Autocorrelation and periodicity

Background

In this lesson, we will explore how to examine relationships among observations that are made in a sequence (time series). We will focus on characterizing additional patterns within a single time series.

Autocovariance and autocorrelation

Autocovariance for positive values of lag k is defined in a form similar to the cross-covariance:

$$c_{xx}(k) = \frac{1}{N}\sum_{t=1}^{N-k}(x_t - \overline{x})(x_{t+k} - \overline{x})$$

Similarly, autocorrelation is:

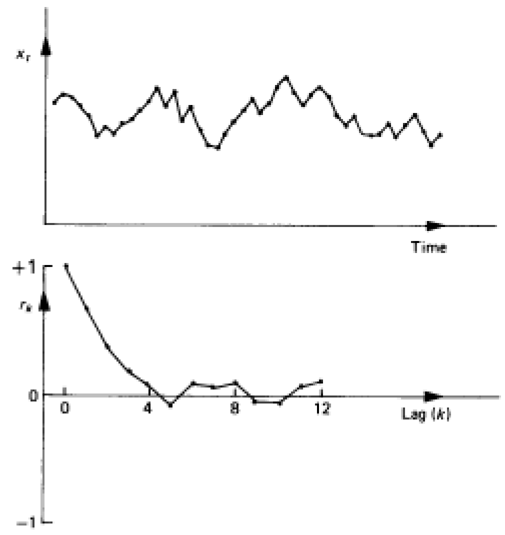
$$r_{xx}(k) = \frac{c_{xx}(k)}{c_{xx}(0)}$$

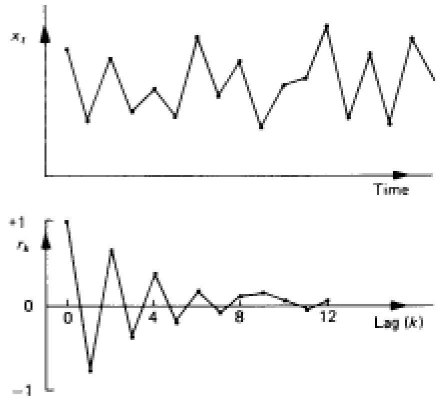
Autocorrelation coefficients and the correlogram (plot of $r_{xx}(k)$ as a function of lag k) can provide insight into the underlying processes (Chatfield 2003).

- *randomness*. low $r_{xx}(k)$ for all $k > 0$.
- *short-term correlations*. rapid drop-off following high $r_{xx}(1)$.
- *non-stationarity* (time series contains a trend). non-zero values of $r_{xx}(k)$ until high value of k.
- *periodic fluctuations* (for the duration corresponding to lag k). $r_{xx}(k)$ will oscillate with the frequency of the periodic fluctuation.

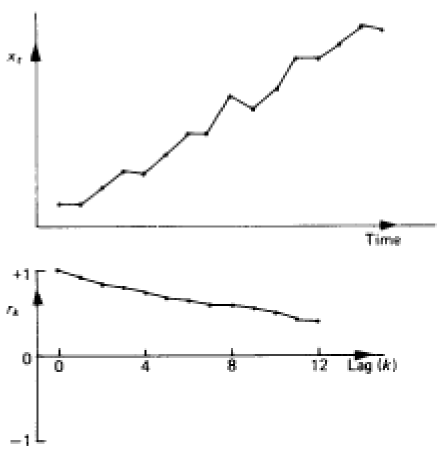
Example correlelograms (notated as "r_k") are shown below. Corresponding time series signals ("x_r") are shown directly above for the first three figures. In the last the figure (lower right quadrant), correlelograms are shown for monthly observations in air temperature (top), and short-term correlations remaining after removal of the seasonal contribution to the signal is removed (bottom).

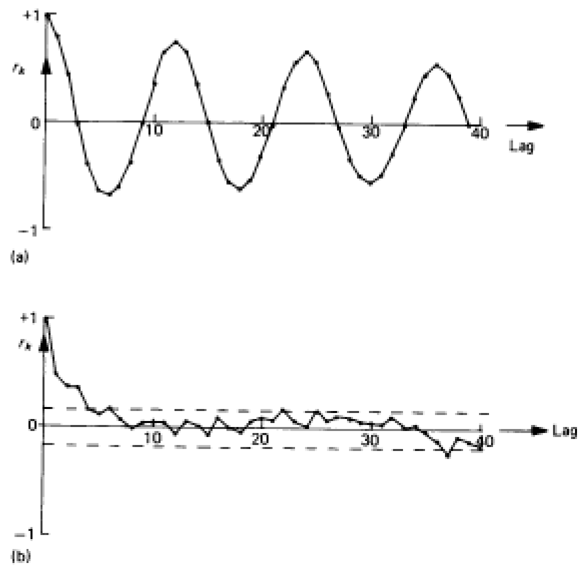
short-term corrlelation alternating time series





non-stationary time series short-term correlation after removal of seasonal variation





Figures 2.1–2.4 from Chatfield (2003).

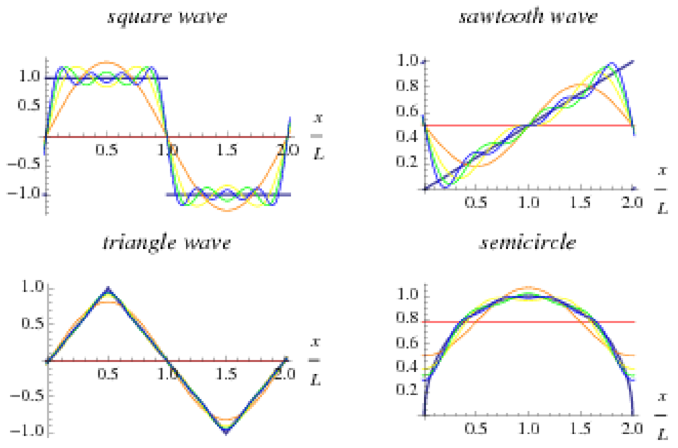
Such analyses can be used as a basis for constructing a purely statistical models of time series (*ARIMA* models) primarily used for forecasting, but we can also isolate and describe relationships among sequential observations using autocorrelations.

