=heat.colors(20))
```

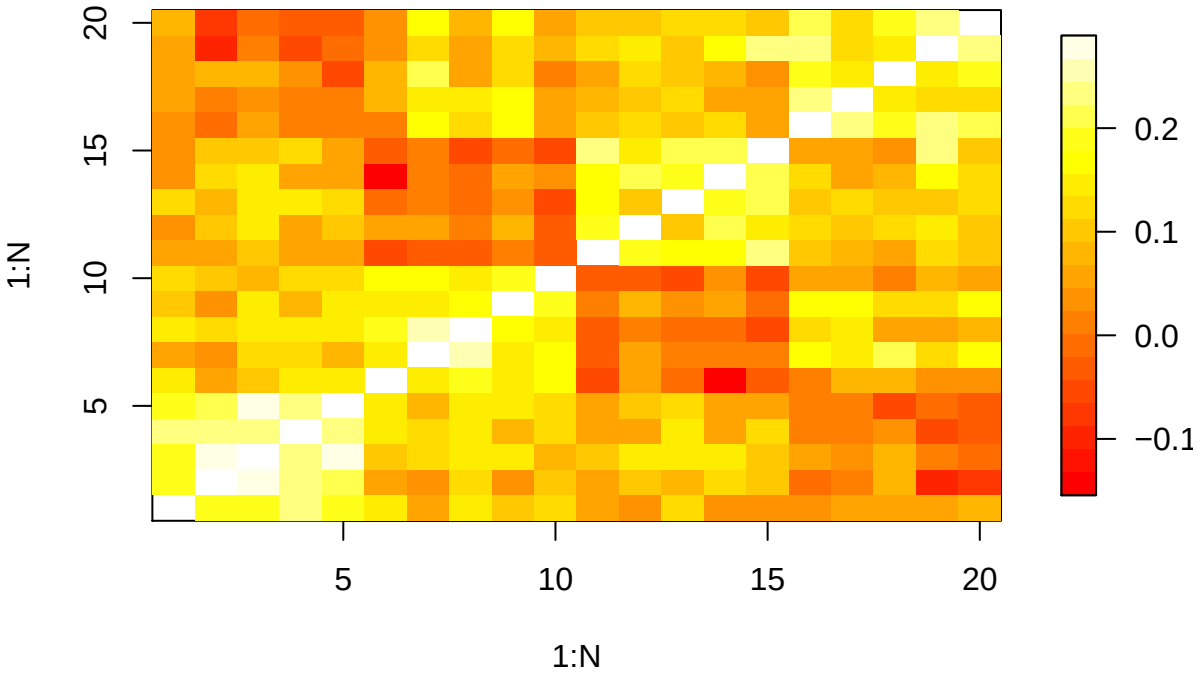


```
sm12<-synmat(dat=d,times=1:Tmax,method="ReXWT",tsrange=c(11,13))
fields::image.plot(1:N,1:N,sm12,col=heat.colors(20))
```



This timescale-specific approach reveals the structure of the data better than a correlation approach.

```
sm<-synmat(dat=d,times=1:Tmax,method="pearson")
fields::image.plot(1:N,1:N,sm,col=heat.colors(20))
```



Several additional synchrony measures with which `synmat` can construct synchrony matrices are described in the documentation of the function.

Important note: synchrony matrices can have negative values for some of the methods provided by `synmat`. This is appropriate,

since correlation and other measures of synchrony can be negative, but it complicates cluster detection (see next section).

