Spectral analysis

Daniel Buscombe

Partitioning of a signal, which is any time (or spatial) series, into components. Those components are defined in terms of their duration (or length). The amount of variation in the signal is computed at those different durations or lengths.

This is done by considering the signal to be the sum of many simpler time series that have the form of regular sinusoids that have different amplitudes and wavelengths.

These sinusoids are calculated so that they are orthogonal (just like in principal components). In fact, spectral analysis is like PCA, but instead of finding out what the form of the orthogonal basis functions are, as in PCA, in spectral analysis those functions are assumed (i.e. they are sinusoids).

The sum of all the sinusoids is equal to the original series, so spectral analysis is the process that works out how many sinusoids, at what wavelengths, are required recreate the signal. It does this in such a way that the total variance of all the sinusoids equal the variance in the signal.

Spectral analysis is also known as harmonic analysis, Fourier analysis, and frequency analysis. Examination of the frequency components of a time series can provide a valuable insight into the behaviour of the process that generated the sequence.

Periodic Signals

First, let's create a sinusoid and make some definitions

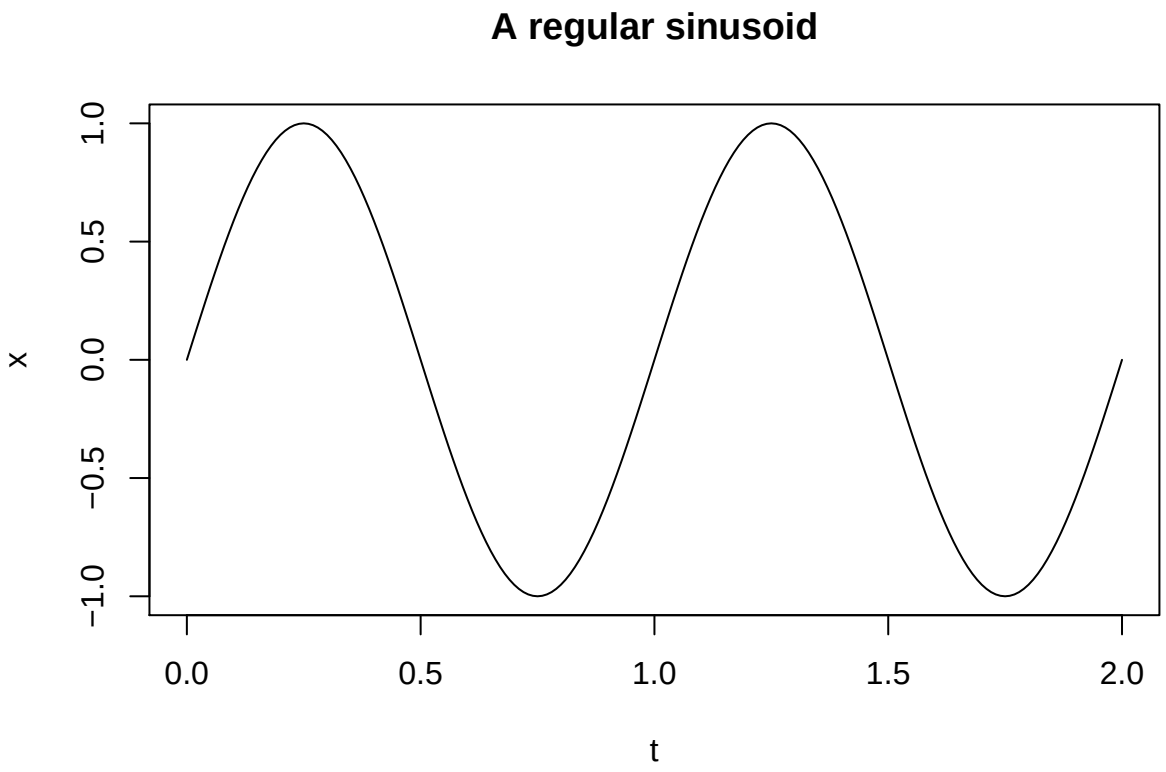
Let's remind ourselves what a sinusoid is

$x(t) = A\sin(\omega t)$

A is amplitude

ω is angular frequency in radians (2πf)

```
# Sampling frequency - this is the number of points you will use to define your sinusoid
Fs = 100

# Time vector of 2 second
t = seq(from=0, to=2, by=1/Fs)

# sinusoid function
x =sin(2*pi*t)

#make a simple line plot
plot(t,x, 'l', main='A regular sinusoid')
```



Wavelength/Period

Distance between a point on 1 wave and the equivalent point on the next wave

Wavelength is the distance in space or time, whereas period is the length of time or space for the same point on the wave to pass a fixed reference point

```
peaks = which(x==max(x))
peaks
```

```
## [1]  26 126
```

```
lambda = t[peaks[2]] - t[peaks[1]]
lambda
```

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

Frequency

Number of wave forms per unit time (1/wavelength)

```
f = 1/lambda
f
```

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

```
#number of peaks (x==1) per time (length of time)
f = length(which(x==1)) / max(t)
f
```

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

Amplitude

Half the height of a signal from trough to crest

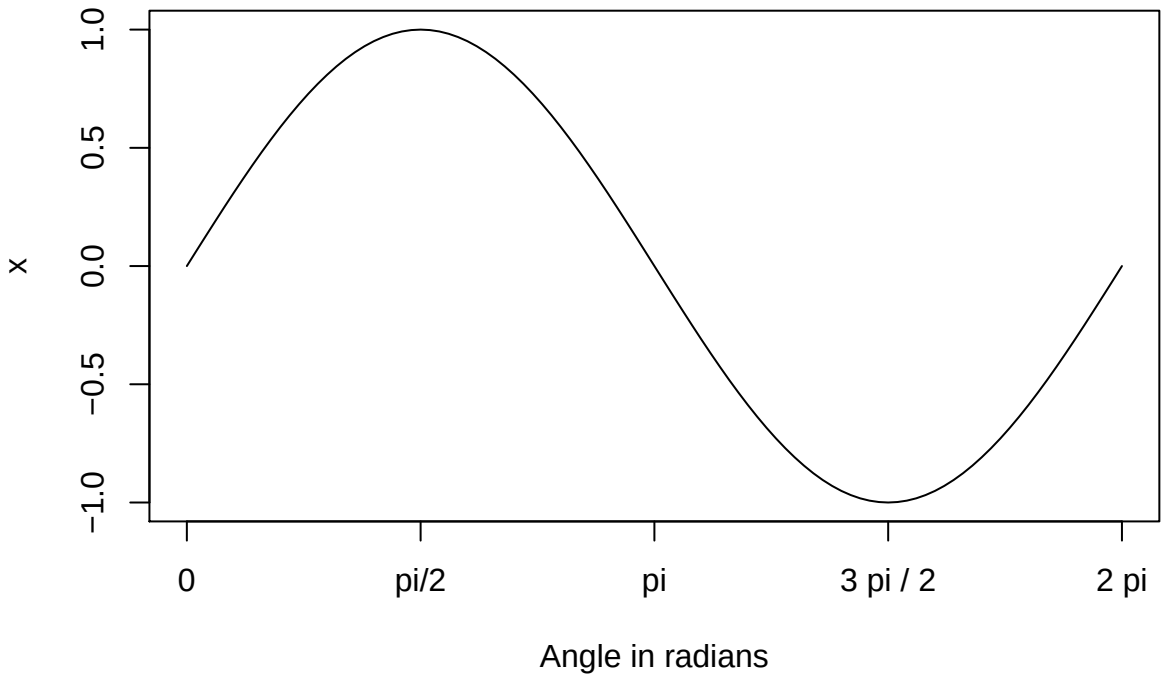
Height of a detrended signal above zero

```
(max(x) - min(x)) /2
```

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

Phase

```
# Time vector of 1 second
t = seq(from=0, to=1, by=1/Fs)

# sinusoid function
x =sin(2*pi*t)

labs = c(0,pi/2,pi,(3*pi)/2,2*pi)

#make a simple line plot and suppress x axis ticks (xaxt)
plot(t*2*pi,x, 'l', xlab='Angle in radians', xaxt = 'n')
axis(1,at=labs,labels=c("0","pi/2","pi","3 pi / 2","2 pi"))
```



We can modify our sinusoid to have phase

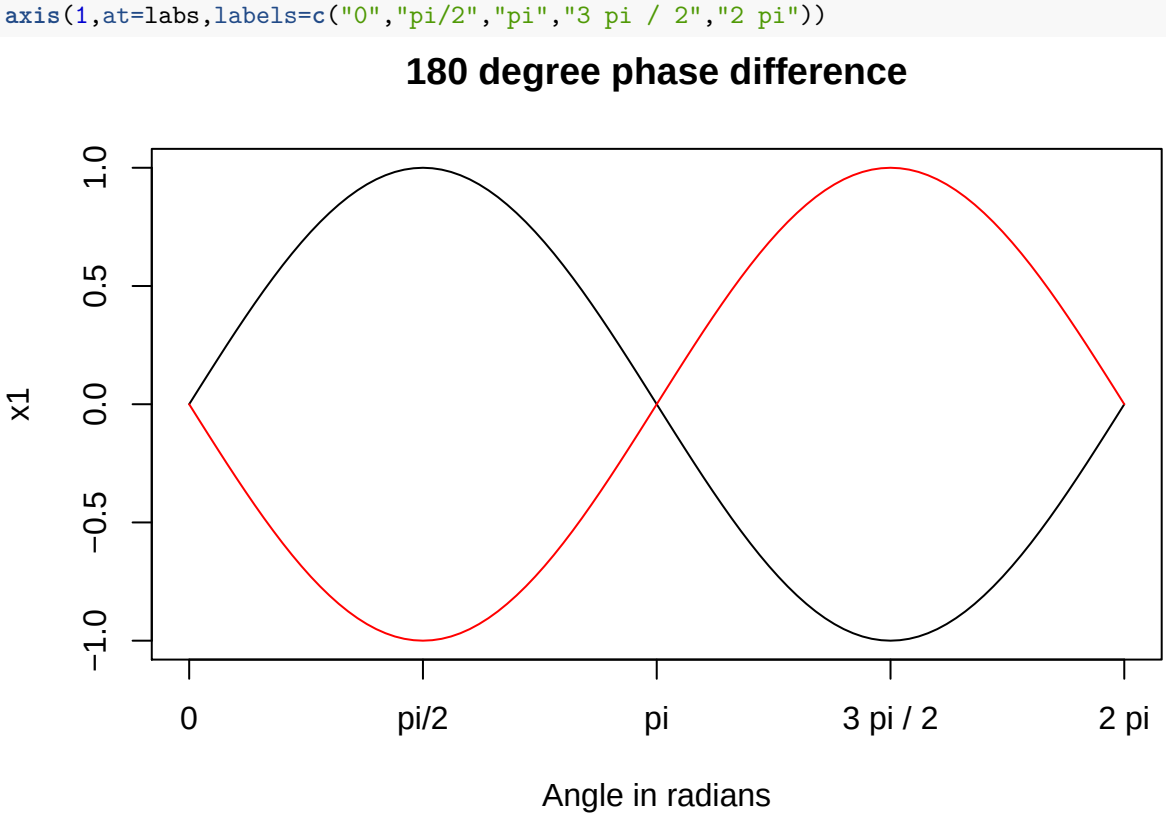
$x(t) = A\sin(\omega t + \phi)$

ϕ is phase in radians

(i.e. the amplitude, wavelength and phase completely describe the waveform)