For instance, these representations can be included in discussions of:

- locally emitted pollutants, which might exhibit short-term autocorrelations, or
- pollutants produced by regional photochemistry and/or transported from non-local sources, which might lead to longer-term autocorrelations.

Power spectrum

We can project a times series (or arbitrary function) onto a basis set consisting of harmonic functions of various frequencies, and determine which coefficients contribute most to the reproduction of the original signal.



From Wolfram Mathworld (http://mathworld.wolfram.com).

Next, we will describe the Fourier series transformation of a sequential series of observations (time series) using notation adapted from Marr and Harley (2002).

The original time series x_t can be represented by a Fourier series:

$$x_t = \sum_{k=0}^{N-1} X(k) \exp(2\pi i \nu_k t)$$

where N is the number of observations, and $\nu_k = k/N$. The new series $X(k)$ defined by the discrete Fourier transform are given by:

$$X(k) = \frac{1}{\sqrt{N}} \sum_{t=0}^{N-1} x_t \exp(-2\pi i \nu_k t)$$

for $k = 0, 1, \ldots, N-1$. The periodogram or power spectrum is defined as:

$$P(\nu_k) = |X(k)|^2$$

and indicates the strength of frequencies in a time series.

The periodogram is a finite Fourier transform of the autocovariance $\{c_{xx}(k)\}$ (Chatfield 2003).

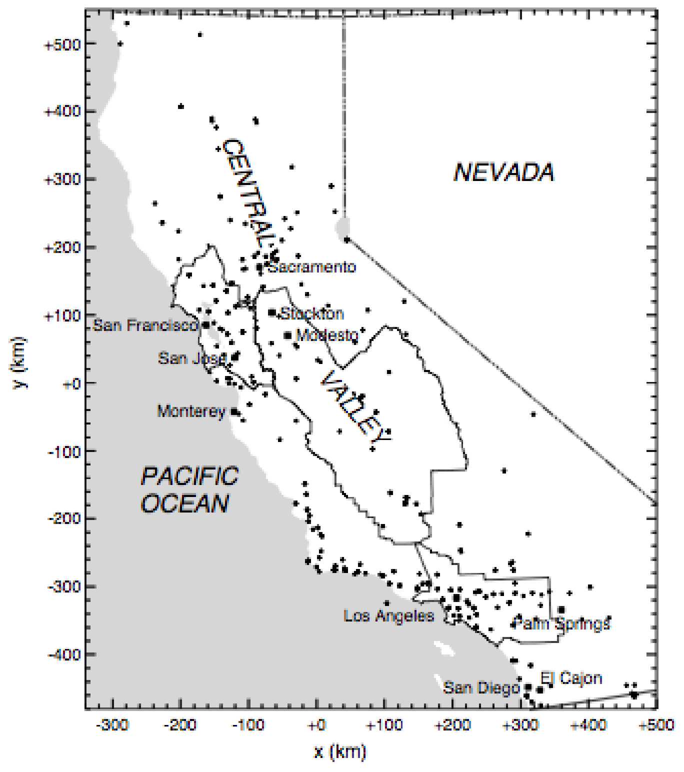


Fig. 2. Map of California showing locations of ambient air quality monitoring sites and outlines of the San Francisco Bay, San Joaquin Valley, and South Coast Air Basins.

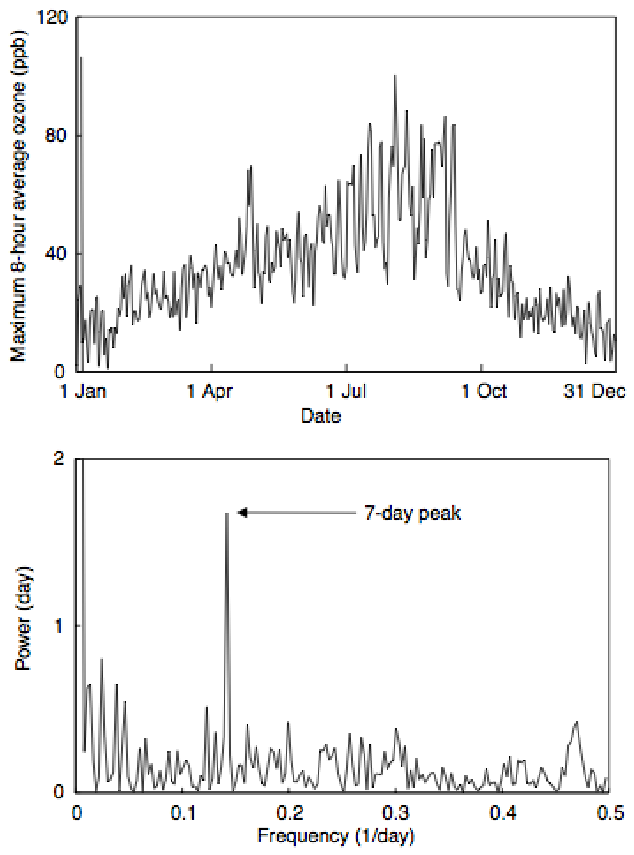


Fig. 3. (a) Daily ozone maxima at Stockton in 1998. (b) Periodogram of the mean-subtracted log ozone time series.

From Marr and Harley (2002).