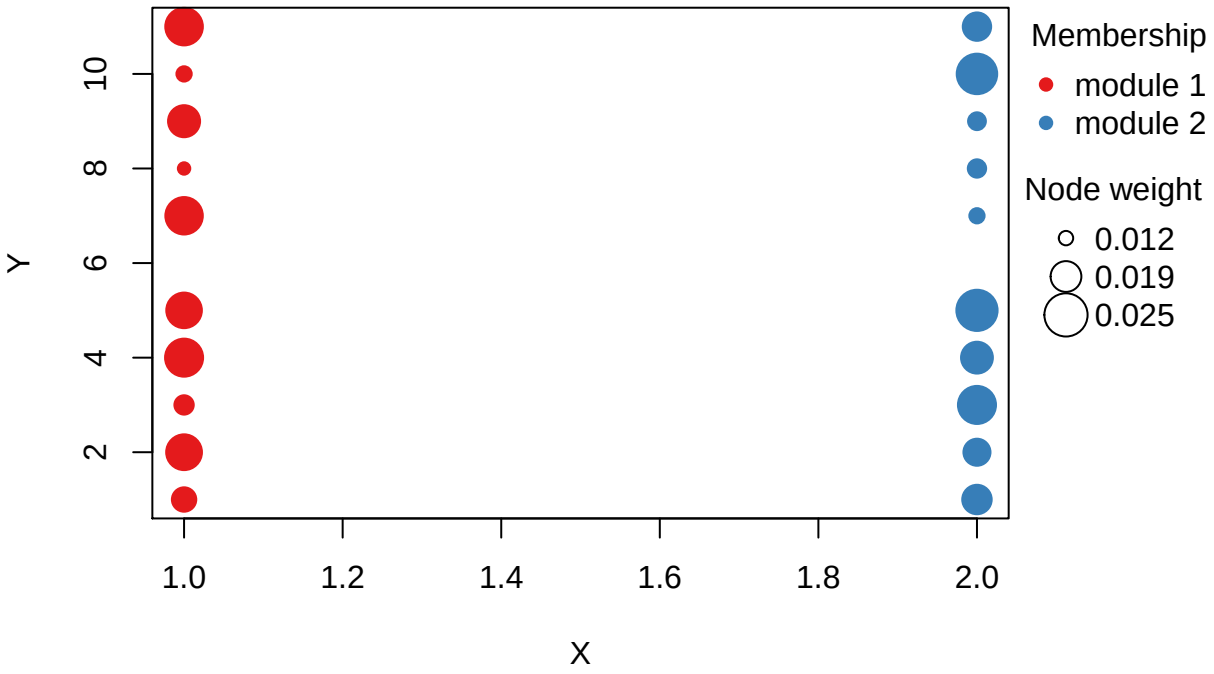
7.2 Clustering

The function `clust` computes network modules/clusters and helps keep information about them organized. That function is also the generator function for the `clust` class. The class has `print` and `summary` and `set` and `get` methods (see the help file for `clust_methods`). The clustering algorithm used is a slight adaptation of that of Newman (2006) - see the next section for details. We illustrate the use of `clust` using the artificial data from the second example of the previous section.

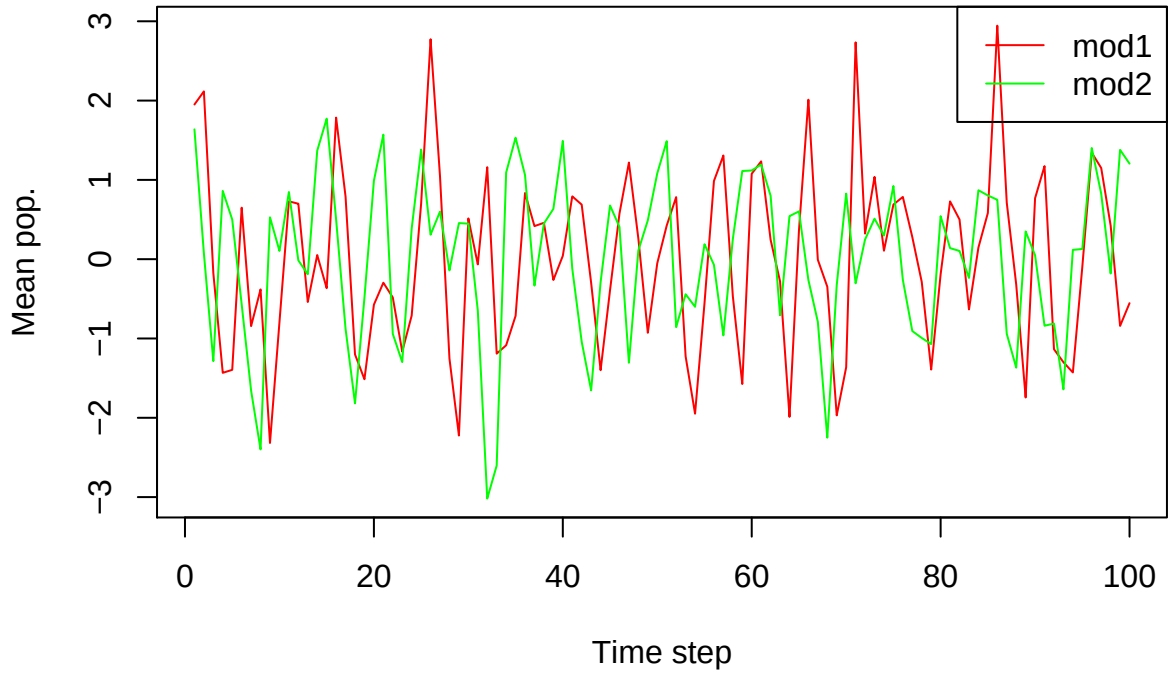
```
#make some artificial coordinates for the geographic locations of where data were measured
coords<-data.frame(X=c(rep(1,10),rep(2,10)),Y=rep(c(1:5,7:11),times=2))