A 180 degree phase difference would mean $\phi = \pi$

```
#phase difference
pd = pi

x1 = 1*sin(2*pi*t)

# sinusoid function with phase shift
x2 = 1*sin(2*pi*t+pd)

labs = c(0,pi/2,pi,(3*pi)/2,2*pi)

#make a simple line plot and suppress x axis ticks (xaxt)
plot(t*2*pi, x1, 'l', xlab='Angle in radians', xaxt = 'n', main='180 degree phase difference')
lines(t*2*pi, x2, col='red')
axis(1,at=labs,labels=c("0","pi/2","pi","3 pi / 2","2 pi"))
```



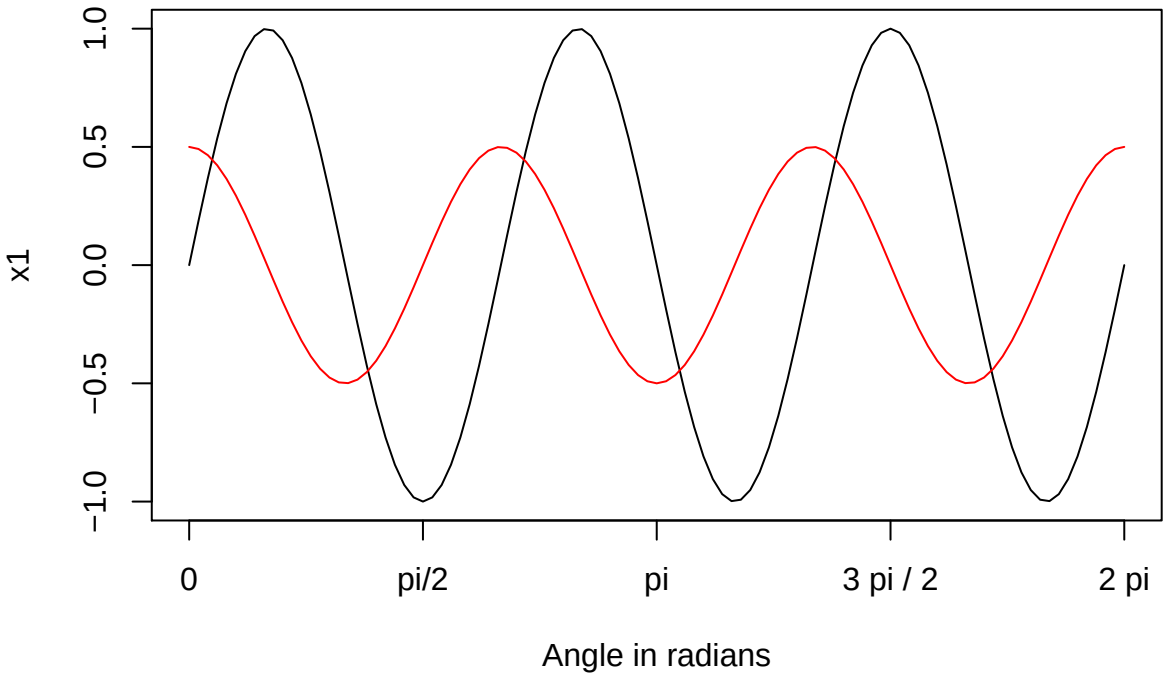
we can also say

$x(t) = A\sin(2\pi kt + \phi)$

where k is the harmonic number (number of waves per unit time)

```
#initial harmonic number
k = 3

#90 degree phase difference
pd = pi/2
A1 = 1
A2 = 0.5
w = 2*pi

x1 = A1*sin(k*w*t)
```

```
# sinusoid function with phase shift
x2 = A2*sin(k*w*t+pd)

labs = c(0,pi/2,pi,(3*pi)/2,2*pi)

#make a simple line plot and suppress x axis ticks (xaxt)
plot(t*2*pi, x1, 'l', xlab='Angle in radians', xaxt = 'n')
lines(t*2*pi, x2, col='red')
axis(1,at=labs,labels=c("0","pi/2","pi","3 pi / 2","2 pi"))
```



Harmonics and signals

Let's take the signal X and convert any point on X, which we'll call x, to radians

$\theta = \frac{2\pi x}{X}$

so X is distributed over 0 to 2π radians. The equation for this curve is

$X = A\cos(\theta)$

Now k is interpreted as a wavenumber (number of waves per unit phase), so

$X = \cos(k\theta)$

And phase shifts can be written as

$X = \cos(\theta - \phi)$

Combining, we get the equation for any component

$X_k = A_k \cos(k\theta - \phi_k)$

We can linearize using the identity

$\sin(a + b) = \sin(a)\cos(b) + \cos(a) + \sin(b)$

which gives us

$x_k = A_k \cos(\phi_k) \cos(k\theta) + A_k \sin(\phi_k) \sin(k\theta)$

Simplifying, and combining waves with different amplitudes, wavelengths and phase we can generalize the above to any signal as the sum of periodic components. we get the Fourier relationship

$X = \sum_{k=0}^{\infty} \alpha_k \cos(k\theta) + \beta_k \sin(k\theta)$

where the α and β terms combine amplitude and phase, i.e.

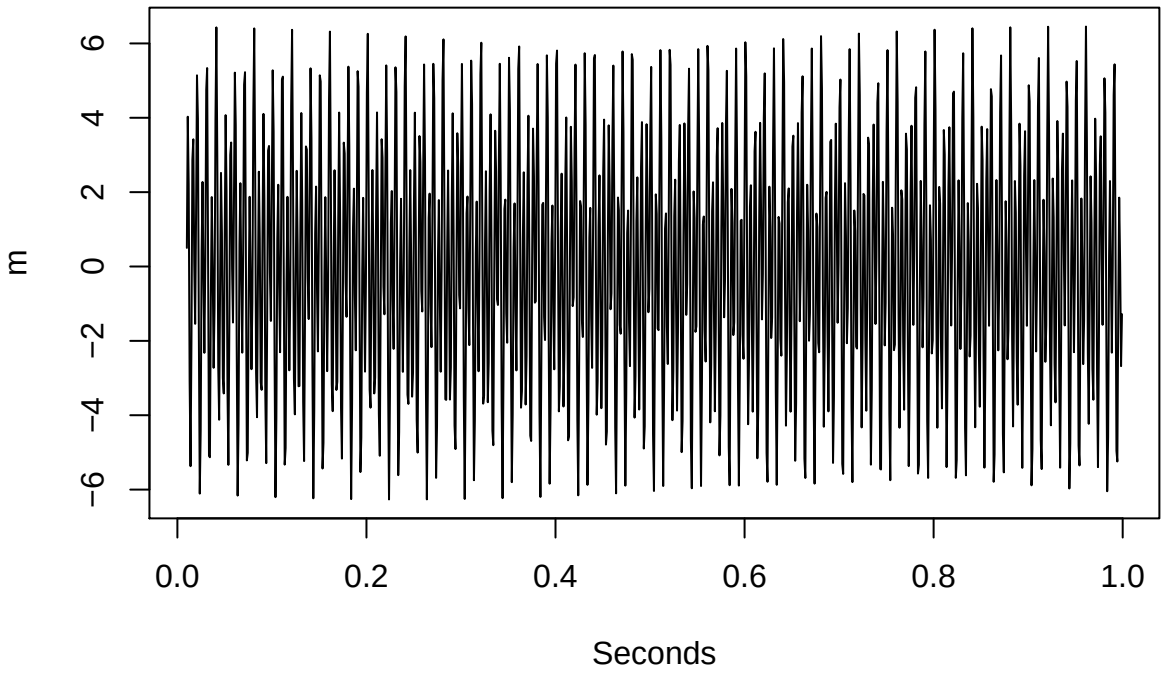
$\alpha_k = A_k \cos(\phi_k)$

and

$\beta_k = A_k \sin(\phi_k)$

Each wave component is called a harmonic. Notice that the above is a linear system so could be solved using matrix algebra.

```
# sample frequency
Fs = 1024

A=c(4,2,1,0.5)

k=c(200,100,75,50)

t = seq(from=0.01, to=1, by=1/Fs)

X = A[1]*sin(w*t*k[1]) + A[2]*cos(w*t*k[2]) + A[3]*sin(w*t*k[3]) + A[4]*cos(w*t*k[4])

plot(t,X, 'l', xlab='Seconds', ylab='m')
```



Fourier transform

The Fourier transform converts a wave from the time domain to the frequency domain. A plot of the frequency versus amplitude of sine wave components is a frequency spectrum

$Y_f = \sum_{n=0}^{N-1} x_n e^{i2\pi kn/N}$

where Y_f is the amount of frequency f in the signal. Each (kth) number is a complex number containing the amplitude and phase shift

N is the number of samples

n is the current sample

k is the current frequency, between 0 Hz and N-1 Hz

n/N is the percent of the time we've gone through

$2\pi k$ is the speed in radians/second

This is a Discrete Fourier Transform, which is appropriately for discretely sampled, finite time (or spatial) series.