R demonstration

```
library(dplyr)
library(reshape2)
library(chron)
library(ggplot2)
```

```
source("GRB001.R")
```

```
Sys.setlocale("LC_TIME","C")
options(stringsAsFactors=FALSE)
options(chron.year.abb=FALSE)
theme_set(theme_bw()) # just my preference for plots
```

```
df <- readRDS("data/2013/lau-zue.rds")
```

```
id.vars <- c("site", "datetime", "year", "month", "day", "hour", "season", "dayofwk", "daytype")

lf <- melt(filter(df, site=="ZUE"), id.vars=id.vars)
```

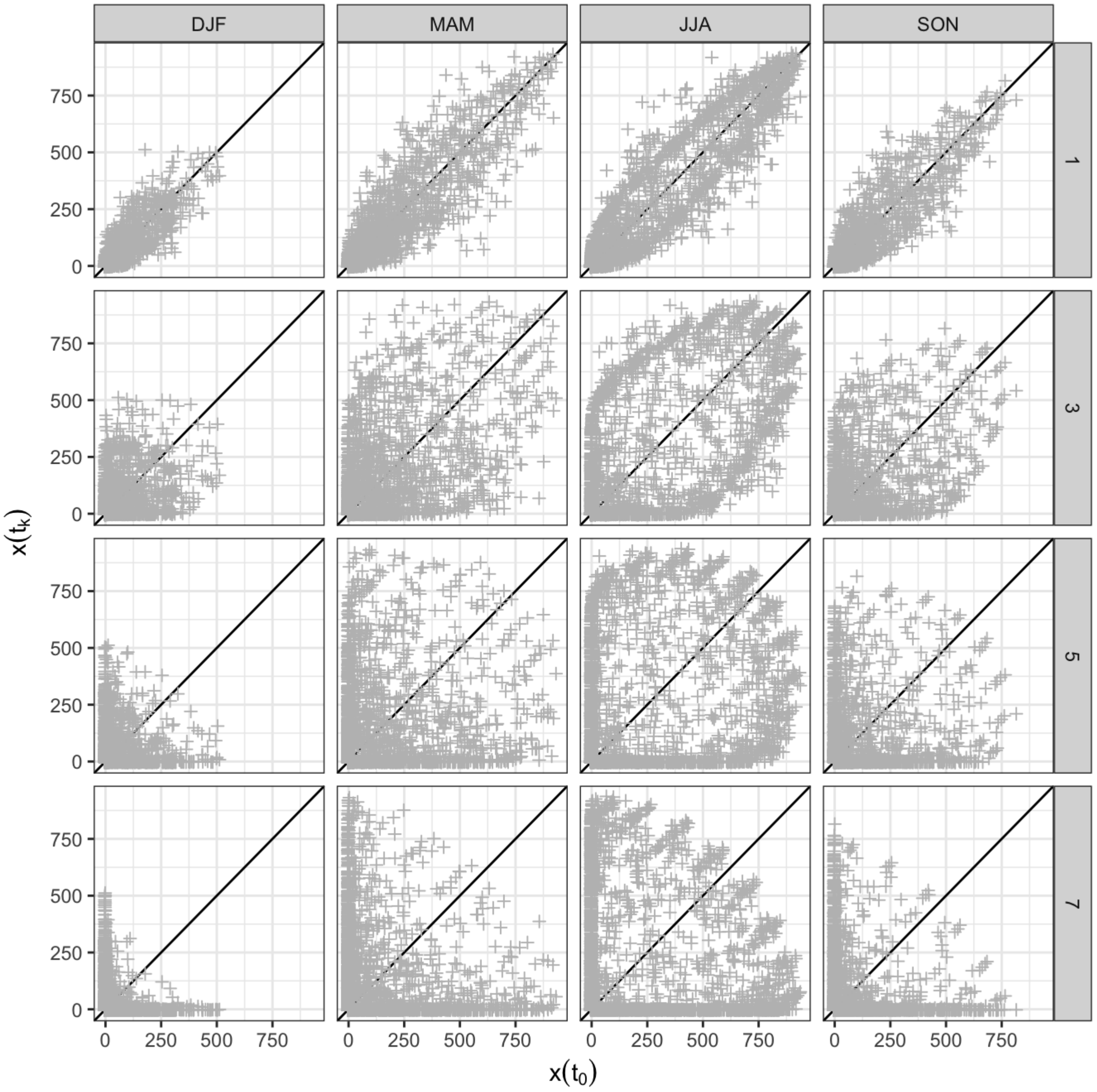
Autocorrelation

We will first visualize the effect of imposing a lag on a variable. We can apply lags to each variable (each time series is segmented by season) with the following code:

