`wq`: Exploring water quality monitoring data (version 0.3-1)

Alan D. Jassby and James E. Cloern

January 29, 2011

Contents

1 Introduction

This package contains functions to assist in the processing and exploration of data from monitoring programs for aquatic ecosystems. The name *wq* stands for *w*ater *q*uality and reflects a focus on time series data for physical and chemical properties of water, as well as the plankton. The package is intended for programs that sample approximately monthly at discrete stations, a feature of many legacy data sets. Although our emphasis is mainly estuarine and nearshore coastal ecosystems, most functions should be applicable for a wide range of systems, from freshwater to open ocean.

The approach used here involves transformation of external data files into a standard format that existing functions can then handle easily. A conceptualization of this sequence is illustrated in Figure 1. Water quality monitoring programs maintain their *data* in a wide variety of formats, and the first step is to *read* data from an external file and store it in a data frame. Often, the external data are stored or at least transmitted in a comma- or tab-delimited format and can be easily handled with `read.table` or one of its variants. Some *clean*ing or manipulation of the data set may take place during the import process, but more substantive ones are often undertaken immediately after. Typical modifications include renaming variables, dropping unnecessary variables and observations, and coercing variables to different classes. These modifications are chosen with regard to ease of use and the intended analysis, but also in order to facilitate construction of an object with a standardized format. Before constructing this object, though, we may want to *derive* new variables from the original ones (e.g., salinity from conductivity). Next, we *generate* the standardized "wq data" object, which is a member of the `WqData` class defined in this package. We can then *reshape* into various forms—matrix, list, time series vector, data frame, etc.—depending on the analysis. At this point, the data are finally in a form that we can *analyze and visualize*. Some functions may be able to explore a `WqData` object directly without any additional reshaping.

This package is intended to facilitate all of these activities. We will illustrate some of the steps in Figure 1 using the accompanying data set `sfbay`. The exercise should demonstrate most of the current capability of the package and make its use more clear.

```
> library(wq)
```

2 Preparing data from an external file

Our starting point is a comma-delimited file downloaded on 2009-11-17 from the U.S. Geological Survey's water quality data set for San Francisco Bay (`http://sfbay.wr.usgs.gov/access/wqdata`). The downloaded file, `sfbay.csv`, starts with a row of variable names followed by a row of units, so the first two lines are skipped during import and simpler variable names are substituted for the originals. Also, only a subset of stations and years is used in order to keep `sfbay.csv` small:

```
> sfbay <- read.csv("sfbay.csv", header = FALSE, as.is = TRUE,
+     skip = 2)
> names(sfbay) <- c("date", "time", "stn", "depth", "chl", "dox",
+     "spm", "ext", "sal", "temp", "nox", "nhx")
```

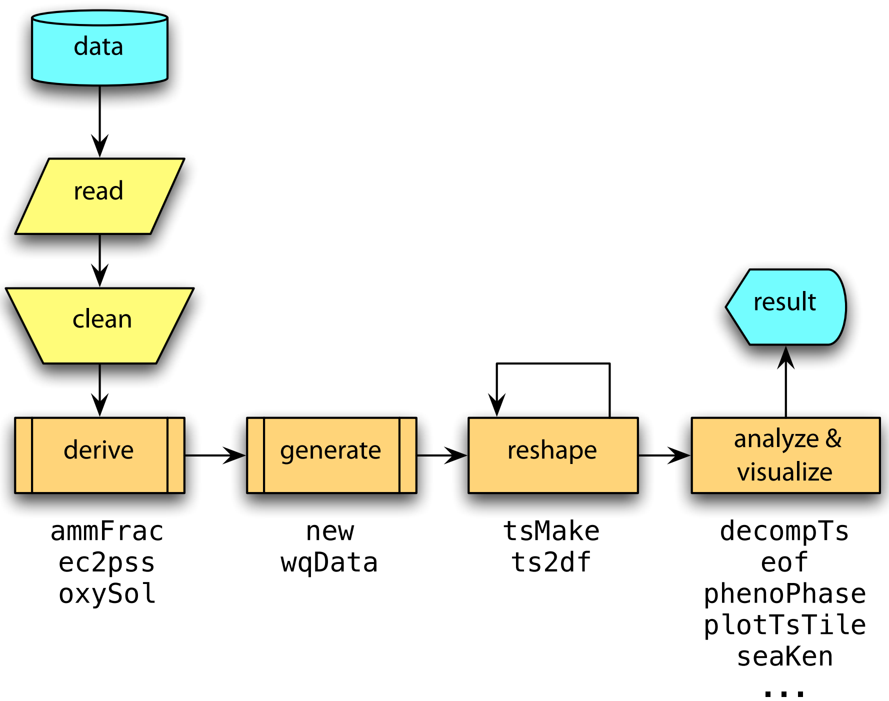

Figure 1: A typical sequence of data analysis. Example functions from the package are listed underneath the corresponding processes in the sequence.

```
> sfbay <- subset(sfbay, stn %in% c(21, 24, 27, 30, 32, 36) & substring(date,
+     7, 10) %in% 1985:2004)
```