#create clusters based on the 5-year timescale range and map them using coords
cl5<-clust(dat=d,times=1:Tmax,coords=coords,method="ReXWT",tsrange=c(4,6))
get_clusters(cl5) #the first element of the list is always all 1s - prior to any splits
```

```
## [[1]]
##  [1] 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
##
## [[2]]
##  [1] 1 1 1 1 1 1 1 1 1 1 2 2 2 2 2 2 2 2 2 2
```

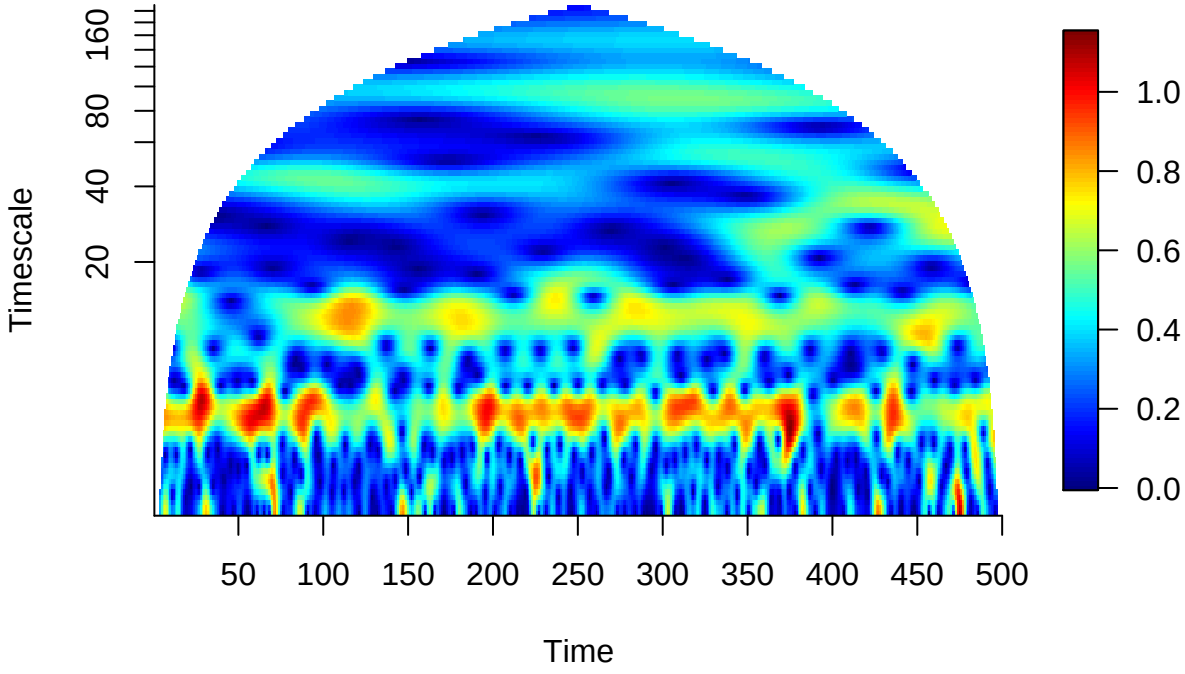
```
#call the mapper here
plotmap(cl5)
```



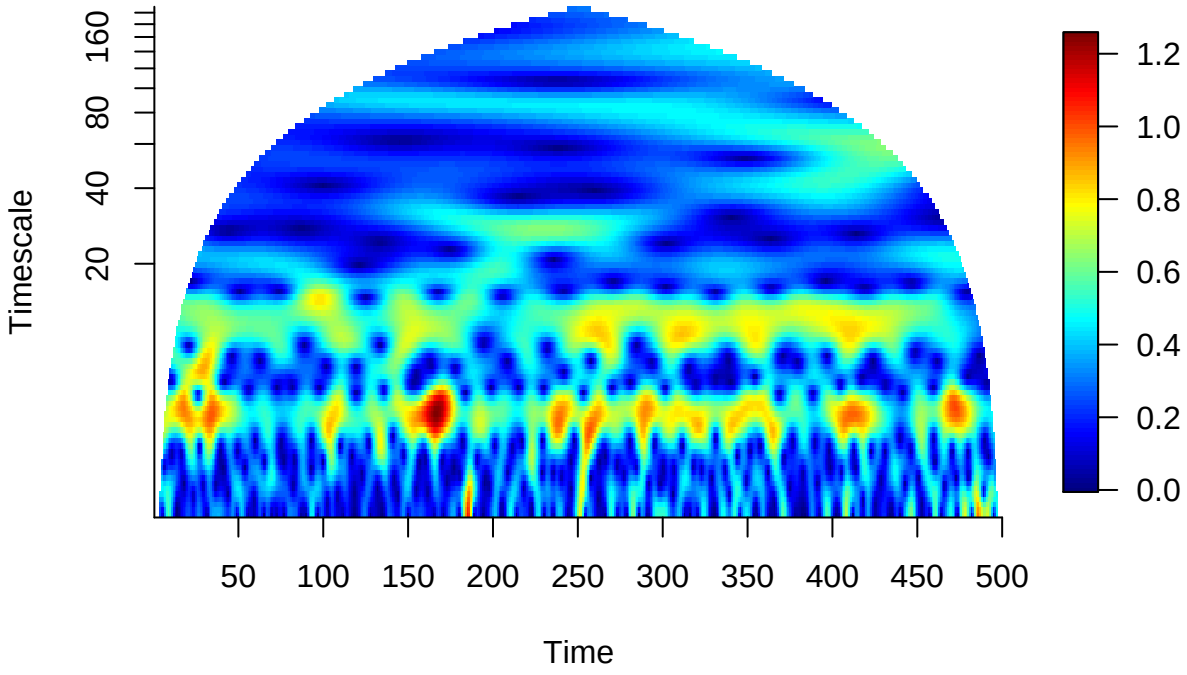

```
#plot mean time series for each module
plot(get_times(cl5)[1:100],get_mns(cl5)[[2]][1,1:100],type='l',col='red',
     ylim=range(get_mns(cl5)),xlab="Time step",ylab="Mean pop.")