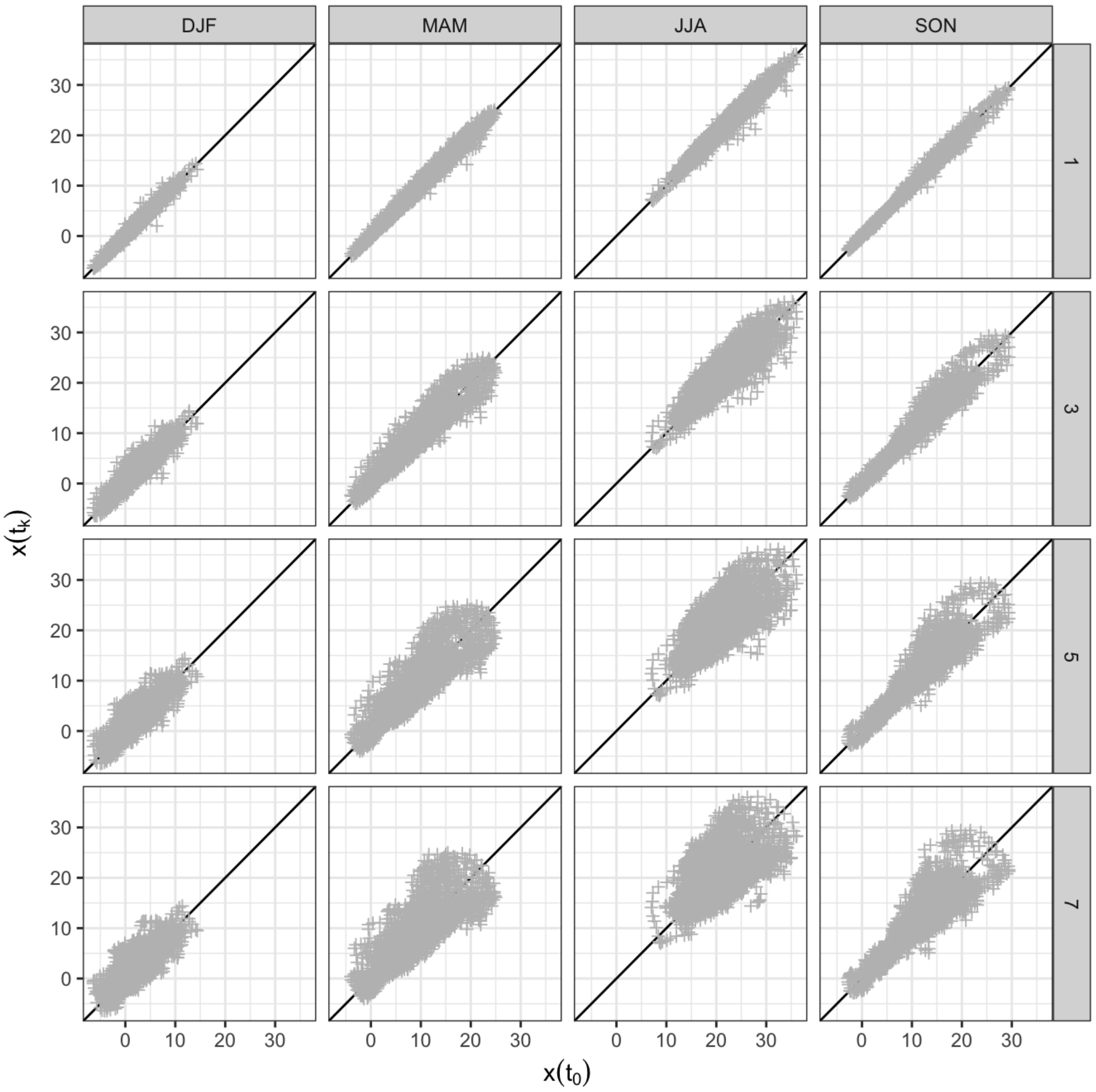
```r
SelfLag <- function(x, k) {
  data.frame(lag=k, x0=head(x,-k), xk=tail(x,-k))
}

lagged <- lf %>% group_by(site, season, variable) %>%
  do(rbind(SelfLag(.[["value"]], 1),
           SelfLag(.[["value"]], 3),
           SelfLag(.[["value"]], 5),
           SelfLag(.[["value"]], 7)))
```

We will plot the effect of the lag (shown across rows) on two meteorological variables: radiation and temperature.

```r
ggp <- ggplot(filter(lagged, variable=="RAD"))+
  geom_abline(intercept=0, slope=1)+
  geom_point(aes(x0, xk), shape=3, color="gray")+
  facet_grid(lag~season)+
  labs(x=expression(x(t[0])), y=expression(x(t[k])))
print(ggp)
```



```
ggp <- ggplot(filter(lagged, variable=="TEMP"))+
  geom_abline(intercept=0, slope=1)+
  geom_point(aes(x0, xk), shape=3, color="gray")+
  facet_grid(lag~season)+
  labs(x=expression(x(t[0])), y=expression(x(t[k])))
print(ggp)
```



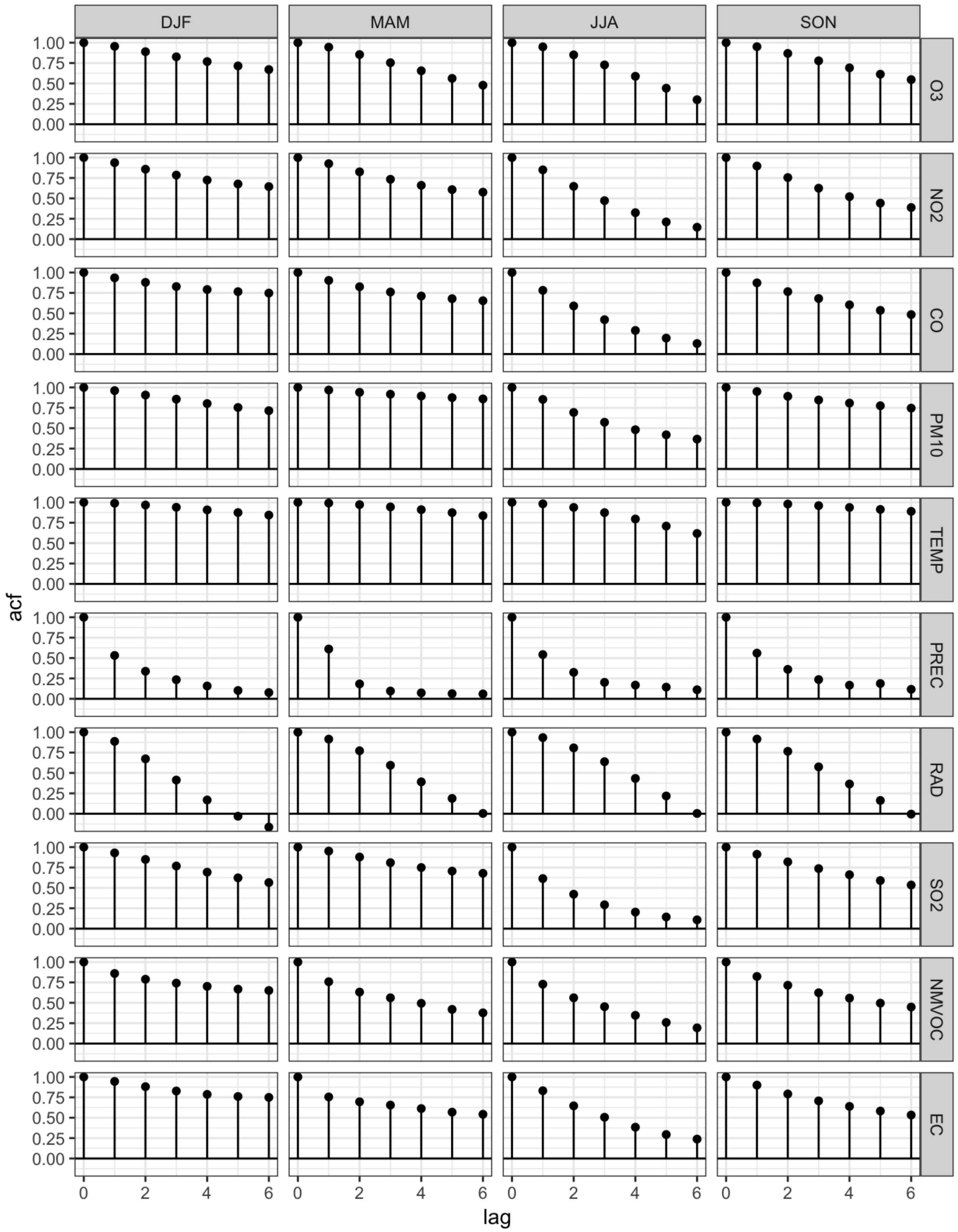
We can see that the autocorrelation in temperature persists for longer period of time (the correlation between lagged values for `TEMP` deteriorates much more slowly with increasing amount of lag than for `RAD`).