The resulting data frame `sfbay` is provided as part of the package, and its contents are explained in the accompanying help file.

```
> head(sfbay)

          date time stn depth chl dox spm ext   sal temp nox nhx
6835 1/23/1985 1120  21     1 5.6  NA  17 1.6 28.15   NA  NA  NA
6836 1/23/1985 1120  21     2 3.4  NA  17 1.6 28.58   NA  NA  NA
6837 1/23/1985 1120  21     6 3.1  NA  18 1.6 28.91   NA  NA  NA
6838 1/23/1985 1120  21    12 3.4  NA  21 1.9 29.36   NA  NA  NA
6841 1/23/1985 1222  24     1 6.2  NA  17 1.6 27.42   NA  NA  NA
6842 1/23/1985 1222  24     2 5.6  NA  18 1.6 27.42   NA  NA  NA
```

The next step is to add any necessary derived variables to the data frame. An initial data set will sometimes contain conductivity rather than salinity data, and we might want to use `ec2pss` to derive the latter. That's not the case here, but let's assume that we want dissolved oxygen as percent saturation rather than in concentration units. Using `oxySol` and the convention of expressing percent saturation with respect to surface pressure:

```
> x <- sample(1:nrow(sfbay), 10)
> sfbay[x, "dox"]

 [1]  9.6   NA   NA  8.4   NA   NA 10.4   NA  7.2   NA

> sfbay1 <- transform(sfbay, dox = round(100 * dox/oxySol(sal,
+     temp), 1))
> sfbay1[x, "dox"]

 [1] 144.0    NA    NA 115.5    NA    NA 126.7    NA 106.9    NA
```

Aside from `ec2pss` and `oxySol`, the function `ammFrac` is available for estimating the fraction of total ammonium in the un-ionized form.

As will be seen below, much of the manipulation work needed to form the `WqData` object is taken care of by a generating function in the package, and there is really nothing more that needs to be done. In fact, not even the renaming of the variables was necessary: only the initial `read.csv` function was required. This is partly due to the way the original data were formatted in the downloaded file and more work may be needed in other cases.

3 The WqData class

We define a standardized format for water quality data by creating a formal (S4) class, the `WqData` class, that enforces the standards, and an accompanying generating function `wqData`. The generating function acts on the suitably-modified data frame and constructs a `WqData` object. The `WqData` object is just a simple extension or subset of the `data.frame` and can be treated as such. The only restrictions it makes is in the column names and classes.

We decided to accommodate two types of sampling `time`, namely, the date either with or without the time of day. The former are converted to the `POSIXct` class and the latter to the `Date` class. A special class `DateTime` is created, which is the union of these two time classes. This was done because the use of classes that combine date and time of day require an additional level of care with respect to time zone (Grothendieck and Petzoldt 2004). Almost all analyses of these low-frequency sampling programs are concerned only with the date, and this additional burden and possible source of error seems unwarranted when not necessary.

Surface location is specified by a `site` code, as the intention is to handle discrete monitoring programs as opposed to continuous transects. Latitude-longitude and distances from a fixed point are implicit in the `site` code and can be recorded in a separate table (see `sfbayVars`). The `depth` is specified separately as a number. Other information that may not be depth-specific, such as the mean vertical extinction coefficient in the near-surface layer, can be coded by a negative depth number. The last two fields in the data portion of a `WqData` object are the `variable` code and the `value`. The variables are given as character strings and the values as numbers. As in the case of the sampling site, additional information related to the variable code can be maintained in a separate table (see `sfbayVars`).

4 Creating a WqData object

Like all S4 classes, `WqData` has a generating function called `new` automatically created along with the class. This function, however, requires that its data frame argument already have a fairly restricted form of structure. In order to decrease the manipulation required of the imported data, a separate, less restrictive generating function called `wqData` is available. This function is more forgiving of field names and classes and does a few other "cleanup" tasks with the data before calling `new`. Perhaps most useful, it converts data from a "wide" format with one field per variable into the "long" format used by the `WqData` class. For example, `sfbay` can be converted to a `WqData` object with a single command:

```
> sfb <- wqData(sfbay, c(1, 3:4), 5:12, site.order = TRUE, type = "wide",
+     time.format = "%m/%d/%Y")
> head(sfb)
```

```
        time site depth variable value
1 1985-01-23  s21     1      chl   5.6
2 1985-01-23  s21     2      chl   3.4
3 1985-01-23  s21     6      chl   3.1
4 1985-01-23  s21    12      chl   3.4
5 1985-01-23  s24     1      chl   6.2
6 1985-01-23  s24     2      chl   5.6
```

There is a `summary` method for this class that tabulates the number of observations by site and variable, as well as the mean and quartiles for individual variables:

```
> summary(sfb)
```

```
date range:  1985-01-23 to 2004-12-14

$observations

       chl  dox  spm  ext  sal temp  nox  nhx
  s21 5164 3673 3903  159 5379 5385  135  135
  s24 3340 2246 2405  146 3485 3480  123  123
  s27 3927 2676 2848  150 4119 4118  142  142
  s30 4496 2922 3106  147 4725 4720  165  164
  s32 3560 2608 2763  129 3786 3777  141  141
  s36 1576 1380 1438   23 1678 1676  101  101

$quartiles
     Min. 1st Qu. Median   Mean 3rd Qu.   Max.
chl  0.10   2.100   3.70  7.479   7.600 221.20
dox  4.10   7.200   8.00  8.140   8.800  15.90
spm  1.00  11.000  20.00 34.050  35.000 983.00
ext  0.20   1.200   1.50  1.762   1.900  12.70
sal  3.80  22.330  26.78 25.330  29.570  32.59
temp 7.24  12.890  15.12 15.500  17.890  24.61
nox  0.01  12.380  22.69 28.550  39.220 247.80
nhx  0.01   2.252   5.14  5.525   8.398  20.78
```