We can gain geometric insight into the above by looking at it in complex polar notation

$Y_f = \sum_{n=0}^{N-1} x_n \left(\cos \frac{2\pi kn}{N} - i \sin \frac{2\pi kn}{N}\right)$

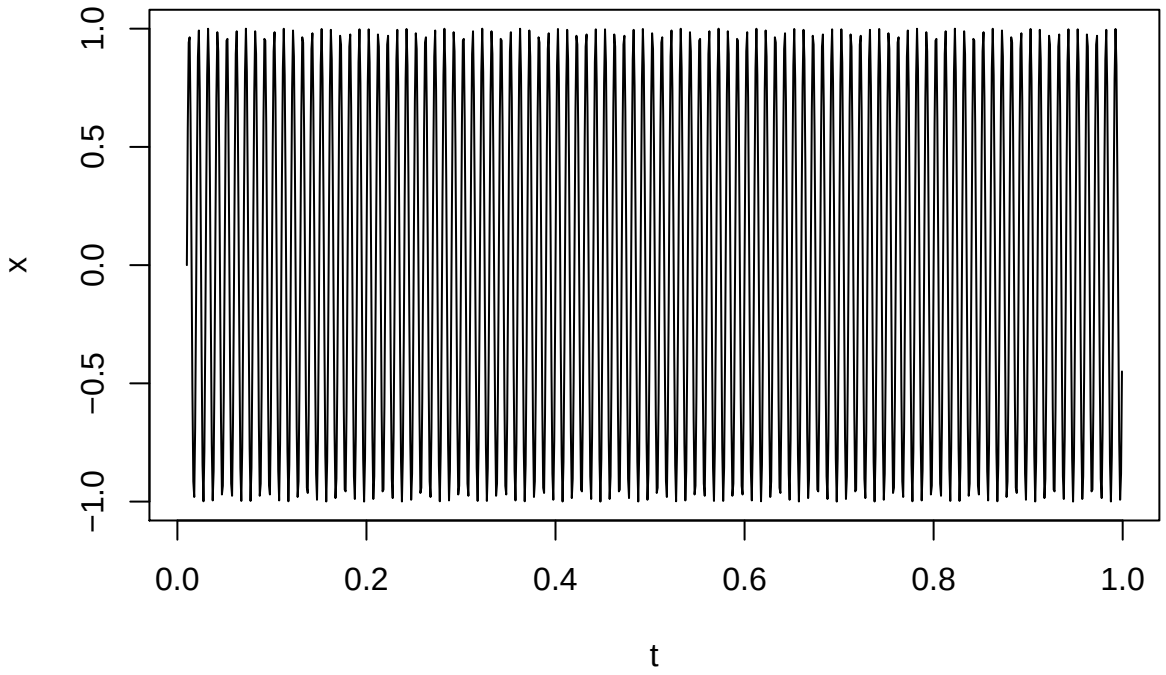
The discrete Fourier series is a complex series of the same length as the original series, and is composed of a real cosine part and an imaginary sine part. These parts are orthogonal.

The frequency resolution (binwidth) is f_s/N, where f_s is the sample frequency. What does that imply for sample design?

Spectral analysis of idealized signals

Let's create a sinusoid with a frequency of 100 Hz

```
# Sampling frequency
Fs = 1024

#wavenumber
k = 100

#amplitude
A = 1

# Time vector of 1 second
t = seq(from=0.01, to=1, by=1/Fs)

x = A*sin(w*t*k+0)

plot(t,x, 'l')
```

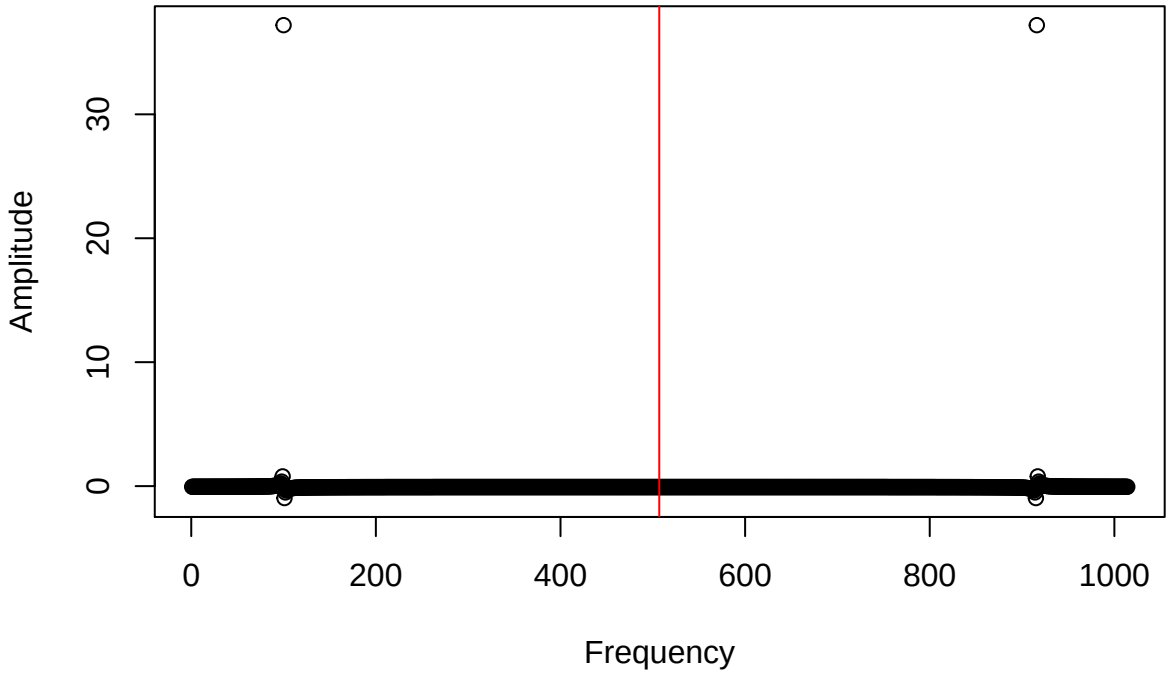


We can obtain the frequency spectrum of the signal using the fft function, that implements a Fast Fourier Transform algorithm.

```
n = length(x)
p = fft(x)
```

Let's plot the amplitude of the frequencies in the signal. What do you notice?

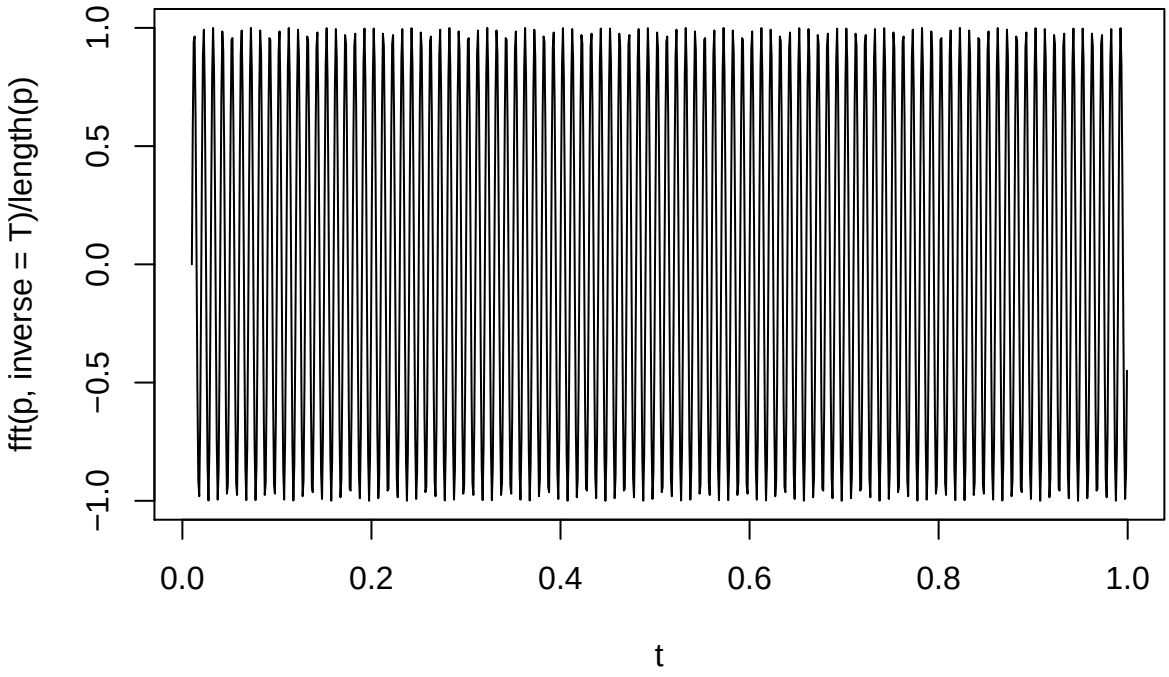
```
plot(seq(n), Re(p), xlab='Frequency', ylab='Amplitude')
abline(v=n/2, col="red")
```



We can go back and forward between the time and frequency components to recover the original signal:

```
plot(t,fft(p, inverse=T)/length(p),'l')
```

```
## Warning in xy.coords(x, y, xlabel, ylabel, log): imaginary parts discarded
## in coercion
```



Nyquist's and Parseval's theorems

The fundamental frequency, f_s/N, is the lowest frequency which can be determined from the time series. The highest frequency that can be resolved is the Nyquist frequency, $f_s/2$.

You should ensure that the sampling frequency is chosen such that oscillations at frequencies greater than half the sampling frequency are minimized.

Parseval's theorem says that the total area under the spectrum is equal to the total variance of the time series from which it was calculated

Computing the periodogram

The fourier transform returned by the fft function contains both magnitude and phase information and is given in a complex representation (i.e. returns complex numbers).

We select just the first half since the second half is a mirror image of the first. We don't use the Fourier coefficients beyond the Nyquist frequency

```
nUniquePts = ceiling((n+1)/2)
p = p[1:nUniquePts]
```

By taking the absolute value of the fourier transform we get the information about the magnitude of the frequency components.

```
p = abs(p)
```

Now we scale the magnitude by the number of points so that the magnitude does not depend on the length of the signal or on its sampling frequency.

```
p = p / n
```

Then we square it to get the power

```
p = p^2
```

To compensate for only using half the Fourier series, they are doubled (except for 0 and N/2)