We will calculate the autocorrelation coefficient for up to a lag of 6 intervals (hours in our case).

```
Autocorrelation <- function(x, ...) {
  with(acf(x, ..., na.action=na.pass, plot=FALSE),
       data.frame(lag, acf))
}

autocorr <- lf %>% group_by(site, season, variable) %>%
  do(Autocorrelation(.[["value"]], lag.max=6))
```

We can view the correlelograms for all variables:

```
ggp <- ggplot(autocorr)+
  geom_hline(yintercept=0)+
  geom_point(aes(lag, acf))+
  geom_segment(aes(lag, 0, xend=lag, yend=acf))+
  facet_grid(variable~season)+
  labs(x="lag", y="acf")
print(ggp)
```



Power spectrum

Example with single variable

R has a built-in function, `spectrum`, to calculate the power spectrum. For illustration, we will apply it to hourly time series from July. For further details, see `?spectrum`.

```
ix <- df[["month"]]=="Jul"
spec <- spectrum(df[ix,"CO"])
```

```
## Error in na.fail.default(as.ts(x)): missing values in object
```

We see that there is an error on account of missing values. Oftentimes it is common to assess the extent of missing values by a quantity known as "percent (or fraction) data recovery." If the recovery is less than 75% (as a rule of thumb), you may want to flag any aggregated statistic calculated over this period (averages, periodicity, etc.) as unrepresentative/unreliable.

```
PercentRecovery <- function(x) {
  length(na.omit(x))/length(x)*1e2
}

PercentRecovery(df[ix,"CO"])
```

```
## [1] 99.79839
```

In this case, our data recovery is acceptable and we assume that interpolating over a few points will not affect the analysis too much. There are additional R packages (e.g., VIM) for visualizing patterns in missing data that may affect the analysis.

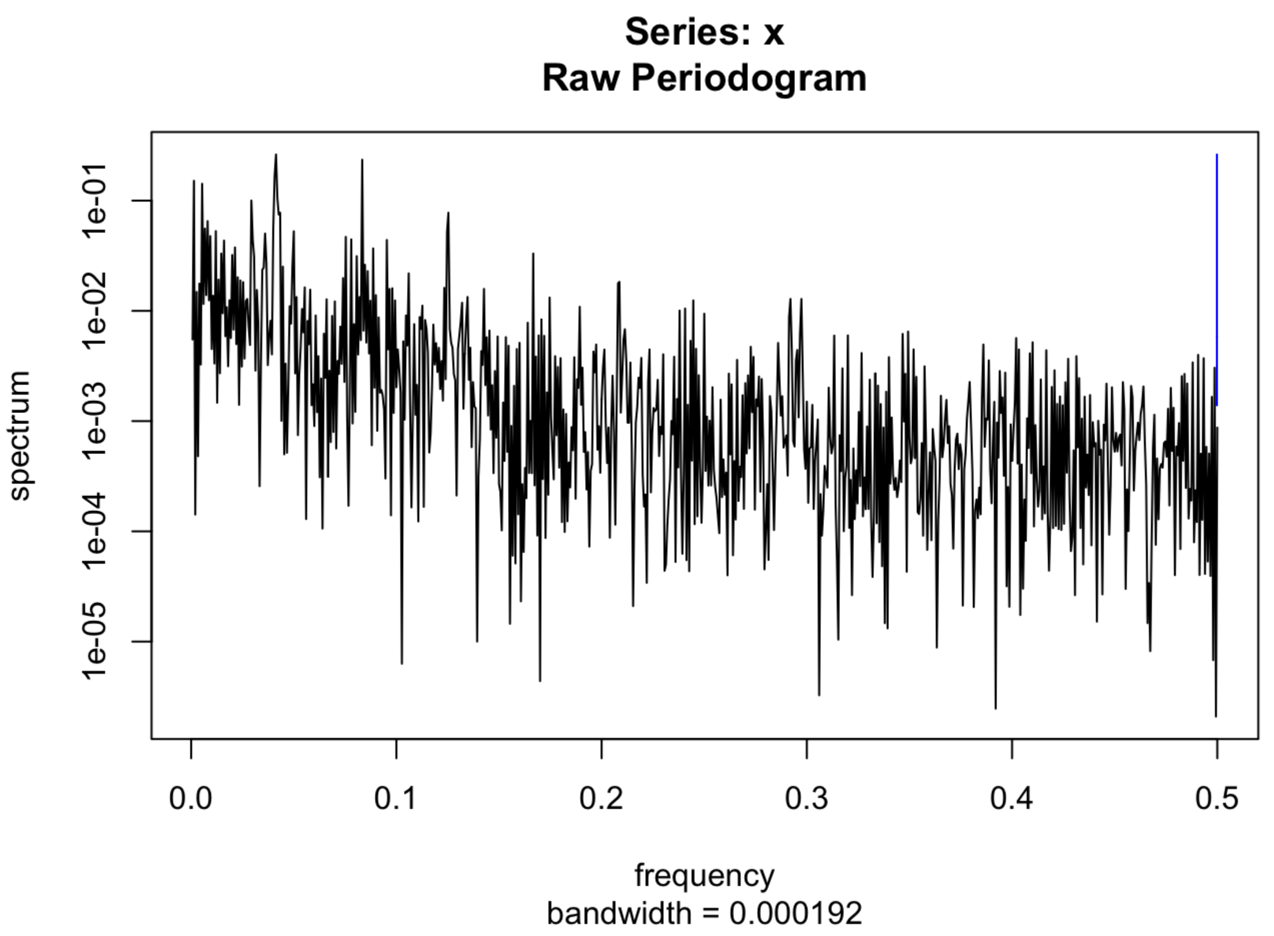
Furthermore, there is a class of statistical techniques called "imputation" for replacing missing values. For our lesson, we will simply use linear interpolation. This is accomplished using the `approx` function built into R. For further details, see `?approx`.

```
out <- approx(df[ix,"datetime"], df[ix,"CO"], df[ix,"datetime"])
str(out)
```

```
## List of 2
##  $ x: num [1:1488] 15887 15887 15887 15887 15887 ...
##  $ y: num [1:1488] 0.2 0.2 0.2 0.2 0.2 0.3 0.4 0.45 0.35 0.35 ...
```