Plotting a "`WqData`" object produces a page for each variable specified, each page containing a strip plot of the values for each site (Figure 2). If no variables are specified, then the first 10 will be plotted:

```
> plot(sfb, vars = c("chl", "dox", "spm"), num.col = 2)
```

Apart from `summary` and `plot`, existing methods for data frames will produce an object of class "`data.frame`" rather than one of class "`WqData`".

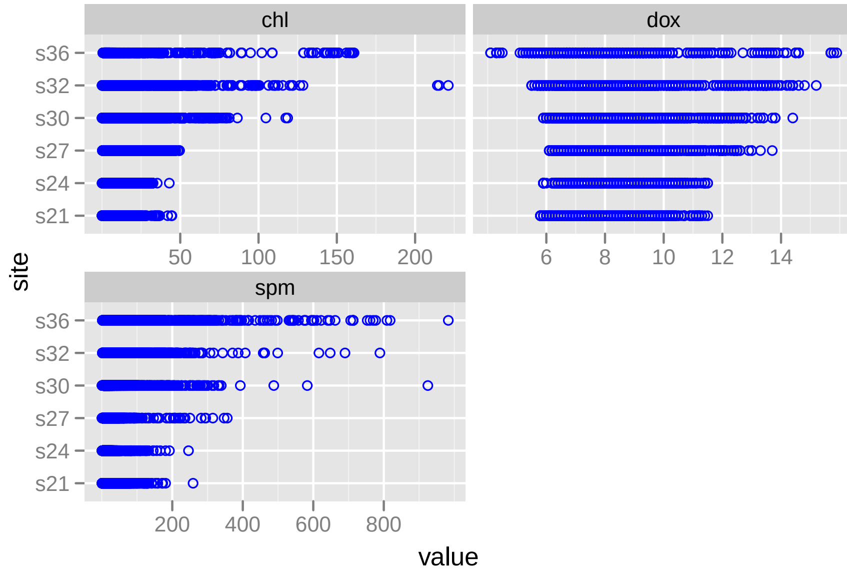

Figure 2: Plotting specific variables of a "WqData" object, in this case chlorophyll-a, dissolved oxygen, and suspended particulate matter.

5 Reshaping

Historical water quality data are often suitable for analyzing as monthly time series, which permits the use of many existing time series functions. `tsMake` is a function for `WqData` objects that creates monthly time series for all variables at a single site or for a single variable at all sites, when the option `type = "ts.mon"`. All replicates are first averaged and then the mean is found for the depth layers of interest. If no layers are specified, all depths will be used. The `layer` argument allows for flexibility in specifying depths, including negative depths used as codes for, say, "near botton" or "entire water column". The default time series plot is convenient for a quick look at the series (Figure 3):

```
> y <- tsMake(sfb, focus = "chl", layer = c(0, 5))
> y[1:6, ]
```

```
          s21       s24       s27       s30       s32    s36
[1,] 4.500000  5.900000       NaN  1.300000  2.650000  6.250
[2,]      NaN       NaN       NaN  1.600000  5.550000    NaN
[3,] 5.858333 10.654167 12.291667 12.787500 11.866667 40.100
[4,] 4.638889  5.916667  8.133333  8.388889 11.455556  4.525
[5,] 2.575000  2.058333  1.566667  1.183333  1.725000    NaN
[6,] 3.025000  1.875000  1.441667  1.133333  1.641667  3.000
```