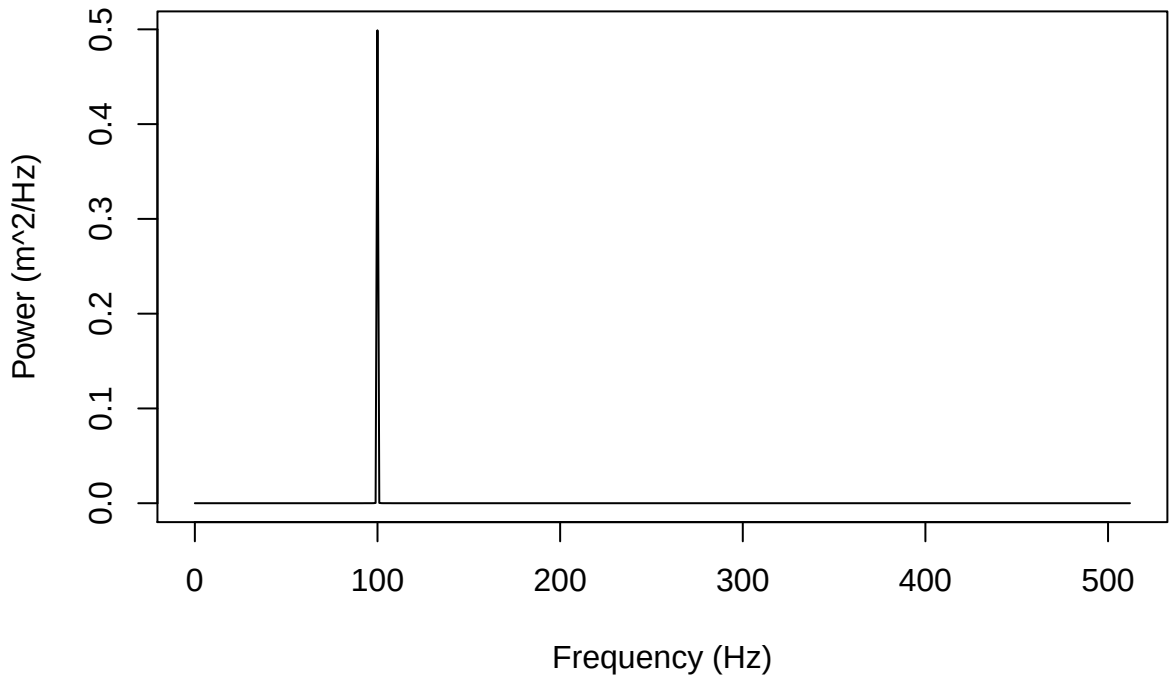
```
# multiply by two
# odd nfft excludes Nyquist point
# check if odd by checking the remainder after division by 2
if (n %% 2 > 0){
  p[2:length(p)] = p[2:length(p)]*2 # we've got odd number of points fft
} else {
  p[2: (length(p) -1)] = p[2: (length(p) -1)]*2 # we've got even number of points fft
}
```

Now we create the frequency array. Remember that is kf_s/N

```
freqArray = (0:(nUniquePts-1)) * ( Fs/n)
```

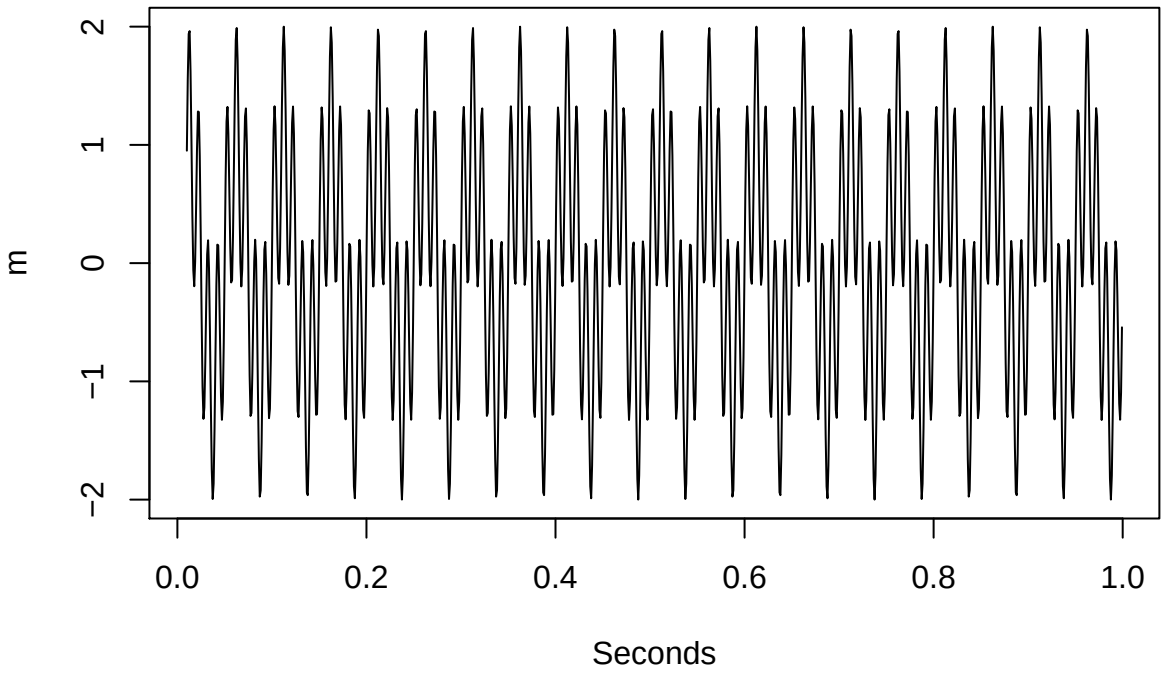
we plot the power by frequency in Hz

```
plot(freqArray, p, type='l', col='black', xlab='Frequency (Hz)', ylab='Power (m^2/Hz)')
```



Let's create a new time-series that is a combination of 2 sinusoids, 1 at 100 and one at 20 Hz

```
x =sin(w*t*100) + sin(w*t*20)

plot(t,x, 'l', xlab='Seconds', ylab='m')
```

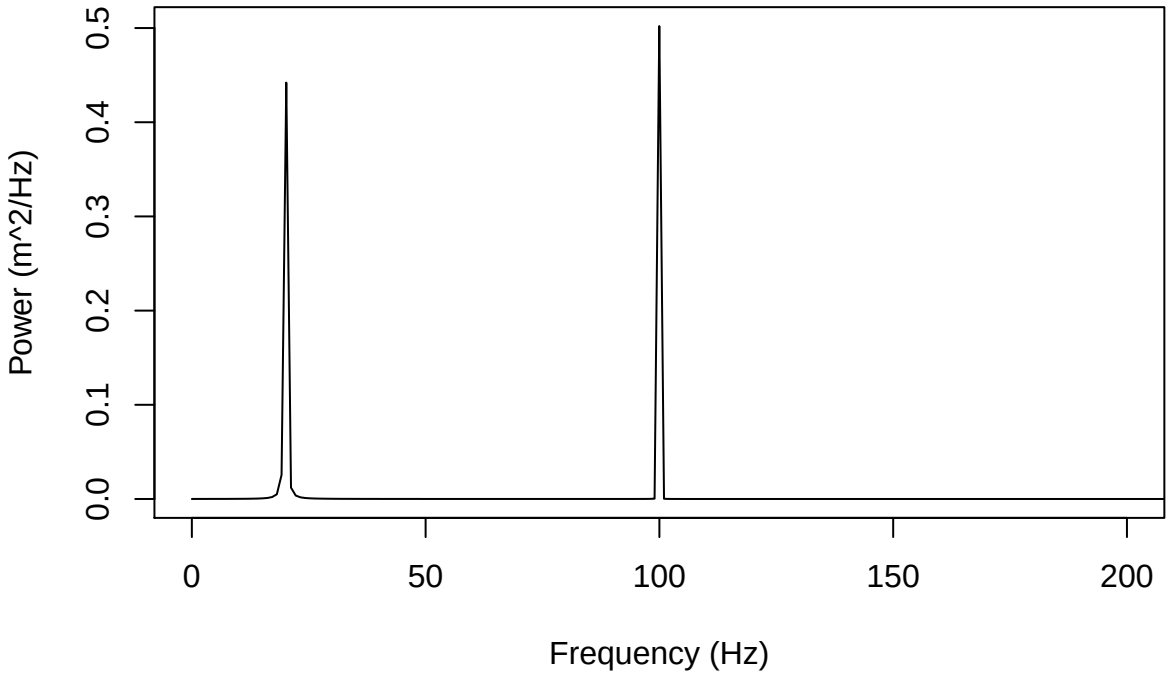


Let's create our own function that computes the power spectrum

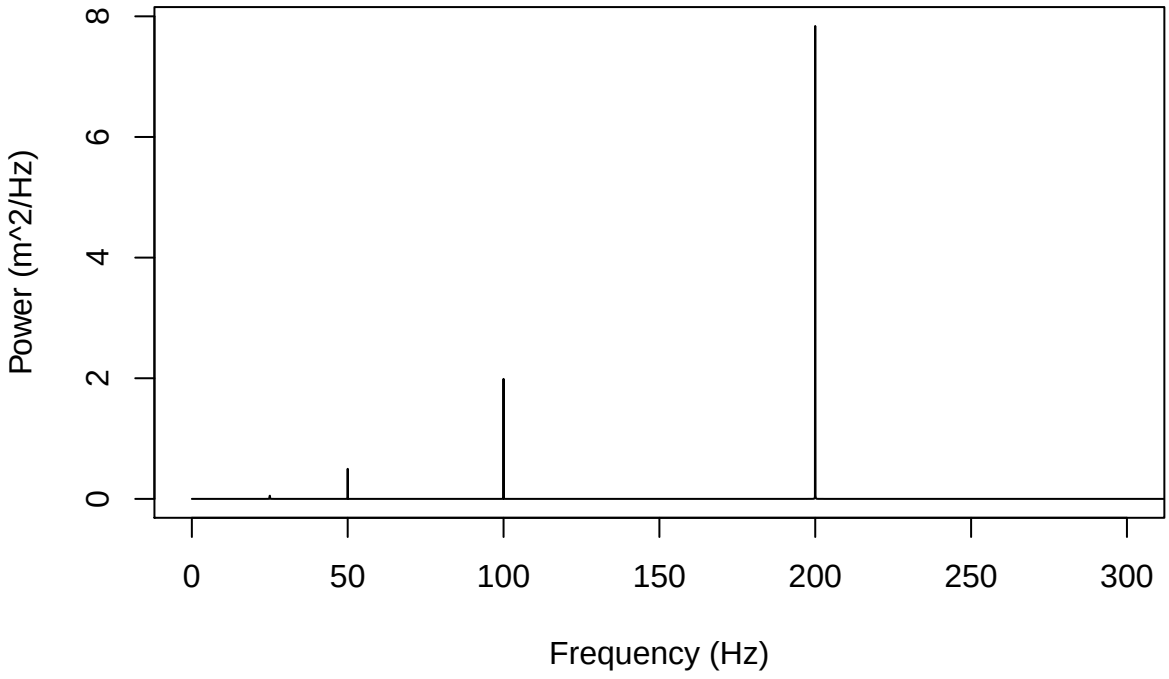
```
pgram = function(x, Fs) {

n = length(x)
p = fft(x)

nUniquePts = ceiling((n+1)/2)
p = p[1:nUniquePts] #select just the first half since the second half
                    # is a mirror image of the first
p = abs(p)  #take the absolute value, or the magnitude

p = p / n #scale by the number of points so that
          # the magnitude does not depend on the length
          # of the signal or on its sampling frequency
p = p^2  # square it to get the power

# multiply by two
# odd nfft excludes Nyquist point
if (n %% 2 > 0){
  p[2:length(p)] = p[2:length(p)]*2 # we've got odd number of points fft
} else {
  p[2: (length(p) -1)] = p[2: (length(p) -1)]*2 # we've got even number of points fft
}

freqArray = (0:(nUniquePts-1)) * ( Fs/n) #  create the frequency array

out = list("freqArray"=freqArray, "power" = p)
return(out)
}
```

```
ps = pgram(x,Fs)
```

```r
plot(ps$freqArray, ps$power, type='l', col='black',
     xlab='Frequency (Hz)', ylab='Power (m^2/Hz)', xlim=c(0,200))
```




```r
# Time vector of 10 second
t = seq(from=0.1, to=10, by=1/Fs)
x = 4*sin(w*t*200) + 2*cos(w*t*100) + sin(w*t*50) + 0.5*cos(w*t*25)

ps = pgram(x,Fs)
plot(ps$freqArray, ps$power, type='l', col='black',
     xlab='Frequency (Hz)', ylab='Power (m^2/Hz)', xlim=c(0,300))
```