lines(get_times(cl5)[1:100],get_mns(cl5)[[2]][2,1:100],type='l',col='green')
legend(x="topright",legend=c("mod1","mod2"),lty=c(1,1),col=c("red","green"))
```




```
#create wavelet mean fields for each module and plot
cl5<-addwmfs(cl5)
plotmag(get_wmfs(cl5)[[2]][[1]])
```



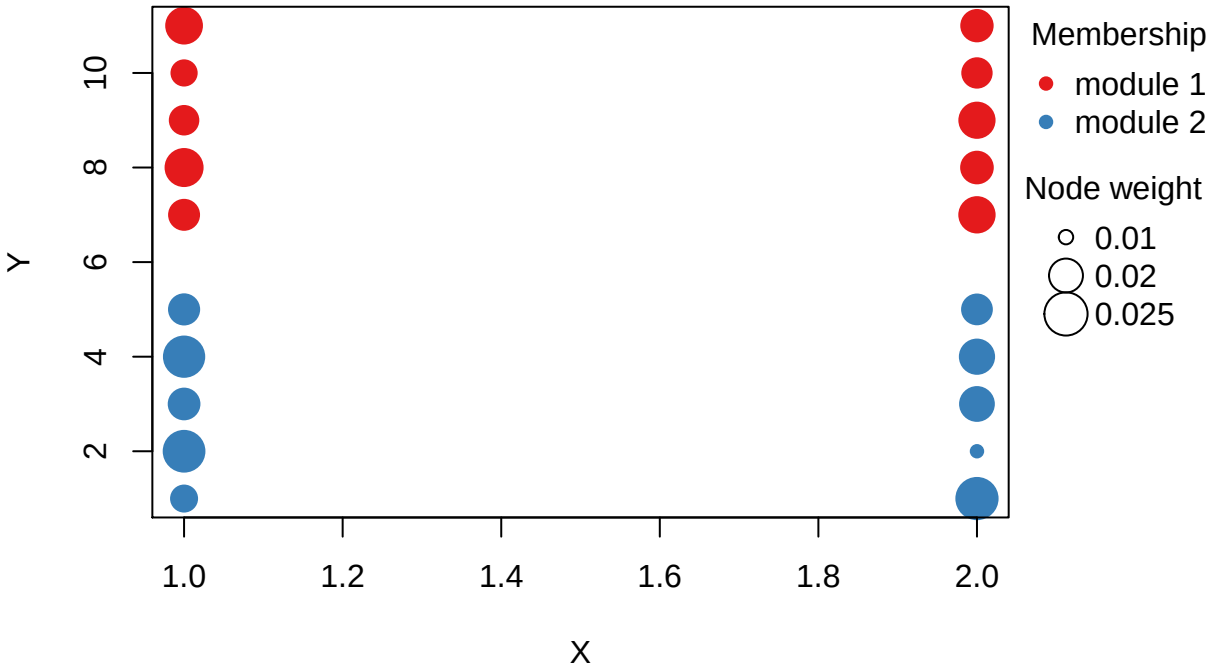

```
plotmag(get_wmfs(cl5)[[2]][[2]])
```



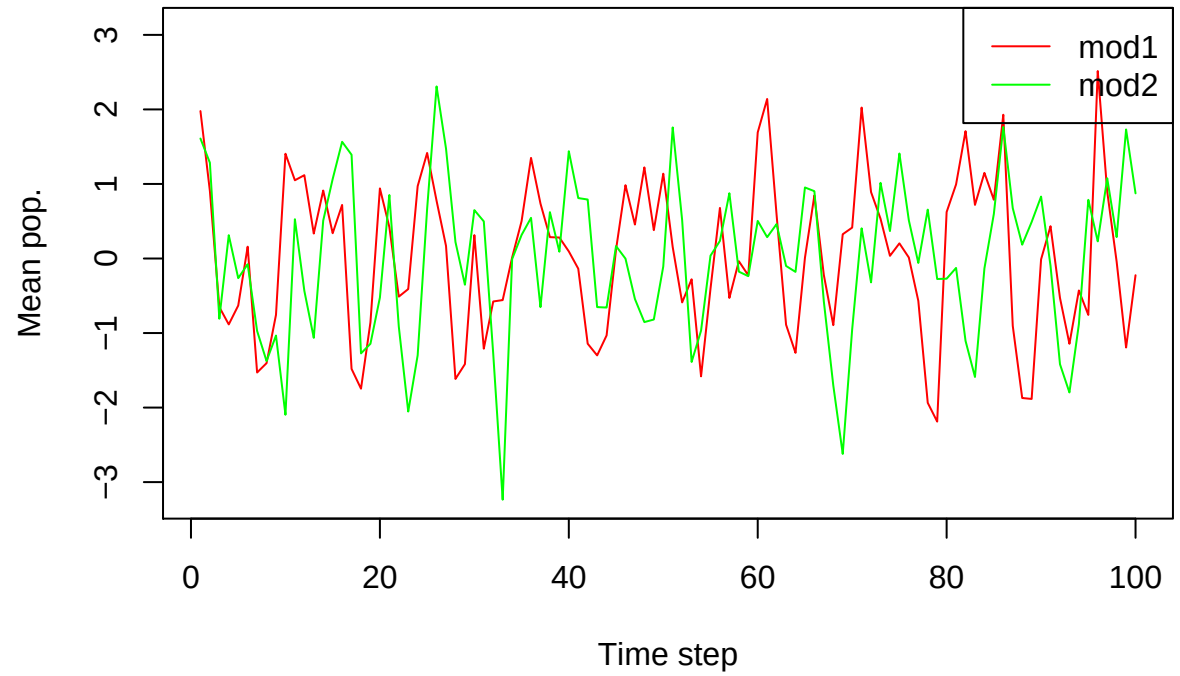

```
#create clusters based on the 12-year timescale range and map them using coords
cl12<-clust(dat=d,times=1:Tmax,coords=coords,method="ReXWT",tsrange=c(11,13))