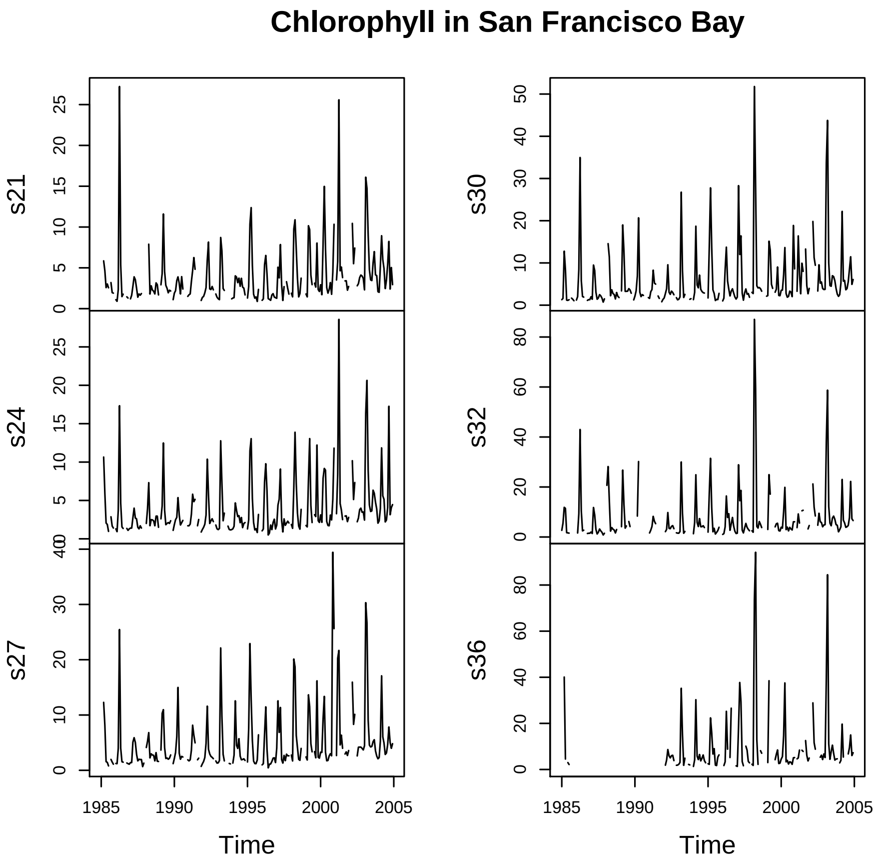

Figure 3: Monthly mean chlorophyll (μg L^{-1}) in 0-5 m layer of San Francisco Bay.

```
> tsp(y)

[1] 1985.000 2004.917   12.000

> plot(y, main = "Chlorophyll in San Francisco Bay")
```

If the option `type = "zoo"`, then `tsMake` produces an object of class `"zoo"` containing values by date of observation, rather than a monthly time series.

```
> head(tsMake(sfb, focus = "chl", layer = c(0, 5), type = "zoo"))

              s21      s24       s27       s30       s32  s36
1985-01-23  4.500  5.90000       NaN  1.300000  2.650000  6.25
1985-02-27    NaN      NaN       NaN  1.600000  5.550000   NaN
```

```
1985-03-07  4.800  3.90000  5.200000  5.033333  5.166667   NaN
1985-03-13  2.600  9.35000  7.066667  5.066667  4.500000   NaN
1985-03-21    NaN  7.70000 13.300000 10.200000  4.700000   NaN
1985-03-29 10.175 21.66667 23.600000 30.850000 33.100000 40.10
```

There are several functions for further reshaping of time series, which prepare them for use in specific analyses. `ts2df` converts a monthly time series vector to a year × month data frame. Leading and trailing empty rows are removed, additional rows with missing data are optionally removed, and the data frame can be reconfigured to represent a local "water year":

```
> chl27 <- sfbayChla[, "s27"]
> tsp(chl27)

[1] 1978.000 2009.583   12.000

> chl27 <- round(chl27, 1)
> head(ts2df(chl27))

     Jan Feb Mar  Apr  May Jun Jul Aug Sep Oct Nov Dec
1978 1.1 2.8 5.5  2.7  3.4 1.9 1.6  NA 1.7 2.1 2.2 1.7
1979 1.9 1.8 2.4  3.8  2.3 4.8 1.6 3.9 2.1 1.2 1.1  NA
1980 1.3 1.9 2.1 10.2  3.4 2.1 1.1 1.4 1.6 1.4 1.7 1.3
1981  NA 1.7 2.0  9.1   NA  NA  NA  NA  NA  NA  NA  NA
1982 2.8 4.5 6.5  9.3  8.2 3.4 1.4  NA 2.1 1.8 1.7 1.0
1983  NA 1.4 7.0 16.4 16.6 5.4 1.4 1.7 2.0 1.5 1.5 1.4
```

Another example of its use is shown in Section 6.2 below. Another similar reshaping function is `mts2ts`, which converts a matrix time series to a vector time series for various analyses. It first aggregates the multivariate matrix time series by year, then converts it to a vector time series in which the "seasons" correspond to these annnualized values for the original variables. The `seas` parameter enables focusing the subsequent analysis on seasons of special interest, or to ignore seasons where there are too many missing data. The function can be used in conjunction with `seaKen` to conduct a Regional Kendall trend analysis, as described in Section 6.1 below:

```
> y <- window(sfbayChla, s = 2005, end = c(2009, 12))
> mts2ts(y, seas = 2:4)

Time Series:
Start = c(2005, 1)
End = c(2009, 16)
Frequency = 16
 [1]  5.830000  4.726667  6.013333  4.576667  5.543333  5.606667  5.873333
```

```
 [8]  6.286667  6.453333  7.633333  7.463333  7.810833  8.453333  8.766667
[15]  8.030000  8.410000 18.064444  9.764444 11.970000 12.451111 12.804444
[22] 16.178889 18.130000 20.582222 22.477778 26.878889 26.354444 29.867778
[29] 31.061111 33.715556 32.142222 33.153333  7.884444  6.583333  7.786667
[36]  7.880000  7.878889  9.237778 10.136667 10.234444 10.537778 11.924444
[43] 12.008889 12.078889 13.150000 12.951667 13.016667 15.060000 15.074444
[50] 10.948889 12.485556 13.802222 14.096667 14.777778 15.935556 16.969444
[57] 16.746667 20.165556 21.036667 21.838889 22.320000 23.547222 23.382222
[64] 24.001111  4.661111  4.491111  4.640000  4.671111  4.672222  4.434444
[71]  4.634444  5.232222  5.393333  5.983333  6.820000  7.706667  8.453333
[78]  9.126667  8.067778  8.104444
```