Spectral analysis of ocean tides

```
#install.packages('rtide') # run this one time to install the package on your computer

library(rtide)
```

```
## rtide is not suitable for navigation
```

```
library(ggplot2)
```

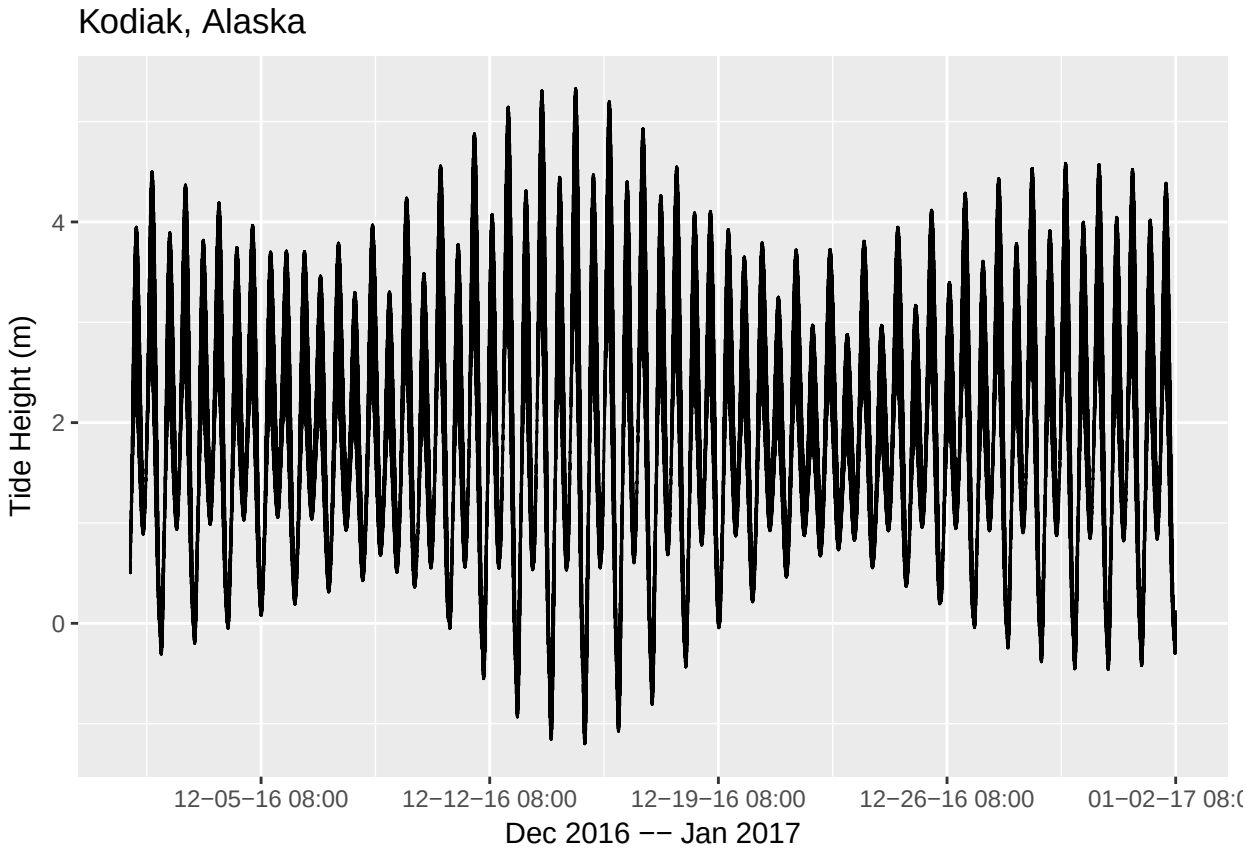
Let's get a tidal record, every 10 minutes, for the month of December 2016

```
data = rtide::tide_height(
  "Kodiak", from = as.Date("2016-12-01"), to = as.Date("2017-01-01"),
  minutes = 10L, tz = "PST8PDT")
```

Scales library allows for date formatting

```
library(scales)
#?scales
```

```
ggplot(data = data, aes(x = DateTime, y = TideHeight)) +
  geom_line() +
  scale_x_datetime(name = "Dec 2016 -- Jan 2017",
                   labels = date_format("%m-%d-%y %H:%M")) +
  scale_y_continuous(name = "Tide Height (m)") +
  ggtitle("Kodiak, Alaska")
```

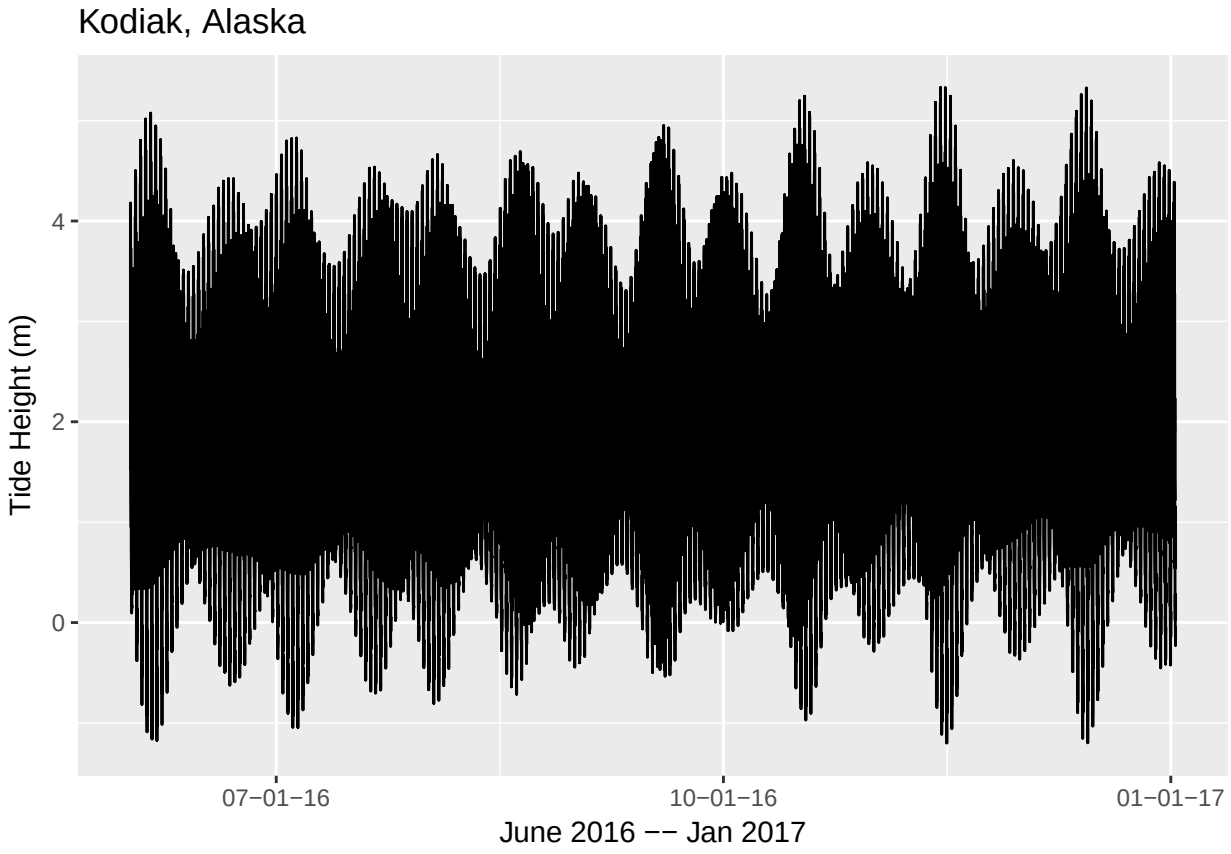


Let's get a longer time-series, from June 2016 to December 2017, this time every hour

```
data = rtide::tide_height(
  "Kodiak", from = as.Date("2016-06-01"), to = as.Date("2017-01-01"),
  minutes = 60L, tz = "PST8PDT")
```