cl12$clusters
```

```
## [[1]]
##  [1] 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
##
## [[2]]
##  [1] 2 2 2 2 2 1 1 1 1 1 2 2 2 2 2 1 1 1 1 1
```

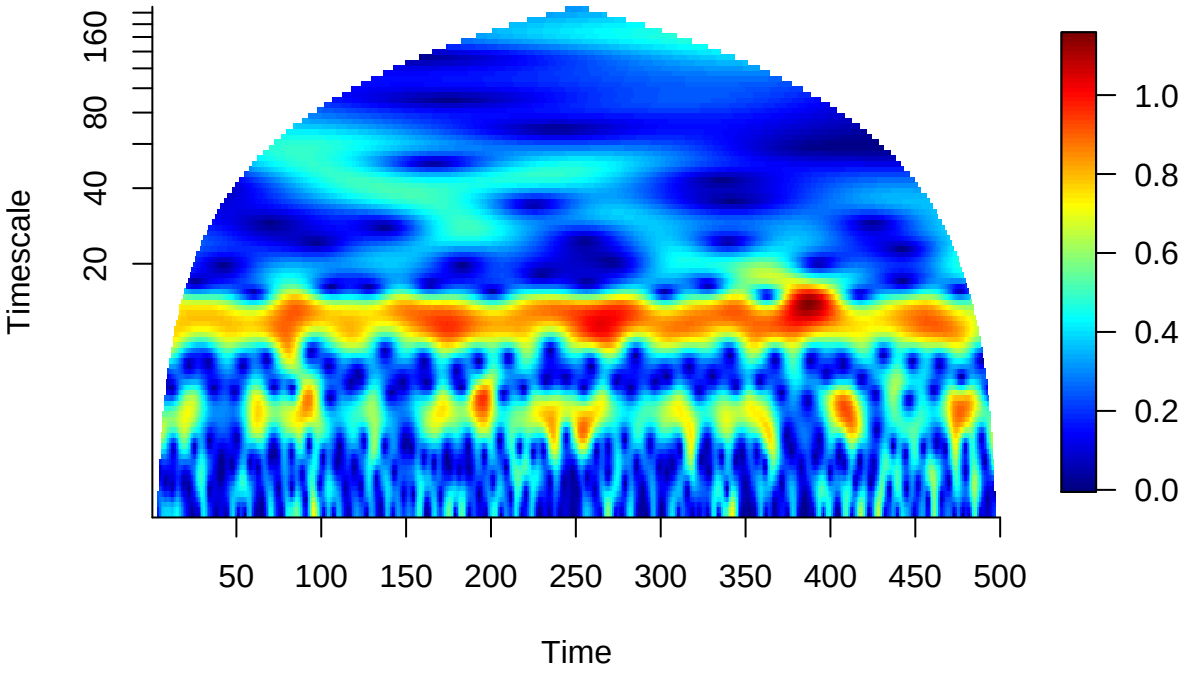
```
#call the mapper here
plotmap(cl12)
```



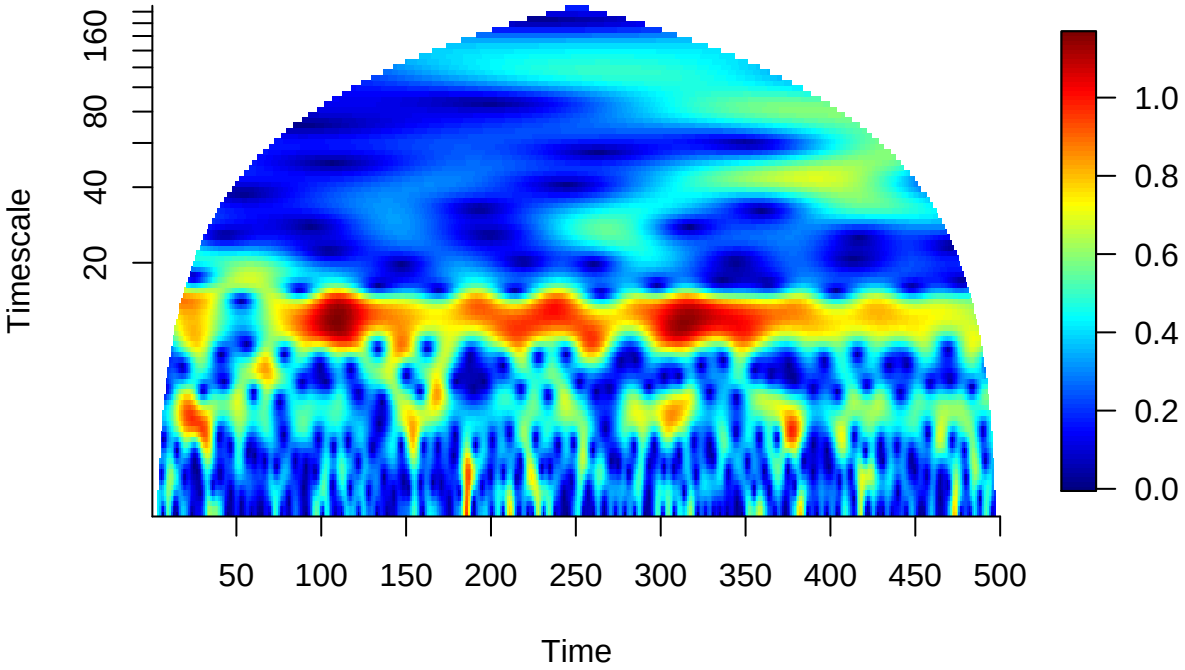

```
#plot mean time series for each module
plot(get_times(cl12)[1:100],get_mns(cl12)[[2]][1,1:100],type='l',col='red',
     ylim=range(get_mns(cl12)),xlab="Time step",ylab="Mean pop.")
lines(get_times(cl12)[1:100],get_mns(cl12)[[2]][2,1:100],type='l',col='green')
legend(x="topright",legend=c("mod1","mod2"),lty=c(1,1),col=c("red","green"))
```




```
#create wavelet mean fields for each module and plot
cl12<-addwmfs(cl12)
plotmag(get_wmfs(cl12)[[2]][[1]])
```




```
plotmag(get_wmfs(cl12)[[2]][[2]])
```



Color intensity on maps of clusters indicates the strength of contribution of a node to its module - details in the next section. The function `addwpmfs` is similar to the function `addwmfs` demonstrated above, but adds wavelet phasor mean field information to a `clust` object.