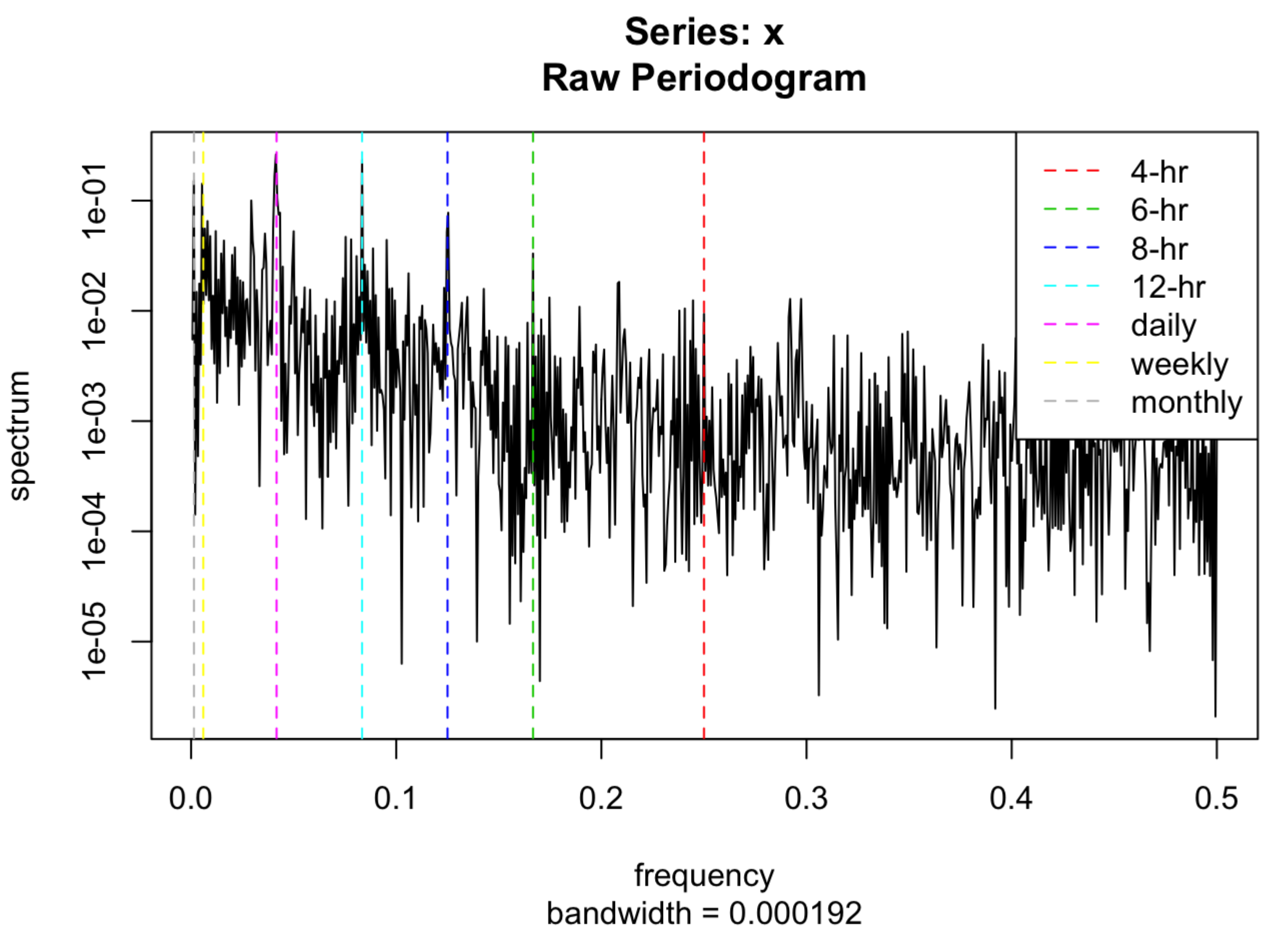
The list element `y` contains the interpolated values, and now we can calculate the spectrum, which also provides us with the plot.

```
spec <- spectrum(out[["y"]])
```



We can add frequencies which correspond to periods of 4, 6, 8, … hours. This is using the *base graphics* plotting functionality (in contrast to ggplot, which we have been using for most other applications) in R:

```
hrs <- c("4-hr"=4, "6-hr"=6, "8-hr"=8, "12-hr"=12, "daily"=24, "weekly"=24*7, "monthly"=24*30)
abline(v=1/hrs, col=seq(hrs)+1, lty=2)
legend("topright", names(hrs), col=seq(hrs)+1, lty=2, bg="white")
```



Application to multiple variables (hourly values)