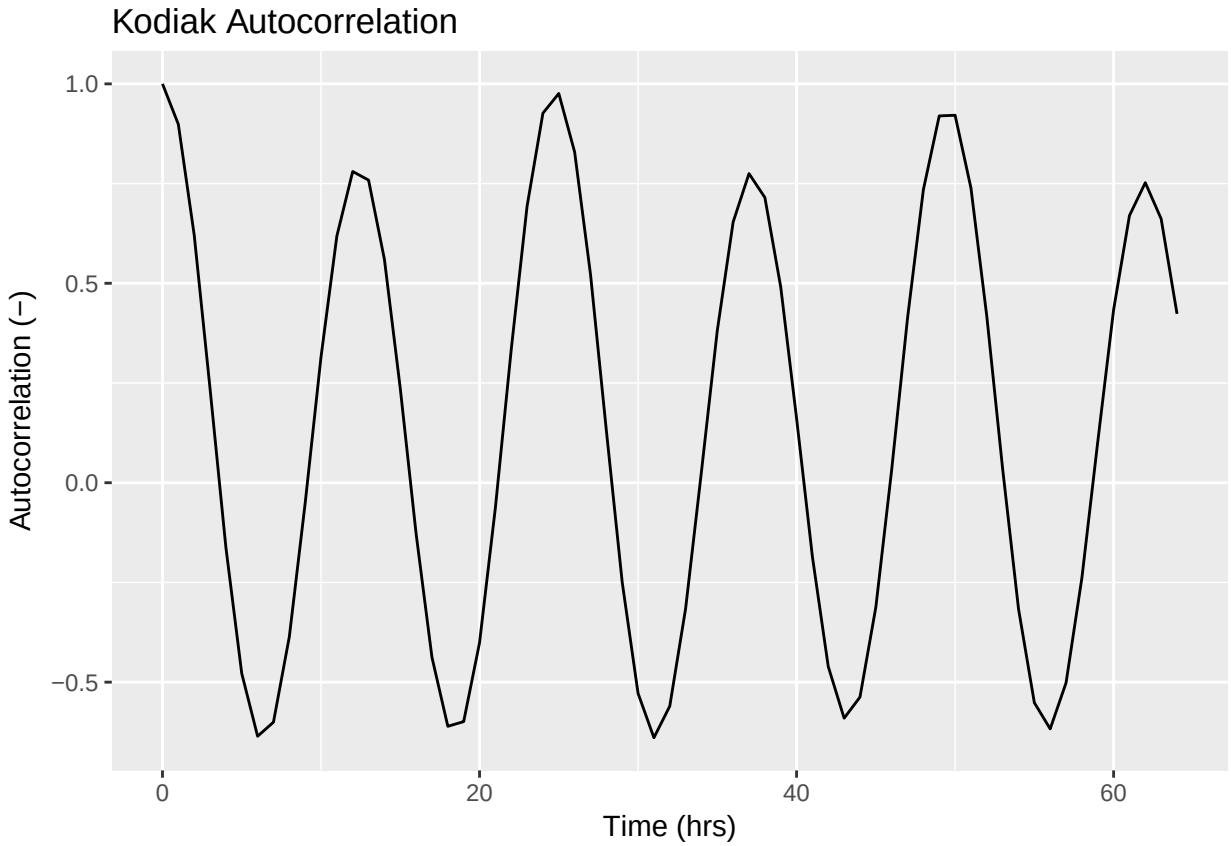
and make a new plot

```
ggplot(data = data, aes(x = DateTime, y = TideHeight)) +
  geom_line() +
  scale_x_datetime(name = "June 2016 -- Jan 2017",
                   labels = date_format("%m-%d-%y", tz="PST8PDT")) +
  scale_y_continuous(name = "Tide Height (m)") +
  ggtitle("Kodiak, Alaska")
```



Autocorrelation function

```
acf1 = acf(data$TideHeight, lag.max = 64, plot = FALSE)

ggplot() + geom_line(aes(x = acf1$lag, y = acf1$acf)) +
  scale_y_continuous(name = "Autocorrelation (-)") +
  scale_x_continuous(name = "Time (hrs)") +
  ggtitle("Kodiak Autocorrelation")
```



The second peak is bigger because the diurnal tide is more similar than the semi-diurnal tide, but the semi-diurnal peaks are more similar to each other

Does the semi-diurnal or the diurnal oscillation contain more variance? To answer this question, we'll need to use spectral analysis

The power spectrum the 'manual' way

Removed the mean from the signal

```
x = data$TideHeight - mean(data$TideHeight)
```

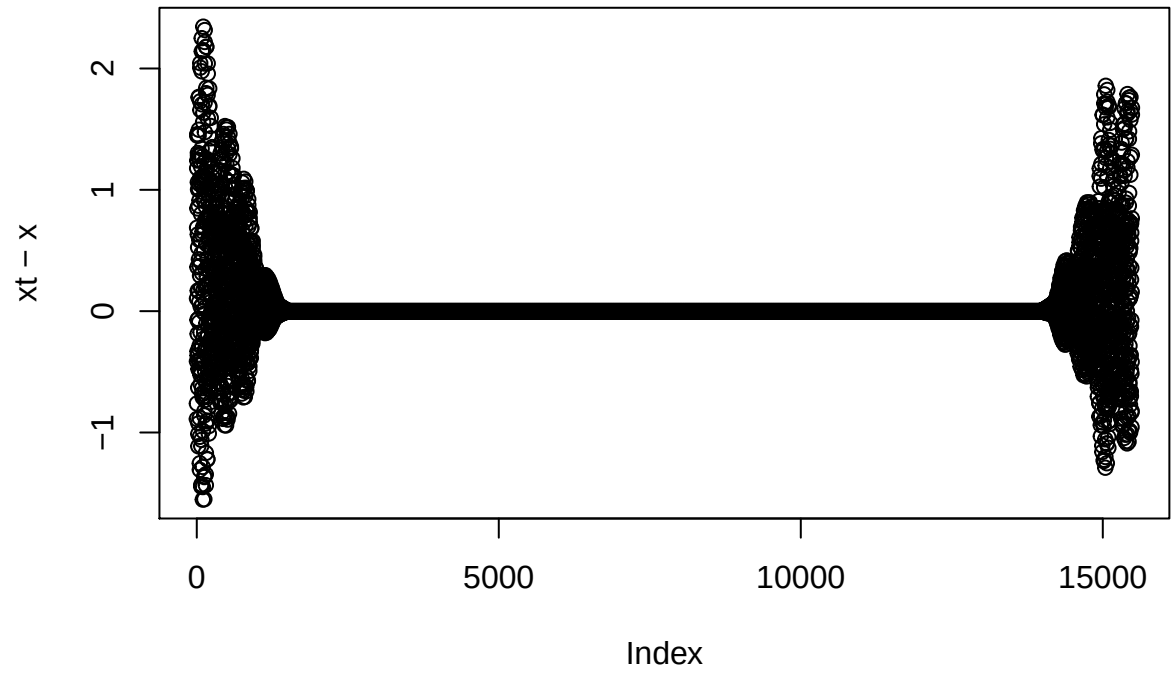
Apply a taper to prevent spectral leakage.

The series is finite length, so there are 'discontinuities' at either end. What this means is that energy 'leaks' into neighbouring frequency bins. This is called 'leakage' and is particularly problematic for narrow banded or short time series.

```
xt = spec.taper(x, p=0.1)
```

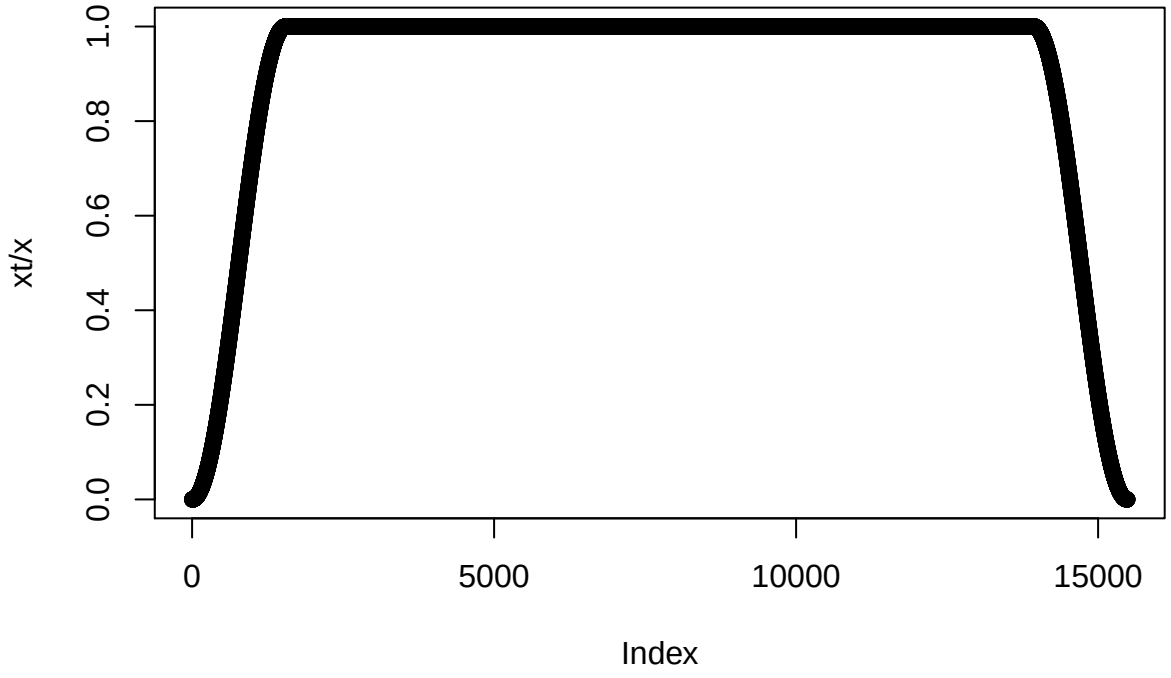
We can visualize what this did by plotting the difference

```
plot(xt - x)
```



or the taper itself

```
plot(xt/x)
```

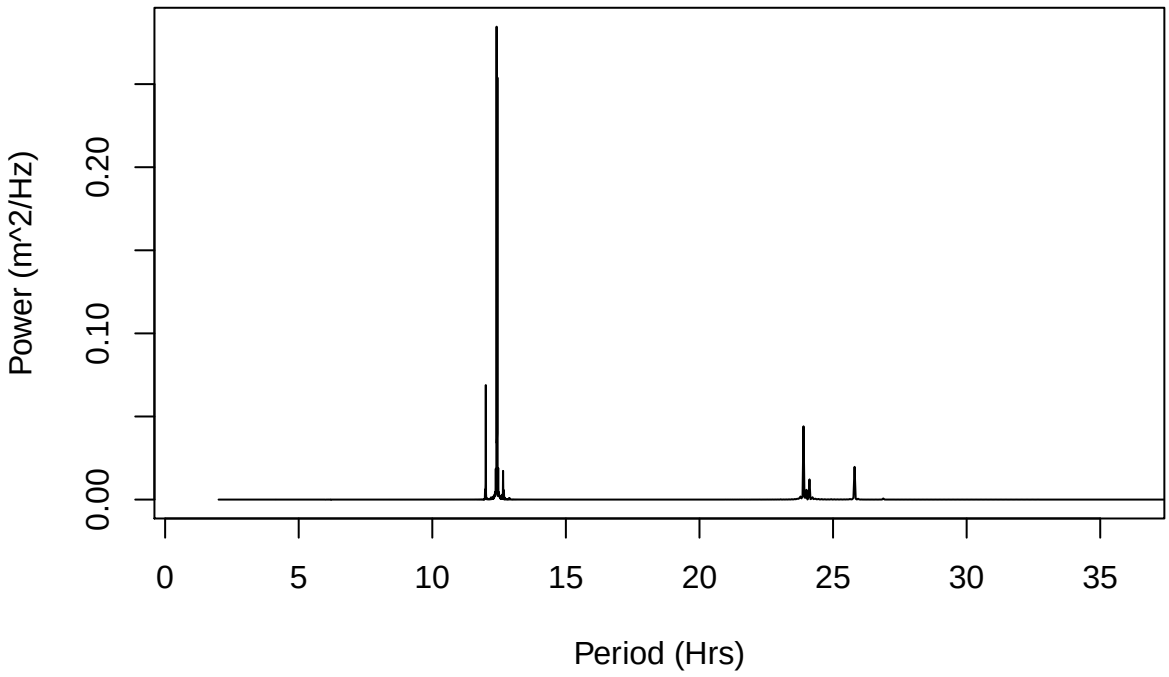


Compute the periodogram

```
ps = pgram(xt,1)
```

Let's plot the period instead of frequency

```
plot((1/ps$freqArray), ps$power, type='l', col='black',
     xlab='Period (Hrs)', ylab='Power (m^2/Hz)', xli=c(1,36))
```