7.3 The clustering algorithm

The clustering algorithm used by `clust` is implemented in `cluseigen`, and is a generalization of the algorithm of Newman (2006). The algorithm makes use of the concept of modularity. Modularity was defined by Newman (2006) for any unweighted, undirected

network paired with a partitioning of the nodes into modules. The modularity is then a single numeric score which is higher for better partitionings, i.e., for partitionings that more effectively group nodes that are more heavily connected and separate nodes that are less connected. The ideal goal would be to find the partitioning that maximizes the modularity score, but this is computationally infeasible for realistically large networks (Newman 2006). Instead, the algorithm of Newman (2006), which is computationally very efficient, provides a good partitioning that is not guaranteed to be optimal but is typically close to optimal (Newman 2006). The modularity itself can be computed rapidly, given a partitioning, with the function `modularity` in `wsyn`.

Modularity is defined in Newman (2006) for unweighted networks, and the definition generalizes straightforwardly to weighted networks for which all weights are non-negative. But synchrony matrices can represent weighted networks for which some weights are allowed to be negative. A generalization of modularity for this more general type of network was defined by Gomez, Jensen, and Arenas (2009), and the `modularity` function computes this generalization. Values are the same as the definition of Newman (2006) for non-negatively weighted networks. The algorithm implemented by `cluseigen` is a slight generalization of the original Newman (2006) algorithm that applies to weighted networks for which some edges are allowed to have negative weight.

We here describe the generalized algorithm, the validity of which was realized by Lei Zhao. Let w_{ij} be the adjacency matrix for a network, so the ijth entry of this matrix is the weight of the edge between nodes i and j, 0 if there is no edge. Let C_i be the community/module to which node i is assigned and let C_j be the same for node j. The original definition of modularity, for non-negative w_{ij}, is

$$Q = \frac{1}{2w}\sum_{ij}\left(w_{ij} - \frac{w_i w_j}{2w}\right)\delta(C_i, C_j),$$

where $w_i = \sum_j w_{ij}$, $w = \frac{1}{2}\sum_i w_i$, and δ is the Kronecker delta function, equal to 1 when $C_i = C_j$ and 0 otherwise.

For the case of partitioning into two clusters, Newman (2006) notes that $\delta(C_i, C_j) = \frac{1}{2}(s_i s_j + 1)$, where we define s_i to be 1 if node i is in group 1 and -1 if it is in group 2. Then

$$Q = \frac{1}{4w}\sum_{ij}\left(w_{ij} - \frac{w_i w_j}{2w}\right)(s_i s_j + 1),$$

and it is easy to show this is

$$Q = \frac{1}{4w}\sum_{ij}\left(w_{ij} - \frac{w_i w_j}{2w}\right)s_i s_j.$$

Defining a matrix $\mathbf{B}$ such that $B_{ij} = w_{ij} - \frac{w_i w_j}{2w}$, we have

$$Q = \frac{1}{4w}\mathbf{s}^\tau \mathbf{B}\mathbf{s},$$

where τ is transpose and the bold quantities are the vectors/matrices composed of the indexed quantities denoted with the same symbol. Newman (2006) presents an argument that a good way to come close to optimizing Q over possible partitions into two modules is to choose s_i to be the same sign as the ith entry of the leading eigenvector of $\mathbf{B}$, if the leading eigenvalue is postive (otherwise the algorithm halts with no splits, returning the trivial "decomposition" into one module). This gives the first split, according to the Newman algorithm, of the network into modules. Subsequent splits are handled in a similar but not identical way described by Newman (2006). (In particular it is incorrect to simply delete edges connecting the two modules from the first split and apply the algorithm to the resulting graphs.)

The generalized modularity of Gomez, Jensen, and Arenas (2009) is defined as follows. Let $w_{ij}^+ = \max(0, w_{ij})$ and $w_{ij}^- = \max(0, -w_{ij})$ so that $w_{ij} = w_{ij}^+ - w_{ij}^-$. Let $w_i^+ = \sum_j w_{ij}^+$, $w_i^- = \sum_j w_{ij}^-$, $w^+ = \frac{1}{2}\sum_i w_i^+$, and $w^- = \frac{1}{2}\sum_i w_i^-$. Then Gomez, Jensen, and Arenas (2009) justifies the definitions

$$Q^+ = \frac{1}{2w^+}\sum_{ij}\left(w_{ij}^+ - \frac{w_i^+ w_j^+}{2w^+}\right)\delta(C_i, C_j),$$

$$Q^- = \frac{1}{2w^-}\sum_{ij}\left(w_{ij}^- - \frac{w_i^- w_j^-}{2w^-}\right)\delta(C_i, C_j),$$

and

$$Q = \frac{2w^+}{2w^+ + 2w^-}Q^+ - \frac{2w^-}{2w^+ + 2w^-}Q^-.$$

This is a generalization of the old definition of Q, in the sense that it reduces to that definition in the case of non-negatively weighted networks. Gomez, Jensen, and Arenas (2009) provides a probabilistic interpretation that generalizes the probabilisticc interpretation of Newman (2006).