6 Analyzing

6.1 Trends

The function `mannKen` does a Mann-Kendall test of trend on a time series and provides the corresponding nonparametric slope estimate. Because of serial correlation for most monthly time series, the significance of such a trend is often overstated and `mannKen` is better suited for annual series, such as this one for Nile River flow:

```
> mannKen(Nile)

$sen.slope
[1] -2.6

$sen.slope.pct
[1] -0.2828085

$p.value
[1] 3.658263e-05

$S
[1] -1387

$varS
[1] 112728.3

$miss
[1] 0
```

`mannKen` can also handle matrix time series, with options for plotting trends in the original units per year, as percent per year, or as Kendall's tau. The first option is suitable when time series are all in the same units, such as chlorophyll-a measurements from different stations. The second makes sense with variables of different units but is not suitable for variables that can span zero (e.g., sea level, or temperature in °C). The last option can always be used but measures the strength of the correlation with time rather than the trend level. Plotted variables can be ordered by the size of their trends, and both statistical significance and excessive missing data are mapped to point colour and shape (see discussion of `seasonTrend` below). When aggregating monthly series to produce an annual series for trend testing, there is a utility function `tsSub` that allows subsetting the months beforehand. It can be useful for avoiding months with many missing data, or to focus attention on a particular time of year (Figure 4):

```
> y <- sfbayChla
> y1 <- tsSub(y, seas = c(1:7, 9:11))
> y2 <- aggregate(y1, 1, mean, na.rm = FALSE)
> mannKen(y)
```

```
     sen.slope sen.slope.pct      p.value     S      varS  miss       tau
s21 0.09190635      2.226021 3.452421e-15 14638 3456081.3 0.355 0.2978775
s22 0.07876923      2.117911 5.211146e-14 13993 3455981.7 0.355 0.2847520
s23 0.09500000      2.278672 9.760948e-17 13359 2585387.7 0.229 0.3300964
s24 0.08837838      2.226759 2.568569e-18 17562 4047306.7 0.204 0.3215600
s25 0.08608696      1.866458 6.861539e-15 15103 3761212.3 0.229 0.2904256
s26 0.09054219      1.751421 8.957084e-12 11202 2695548.7 0.204 0.2691753
s27 0.09766450      1.936613 6.308045e-16 16116 3974588.7 0.204 0.2986878
s28 0.09857143      1.761713 1.549608e-12 11190 2504648.0 0.236 0.2824260
s29 0.09471782      1.694002 1.947846e-13 14326 3796116.7 0.204 0.2737836
s30 0.11876543      1.910050 3.524414e-16 16257 3974652.3 0.204 0.3013011
s31 0.13363636      1.767366 2.822022e-11  9221 1919071.7 0.377 0.2781347
s32 0.14047619      1.817045 7.665073e-14 12917 2984859.0 0.343 0.2899374
s33 0.15735099      1.697400 1.664931e-08  4986  780210.7 0.539 0.2747864
s34 0.15866621      1.721436 5.774792e-10  7135 1325498.3 0.551 0.2757168
s35 0.14066225      1.398234 8.246916e-07  3821  600527.7 0.617 0.2509688
s36 0.14956522      1.577695 1.049397e-08  6204 1174965.3 0.589 0.2598969
```

```
> mannKen(y, plot = TRUE, type = "pct", order = TRUE)
```

A main role for `mannKen` in this package is as a support function for the Seasonal Kendall test of trend (Hirsch et al. 1982, Helsel and Hirsch 2002). The Seasonal Kendall test combines information about trends for individual months (or some other subdivision of the year such as quarters) and produces an overall test of trend for a series. `mannKen` collects certain information on the pattern of missing data that is

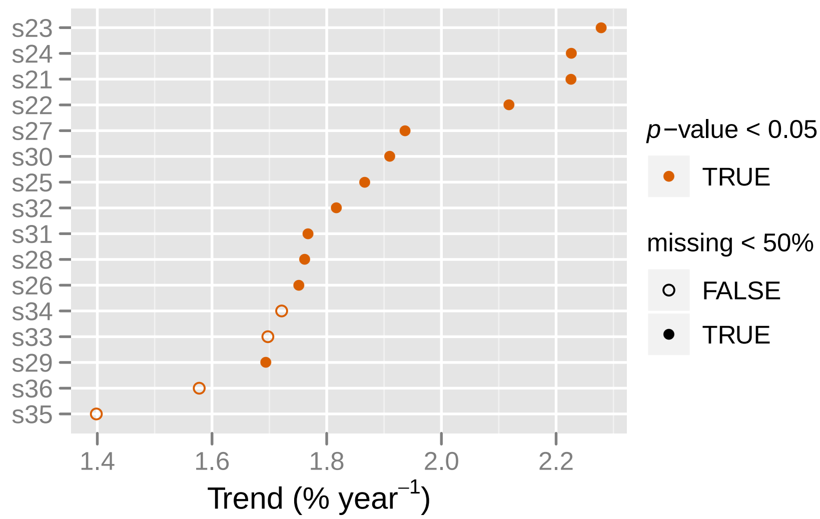

Figure 4: Chlorophyll-a (μg L^{-1} year^{-1}) trends in San Francisco Bay.