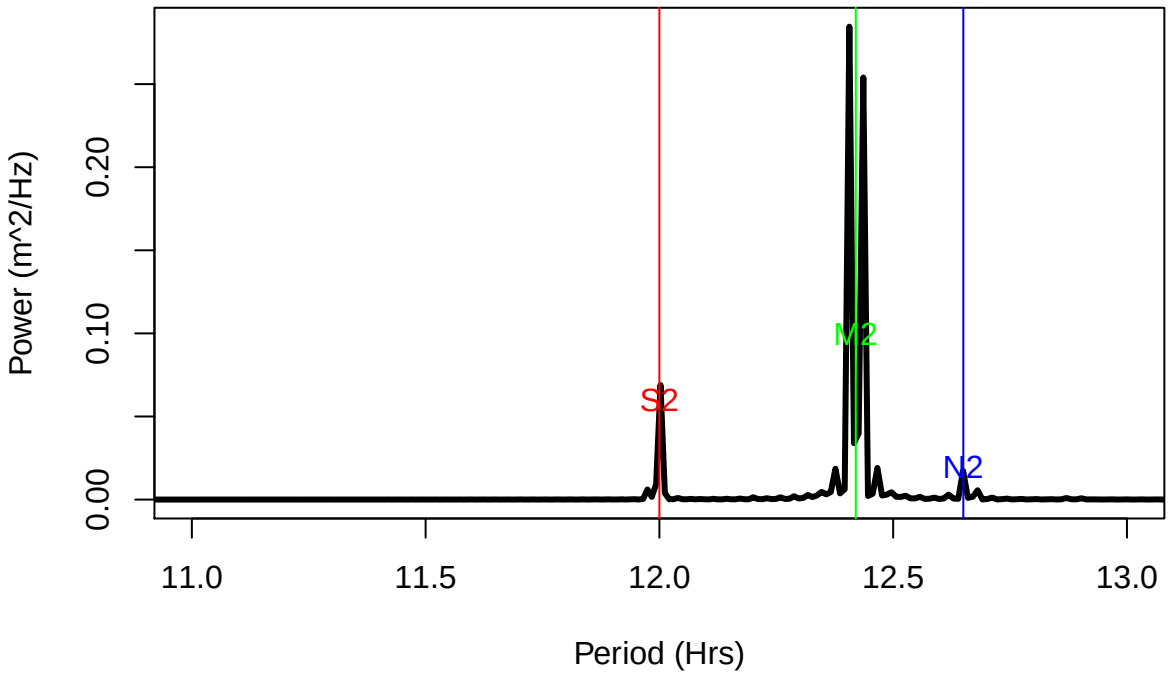
Semi-diurnal tidal harmonic constituents from NOAA are

M2 (principal lunar semidiurnal) = 12.42 hr @ 0.1 sq. m

S2 (principal solar) = 12 hr @ 0.06 sq. m

N2 (large lunar elliptic semidiurnal) = 12.65 hr @ 0.02 sq m

```
plot((1/ps$freqArray), ps$power, type='l', lwd=3, col='black',
     xlab='Period (Hrs)', ylab='Power (m^2/Hz)', xli=c(11,13))

abline(v=c(12, 12.42, 12.65), col=c("red","green","blue"))
text(12, 0.06, "S2", col="red")
text(12.42, 0.1, "M2", col="green")
text(12.65, 0.02, "N2", col="blue")
```



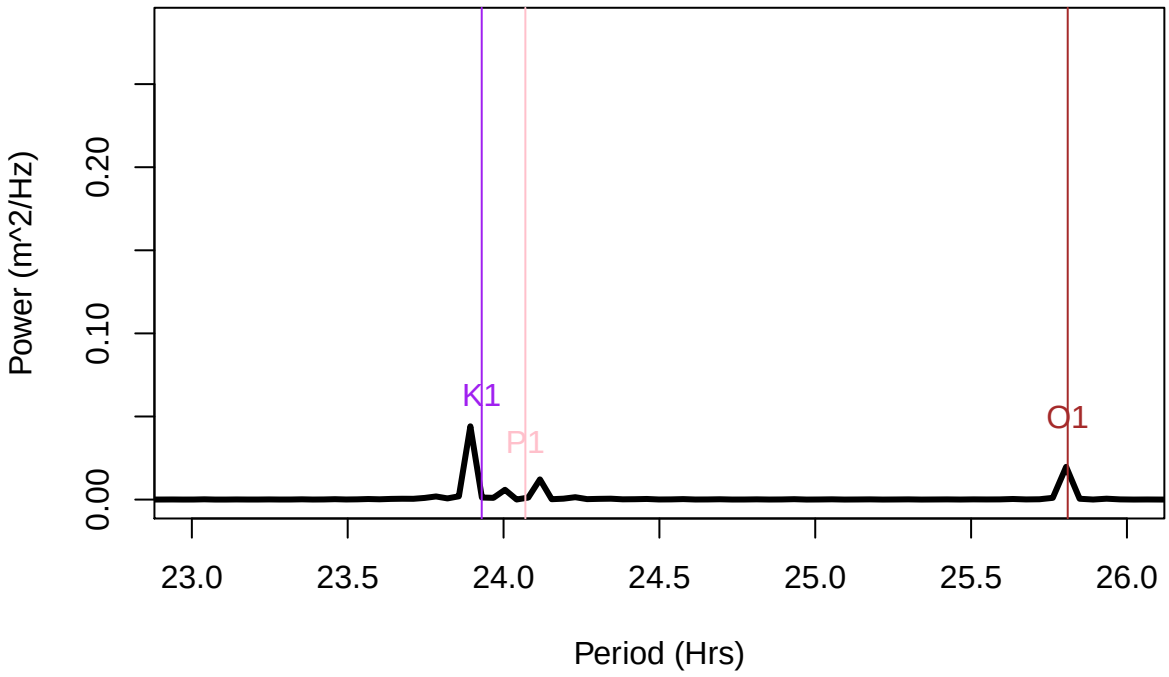
Diurnal tidal harmonic constituents from NOAA are

K1 (lunar) = 23.93 hr @ 0.063 sq. m

O1 (lunar) = 25.81 hr @ 0.05 sq. m

P1 (solar) = 24.07 hr @ 0.035 sq. m

```
plot((1/ps$freqArray), ps$power, type='l', lwd=3, col='black',
     xlab='Period (Hrs)', ylab='Power (m^2/Hz)', xli=c(23,26))

abline(v=c(23.93, 24.07, 25.81), col=c("purple","pink","brown"))
text(23.93, 0.063, "K1", col="purple")
text(25.81, 0.05, "O1", col="brown")
text(24.07, 0.035, "P1", col="pink")
```



To confirm that the value we have computed is indeed the power of the signal, we will also compute the root mean square (rms) of the signal.

Loosely speaking the rms can be seen as a measure of the amplitude of a waveform. If you just took the average amplitude of a sinusoidal signal oscillating around zero, it would be zero since the negative parts would cancel out the positive parts.

To get around this problem you can square the amplitude values before averaging, and then take the square root (notice that squaring also gives more weight to the extreme amplitude values)

```
rms_val = sqrt(mean(xt^2))
rms_val
```

```
## [1] 1.051283
```

This gives us the average tidal amplitude. The range is 2 * amplitude

since the rms is equal to the square root of the overall power of the signal, summing the power values calculated previously with the fft over all frequencies and taking the square root of this sum should give a very similar value

```
sqrt(sum(ps$power))
```

```
## [1] 1.051283
```

Quick demo of the Wiener-Khinchin Theorem

Which says that the power spectrum is the Fourier transform of the autocorrelation function.

First we compute the ACF, up to the lag which is 2 times the length of the periodogram (which was only computed up to N/2)

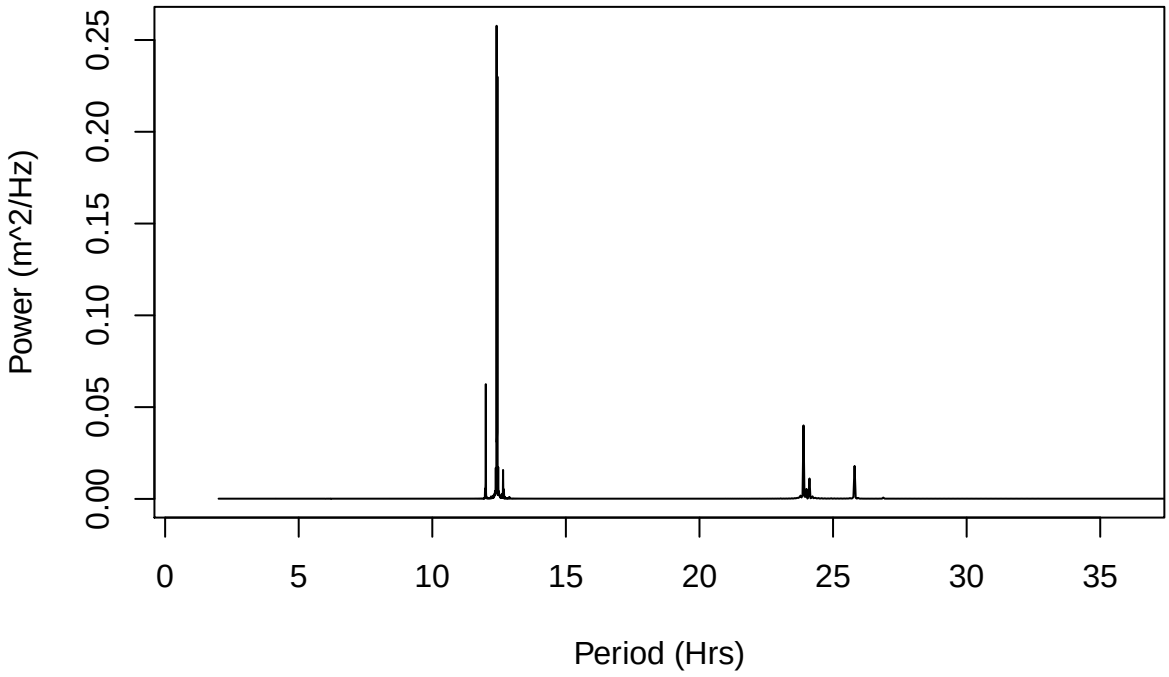
```
n = length(xt)

acf = acf(xt, lag.max = length(ps$freqArray)*2, plot = FALSE)
```