It is straightforward to show

$$Q = \frac{1}{2w^+ + 2w^-}\sum_{ij}\left(w_{ij} - \left(\frac{w_i^+ w_j^+}{2w^+} - \frac{w_i^- w_j^-}{2w^-}\right)\right)\delta(C_i, C_j).$$

Again considering an initial 2-module split and define s_i and s_j as previously, we have

$$Q = \frac{1}{4w^+ + 4w^-}\sum_{ij}\left(w_{ij} - \left(\frac{w_i^+ w_j^+}{2w^+} - \frac{w_i^- w_j^-}{2w^-}\right)\right)s_i s_j,$$

which can be written in matrix form as

$$Q = \frac{1}{4w^+ + 4w^-}\mathbf{s}^\tau \mathbf{E}\mathbf{s}.$$

Because the generalized modularity of Gomez, Jensen, and Arenas (2009) can be written in the same matrix format as the modularity expression of Newman (2006), the same eigenvector-based algorithm for finding a close-to-optimal value of the modularity can be used.

The quantity

$$\frac{1}{2w^+ + 2w^-}\sum_{j}\left(w_{ij} - \left(\frac{w_i^+ w_j^+}{2w^+} - \frac{w_i^- w_j^-}{2w^-}\right)\right)\delta(C_i, C_j)$$

is the contribution of node i to the modularity. It is the extent to which node i is more connected to other nodes in its module than expected by chance (Newman 2006; Gomez, Jensen, and Arenas 2009), and can be interpreted as a strength of membership of a node in its module. The `plotmap` function (demonstrated above) has an option for coloring nodes according to this quantity.

8 Acknowlegements

This material is based upon work supported by the National Science Foundation under grant numbers 17114195 and 1442595, and by the James S McDonnell Foundation. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the authors and do not necessarily reflect the views of the National Science Foundation or the McDonnell Foundation. We thank all users of the package who have reported or will later report ways in which the package could be improved.

References

Addison, PS. 2002. *The Illustrated Wavelet Transform Handbook: Introductory Theory and Applications in Science, Engineering, Medicine and Finance*. New York: Taylor; Francis.

Anderson, TL, LW Sheppard, JA Walter, TL Levine, SP Hendricks, DS White, and Reuman DC. 2018. "The Dependence of Synchrony on Timescale and Geography in Freshwater Plankton." *Limnology and Oceanography* In press.

Gomez, S, P Jensen, and A Arenas. 2009. "Analysis of Community Structure in Networks of Correlated Data." *Physical Review E* 80: 016114.

Grenfell, BT, ON Bjørnstad, and J Kappey. 2001. "Traveling Waves and Spatial Hierarchies in Measles Epidemics." *Nature* 414:

716–23.

Keitt, TH. 2008. “Coherent Ecological Dynamics Induced by Large-Scale Disturbance.” *Nature* 454: 331–35.

Liebhold, A., W.D. Koenig, and O. Bjørnstad. 2004. “Spatial Synchrony in Population Dynamics.” *Annual Review of Ecology Evolution and Systematics* 35: 467–90.

Newman, MEJ. 2006. “Modularity and Community Structure in Networks,” no. 103: 8577–82.

Prichard, D, and J Theiler. 1994. “Generating Surrogate Data for Time Series with Several Simultaneously Measured Variables.” *Physical Review Letters* 73 (7): 951–54.

Schreiber, T, and A Schmitz. 2000. “Surrogate Time Series.” *Physica D* 142: 346–82.

Sheppard, LW, J Bell, R Harrington, and DC Reuman. 2016. “Changes in Large-Scale Climate Alter Spatial Synchrony of Aphid Pests.” *Nature Climate Change* 6: 610–13.

Sheppard, LW, EJ Defriez, PC Reid, and DC Reuman. 2019. “Synchrony Is More Than Its Top-down and Climatic Parts: Interacting Moran Effects on Phytoplankton in British Seas.” *PLos Computational Biology* 15: e1006744.

Sheppard, LW, PC Reid, and DC Reuman. 2017. “Rapid Surrogate Testing of Wavelet Coherences.” *European Physical Journal, Nonlinear and Biomedical Physics* 5: 1.

Viboud, C, ON Bjørnstad, DL Smith, L Simonsen, MA Miller, and BT Grenfell. 2006. “Synchrony, Waves, and Spatial Hierarchies in the Spread of Influenza.” *Science* 312: 447–51.

Walter, JA, LW Sheppard, TL Anderson, JH Kastens, ON Bjornstad, AM Liebhold, and DC Reuman. 2017. “The Geography of Spatial Synchrony.” *Ecology Letters* 20: 801–14.