then used to determine if a Seasonal Kendall test is warranted. In particular, there is an option to report a result only if more than half the seasons are each missing less than half the possible comparisons between the first and last 20% of the years (Schertz et al. 1991):

```
> chl27 <- sfbayChla[, "s27"]
> seaKen(chl27)

$sen.slope
[1] 0.1083333

$sen.slope.pct
[1] 2.148168

$p.value
[1] 1.117981e-25

$miss
    1     2     3     4     5     6     7     8     9    10    11    12
0.286 0.000 0.000 0.000 0.265 0.265 0.265 0.429 0.143 0.143 0.286 0.429
```

An important role, in turn, for `seaKen` in this package is as a support function for `seaRoll`, which applies the Seasonal Kendall test to a rolling window of years, such as a decadal window. There is an option to plot the results of `seaRoll`. `seaKen` is subject to distortion by correlation among months, but the relatively small number of years per window in typical use does not allow for an accurate correction:

```
> seaRoll(chl27, w = 10)
```

```
     sen.slope sen.slope.pct p.value
1987    0.0000         0.000   1.000
1988    0.0258         0.760   0.357
1989        NA            NA      NA
1990        NA            NA      NA
1991        NA            NA      NA
1992    0.0400         1.090   0.078
1993        NA            NA      NA
1994        NA            NA      NA
1995    0.0400         1.010   0.126
1996   -0.0217        -0.567   0.525
1997   -0.0364        -0.900   0.305
1998        NA            NA      NA
1999        NA            NA      NA
2000    0.1380         2.720   0.006
2001        NA            NA      NA
2002        NA            NA      NA
2003    0.2700         4.440   0.000
2004    0.2870         4.570   0.000
2005    0.3160         5.120   0.000
2006    0.2600         3.800   0.000
2007    0.3160         4.380   0.000
2008    0.3090         4.160   0.000
2009        NA            NA      NA
```

The Seasonal Kendall test is not informative when trends for different months differ in sign. The function `seasonTrend` enables visualization of individual monthly trends and can be helpful for, among other things, deciding on the appropriateness of the Seasonal Kendall test. The Theil-Sen slopes are shown along with an indication, using dot colour, of the Mann-Kendall test of significance. The dot shape (filled or empty) indicates whether the proportion of missing values in the first and last fifths of the data is < 0.5 or not (Figure 5).

```
> x <- sfbayChla
> seasonTrend(x, plot = TRUE, ncol = 4, scales = "free_y")
```

The function `trendHomog` can also be used to test directly for the homogeneity of seasonal trends (van Belle and Hughes 1984):