Then we take only the real (magnitude) part of the Fourier coefficients, up to N/2, to obtain spectral density estimates

```
acf2ps = Re(fft(acf$acf))[1:length(ps$freqArray)]
```

We make a plot of period against spectral density, doing the usual normalization based on length and one-sidedness

```
plot((1/ps$freqArray), 2*acf2ps/length(ps$freqArray), xli=c(1,36),'l',
     xlab="Period (Hrs)", ylab="Power (m^2/Hz)")
```



Power spectrum the 'R' way

Periodogram function in the 'spec' library

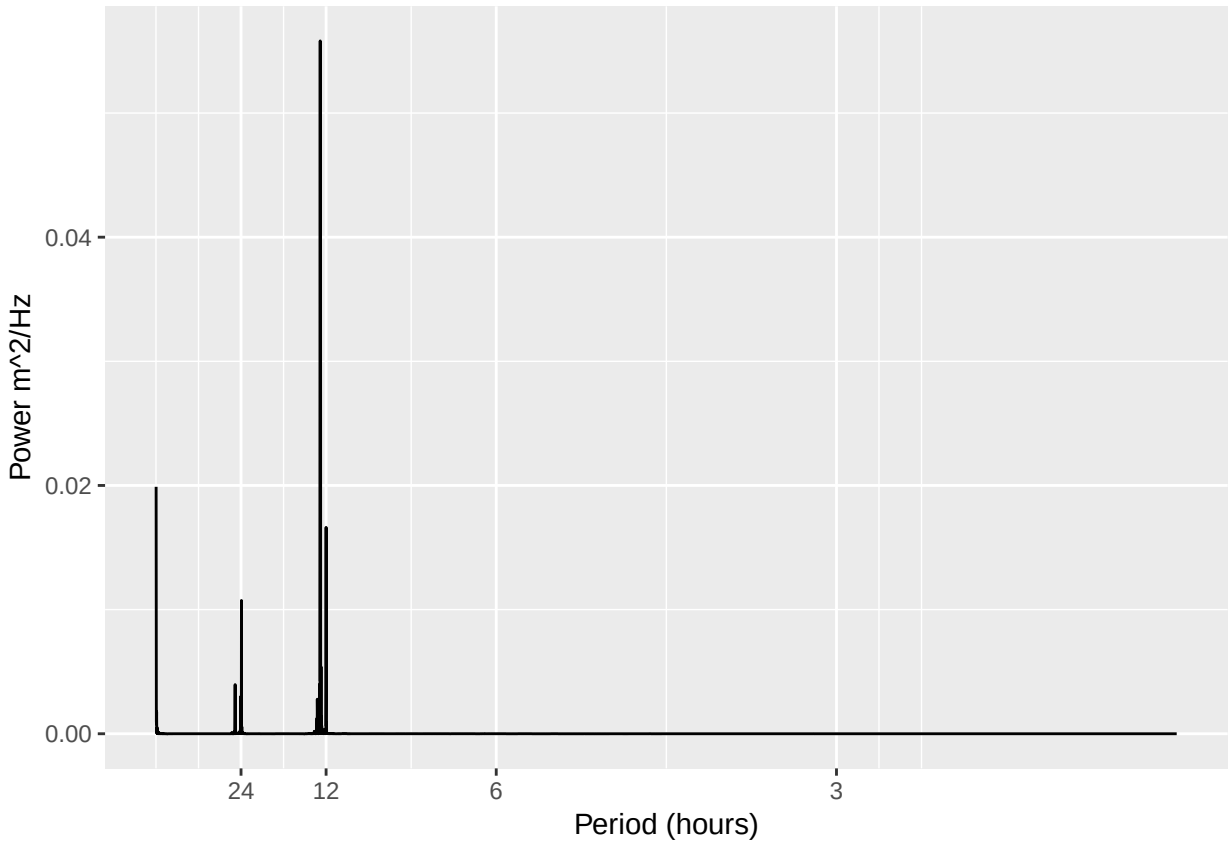
```
raw.spec = spec.pgram(xt, demean=F, plot=F, taper=0)
```

Create a dataframe and plot

Unlike the 'manual' way, the spec.pgram function doesn't normalize by (half) the length of the signal, therefore

```
spec.df = data.frame(freq = raw.spec$freq, spec = raw.spec$spec/n/2)
```

```
# Create a vector of periods to label on the graph, units are in hours
hrs.period = rev(c(0.5, 1, 3, 6, 12, 24))
hrs.labels = rev(c("1/2", "1", "3", "6", "12", "24"))
hrs.freqs = 1/hrs.period #Convert hourly period to hourly freq

spec.df$period = 1/spec.df$freq #freq to period

ggplot(data = subset(spec.df)) + geom_line(aes(x = freq, y = spec)) +
  scale_x_continuous("Period (hours)", breaks = hrs.freqs, labels = hrs.labels) +
  scale_y_continuous("Power m^2/Hz")
```



Let's check the the variance of the power spectrum the 'manual' way ...

```
sqrt(sum(ps$power))
```

```
## [1] 1.051283
```

... is equal to the variance of the power spectrum the 'R' way.

The 'R' way does not take into account that the raw periodogram is one sided, therefore we need to multiply by 2

```
sqrt(sum(spec.df$spec)) * 2
```

```
## [1] 1.053286
```

Confidence Intervals

Confidence limits aboue the spectral estimates are determined from the degrees of freedom, ν, not N

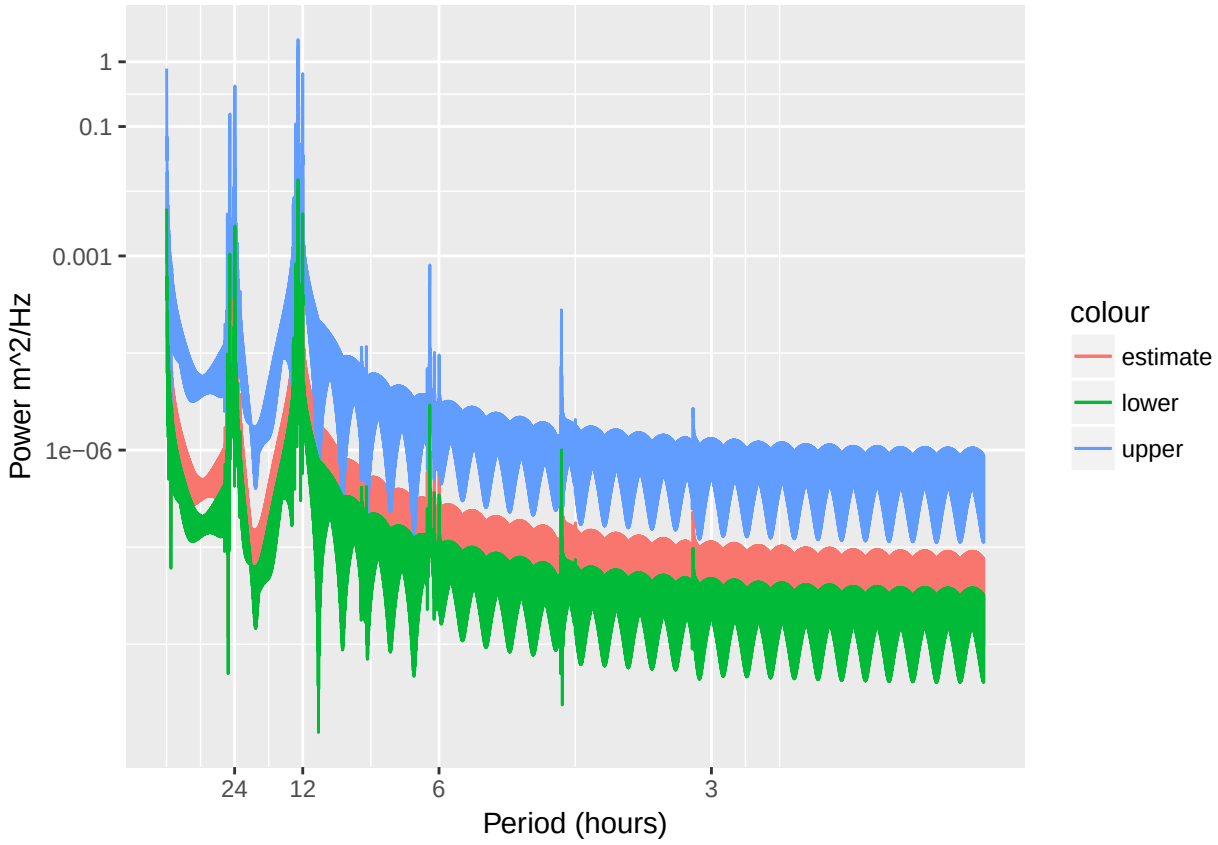
Chi-square with 2 degrees of freedom:

$2\frac{S_k}{\chi^2_{\nu,\alpha/2}} < S_k < 2\frac{S_k}{\chi^2_{\nu,1-\alpha/2}}$

```
alpha = 0.05
U = qchisq(alpha/2, 2)
L = qchisq(1-(alpha/2), 2)

spec.df$CIlower = 2*spec.df$spec/L
spec.df$CIupper = 2*spec.df$spec/U
```

Lets make a new plot and log scale so we can see the

```
ggplot(data = subset(spec.df)) +
  geom_line(aes(x = freq, y = spec, col='estimate')) +
  geom_line(aes(x = freq, y = CIupper, col='upper')) +
  geom_line(aes(x = freq, y = CIlower, col='lower')) +
  scale_x_continuous("Period (hours)", breaks = hrs.freqs, labels = hrs.labels) +
  scale_y_continuous("Power m^2/Hz", trans=log_trans(),
                     breaks=c(.000001, .001, .1, 1), labels=prettyNum)
```



Further reading

http://rstudio-pubs-static.s3.amazonaws.com/9428_1197bd003ebd43c49b429f22ea4f36e5.html