Next, we want to generalize over all seasons and variables. First, let us examine our data recovery:

```
data.frame(lf %>% group_by(site, season, variable) %>%
  summarize(recovery=sprintf("%.0f%%",PercentRecovery(value))))
```

```
##    site season variable recovery
## 1   ZUE    DJF       O3     100%
## 2   ZUE    DJF      NO2     100%
## 3   ZUE    DJF       CO     100%
## 4   ZUE    DJF     PM10     100%
## 5   ZUE    DJF     TEMP     100%
## 6   ZUE    DJF     PREC      98%
## 7   ZUE    DJF      RAD     100%
## 8   ZUE    DJF      SO2     100%
## 9   ZUE    DJF    NMVOC     100%
## 10  ZUE    DJF       EC     100%
## 11  ZUE    MAM       O3     100%
## 12  ZUE    MAM      NO2     100%
## 13  ZUE    MAM       CO     100%
## 14  ZUE    MAM     PM10      99%
## 15  ZUE    MAM     TEMP     100%
## 16  ZUE    MAM     PREC     100%
## 17  ZUE    MAM      RAD     100%
## 18  ZUE    MAM      SO2     100%
## 19  ZUE    MAM    NMVOC     100%
## 20  ZUE    MAM       EC     100%
## 21  ZUE    JJA       O3     100%
## 22  ZUE    JJA      NO2     100%
## 23  ZUE    JJA       CO     100%
## 24  ZUE    JJA     PM10      95%
## 25  ZUE    JJA     TEMP     100%
## 26  ZUE    JJA     PREC     100%
## 27  ZUE    JJA      RAD     100%
## 28  ZUE    JJA      SO2     100%
## 29  ZUE    JJA    NMVOC     100%
## 30  ZUE    JJA       EC     100%
## 31  ZUE    SON       O3      99%
## 32  ZUE    SON      NO2     100%
## 33  ZUE    SON       CO     100%
## 34  ZUE    SON     PM10      99%
## 35  ZUE    SON     TEMP     100%
## 36  ZUE    SON     PREC     100%
## 37  ZUE    SON      RAD     100%
## 38  ZUE    SON      SO2     100%
## 39  ZUE    SON    NMVOC     100%
## 40  ZUE    SON       EC      95%
```

`"%.0f"` is a formatting specification for zero decimal places (`%.0`) applicable to floating point numbers (`f`), which are of mode/class `"numeric"` in R.

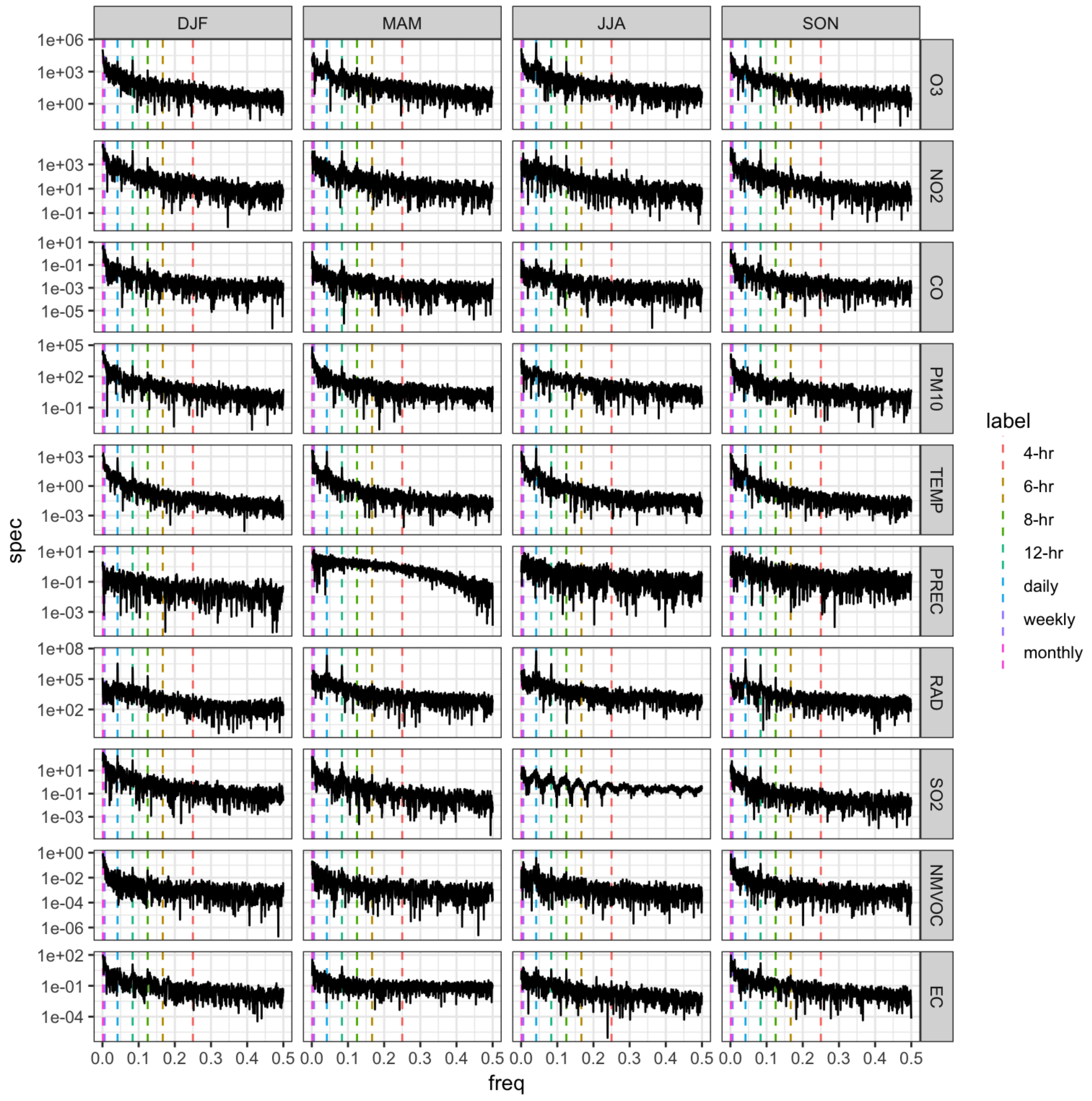
We find again that simple interpolation will likely not affect our results on account of the small number of missing values.