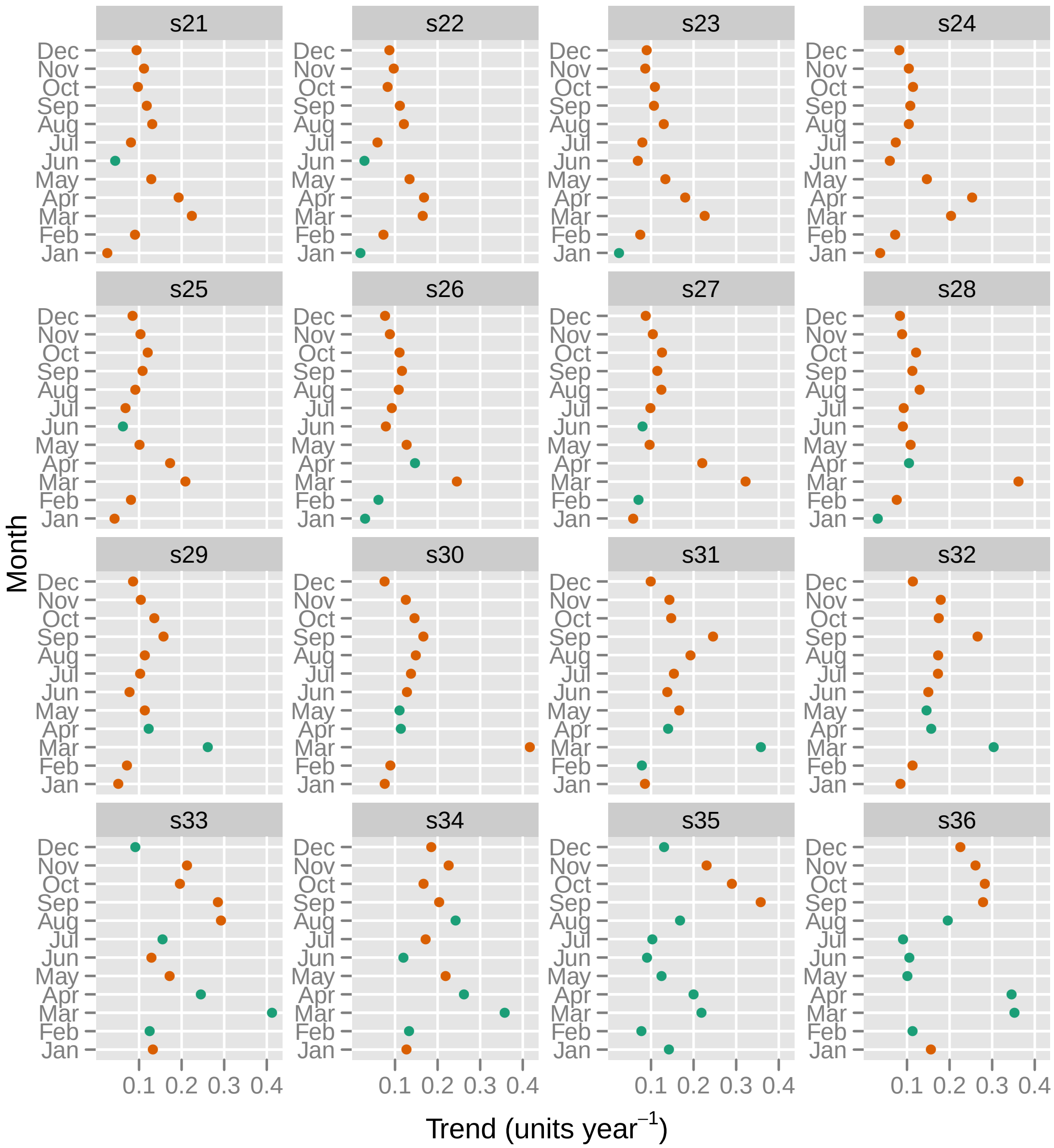

Figure 5: Mann-Kendall tests of chlorophyll trends for individual months at stations in San Francisco Bay. Trends are expressed in original units per year, in this case μg L^{-1} year^{-1}.

```
> x <- sfbayChla[, "s27"]
> trendHomog(x)

$chi2.trend
[1] 118.4498

$chi2.homog
[1] 10.31347

$p.value
[1] 0.5024304
```

A Regional Kendall test is similar to a Seasonal Kendall test, with annual data for multiple sites instead of annual data for multiple seasons (Helsel and Frans 2006). The function `mts2ts` (Section 5) facilitates transforming an annual matrix time series into the required vector time series for `seaKen`, with stations playing the role of seasons. As with seasons, correlation among sites can inflate the apparent statistical significance, so the test is best used with stations from different subregions that are not too closely related, unlike the following example:

```
> chl <- sfbayChla[, 1:12]
> seaKen(mts2ts(chl, 2:4))

$sen.slope
[1] 0.2155

$sen.slope.pct
[1] 2.379796

$p.value
[1] 4.539847e-24

$miss
    1     2     3     4     5     6     7     8     9    10    11    12
0.286 0.286 0.143 0.000 0.286 0.000 0.000 0.000 0.000 0.000 0.286 0.286
```

6.2 Empirical Orthogonal Functions

Empirical Orthogonal Function (EOF) analysis is a term used primarily in the earth sciences for principal component analysis applied to simultaneous time series at different spatial locations. Hannachi et al. (2007) provides a recent comprehensive summary. The function `eof` in this package, based on `prcomp` in the `stats` package, scales the time series and applies a promax rotation to the EOFs.

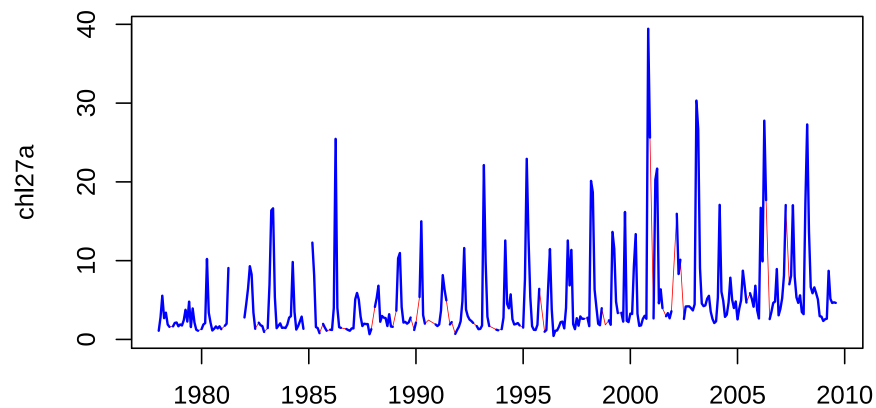

Figure 6: Interpolation of a monthly time series (interpolated data in *red*).

`eof` does not permit `NA`s and some kind of data imputation or omission will usually be required. The function `interpTs` is handy for small data gaps. Here, we use it to bridge gaps of up to three months. The interpolated series is then plotted in red and the original series overplotted in blue (Figure 6).

```
> chl27 <- sfbayChla[, "s27"]
> chl27a <- interpTs(chl27, gap = 3)
> plot(chl27a, col = "red", lwd = 0.5, xlab = "")
> lines(chl27, col = "blue", lwd = 1.5)
```

`eof` requires an estimate of the number of EOFs to retain for rotation. `eofNum` provides a guide to this number by plotting the eigenvalues and their confidence intervals in a "scree" plot. The significance of each eigenvalue is also assessed using *rule N*, which repeatedly computes eigenvalues of the correlation matrix for an appropriately-sized random variable matrix and returns the 0.95 quantiles. Here, we apply `eofNum` to annualized San Francisco Bay chlorophyll data and retain the stations with no missing data, namely, the first 12 stations.

```
> chla1 <- aggregate(sfbayChla, 1, mean, na.rm = TRUE)
> chla1 <- chla1[, 1:12]
> eofNum(chla1, distr = "lognormal", reps = 2000)
```