Let us apply the `spectrum` function more generally:

```
Spectr <- function(x, y) {
  out <- approx(x, y, xout=x)
  spec <- spectrum(out[["y"]], plot=FALSE)
  data.frame(spec[c("freq", "spec")])
}

spec <- lf %>% group_by(site, season, variable) %>%
  do(Spectr(.[["datetime"]],.[["value"]]))
```

We can similarly produce a power spectrum:

```
period <- data.frame(label=factor(names(hrs), names(hrs)), freq=1/hrs)

ggp <- ggplot(spec)+
  geom_vline(aes(xintercept=freq, color=label), data=period, linetype=2)+
  geom_line(aes(freq, spec))+
  facet_grid(variable~season, scale="free_y")+
  scale_y_log10()
print(ggp)
```



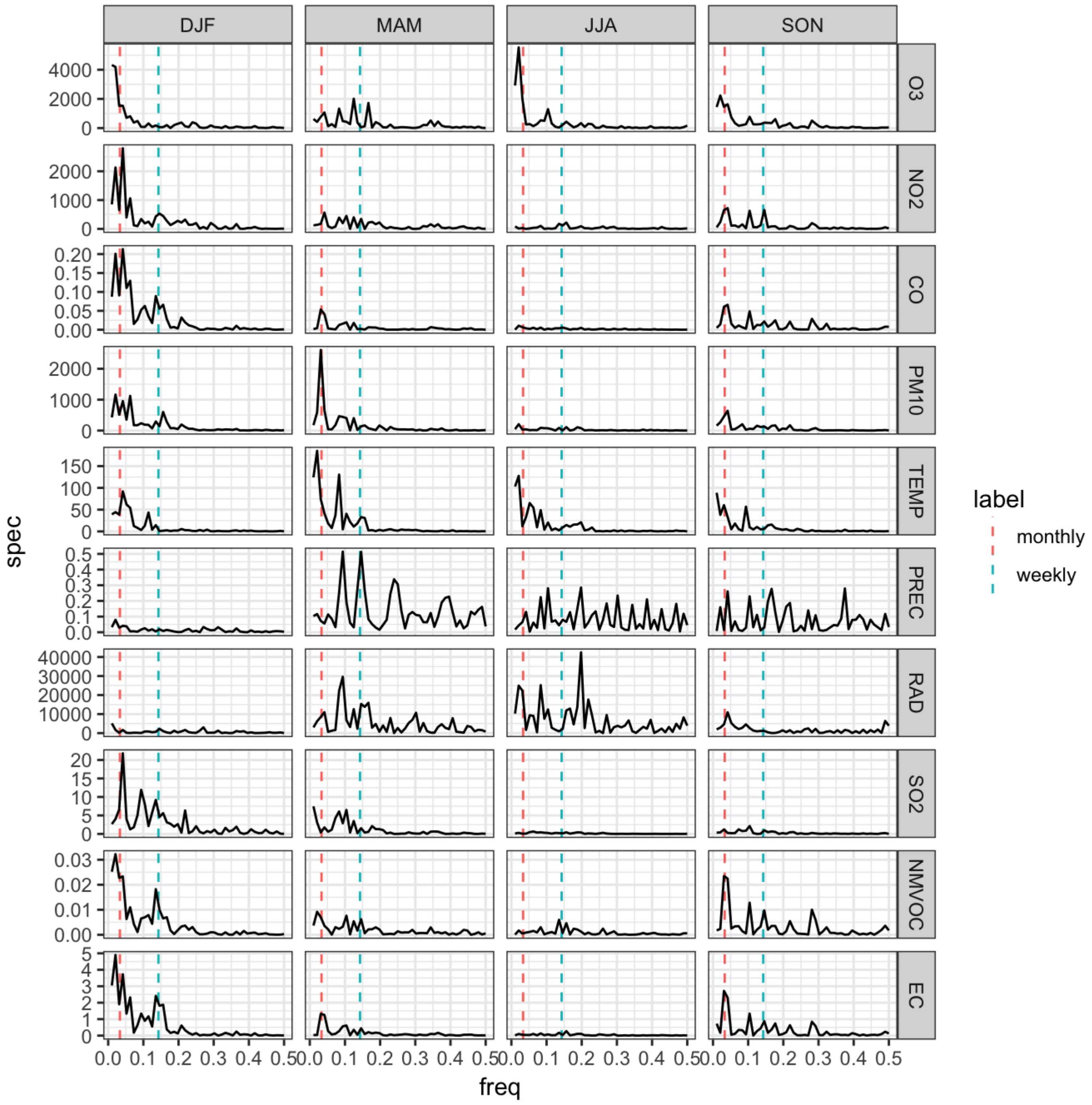
Application to multiple variables (daily values)

We can also calculate daily averages and apply the spectra decomposition:

```
daily <- lf %>% mutate(date=dates(datetime)) %>%
  group_by(site, variable, season, date) %>%
  summarize(value=mean(value, na.rm=TRUE))

spec.daily <- daily %>% group_by(site, season, variable) %>%
  do(Spectr(.[["date"]],.[["value"]]))

period.daily <- data.frame(label=c("weekly", "monthly"),
                           freq =1/c(7, 30))
```

Plot results for daily power spectra:

```
ggp <- ggplot(spec.daily)+
  geom_vline(aes(xintercept=freq, color=label), data=period.daily, linetype=2)+
  geom_line(aes(freq, spec))+
  facet_grid(variable~season, scale="free_y")
print(ggp)
```



References

Chatfield, C. 2003. *The Analysis of Time Series: An Introduction, Sixth Edition*. Chapman & Hall/CRC Texts in Statistical Science. CRC Press.

Marr, Linsey C., and Robert A. Harley. 2002. “Spectral Analysis of Weekday–weekend Differences in Ambient Ozone, Nitrogen Oxide, and Non-Methane Hydrocarbon Time Series in California.” *Atmospheric Environment* 36 (14): 2327–35. doi:10.1016/S1352-2310(02)00188-7 (https://doi.org/10.1016/S1352-2310(02)00188-7).