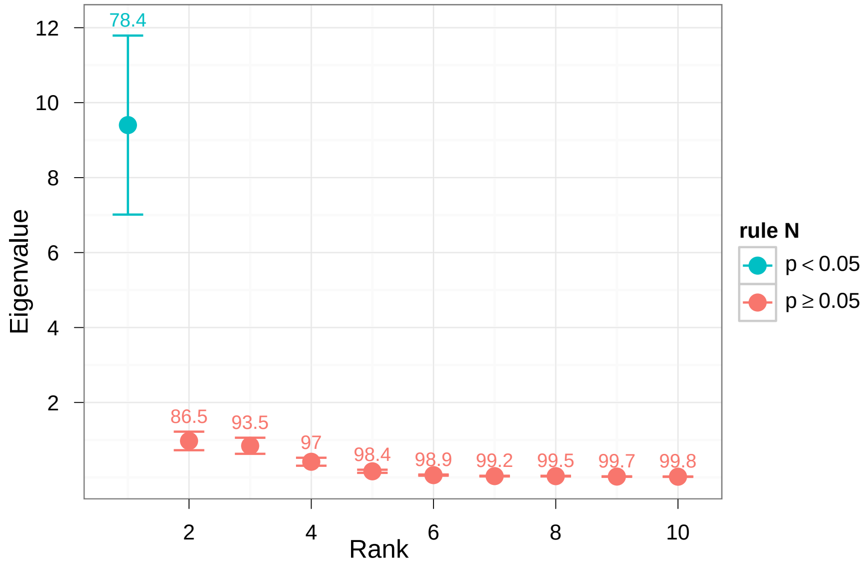

Figure 7: Eigenvalues of the San Francisco Bay chlorophyll time series matrix.

These stations have similar coefficients for the first EOF and appear to act as one with respect to chlorophyll variability on the annual scale (Figure 7). It suggests that further exploration of the interannual variability of these stations can be simplified by using a single time series, namely, the first EOF.

```
> e1 <- eof(chla1, n = 1)
> e1

$REOF
    id      EOF1
1  s21 0.2984840
2  s22 0.2875436
3  s23 0.3074099
4  s24 0.3038324
5  s25 0.3013699
6  s26 0.2686399
7  s27 0.3116476
8  s28 0.2791966
9  s29 0.3042674
10 s30 0.2931426
11 s31 0.2549798
12 s32 0.2445793
```

```
$amplitude
     id        EOF1
1  1978 -3.71779761
2  1979 -3.31653011
3  1980 -3.66943342
4  1981 -2.94304599
5  1982 -2.72889938
6  1983  0.05732382
7  1984 -2.02038749
8  1985 -1.89260439
9  1986 -0.30543129
10 1987 -4.18310354
11 1988 -2.38621346
12 1989 -1.07971835
13 1990 -0.90950909
14 1991 -3.05696910
15 1992 -2.71675623
16 1993 -0.64605278
17 1994 -2.17668147
18 1995  2.32949446
19 1996 -2.59126388
20 1997  0.86074181
21 1998  3.36503739
22 1999  2.70185298
23 2000  3.23687896
24 2001  2.78555990
25 2002  1.53489367
26 2003  5.42194986
27 2004  1.40560267
28 2005  0.58759133
29 2006  6.27913918
30 2007  3.92929848
31 2008  5.84503308

$eigen.pct
 [1] 78.4  8.1  7.0  3.5  1.4  0.5  0.3  0.3  0.2  0.2  0.1  0.1

$variance
[1] 78.4
```

The function `eofPlot` produces a graph of either the EOFs or their accompanying time series. In this case, with `n = 1`, there is only one plot for each such graph (Figure 8).

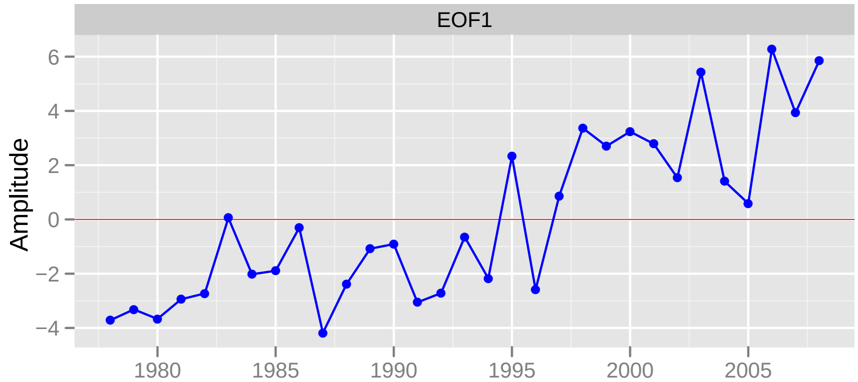

Figure 8: Time series for the first EOF of the San Francisco Bay chlorophyll time series matrix.

```
> eofPlot(e1, type = "amp")
```

Principal component analysis can also be useful in studying the way different seasonal "modes" of variability contribute to overall year-to-year variability of a single time series (Jassby 1999). The basic approach is to consider each month as determining a separate annual time series and then to calculate the eigenvalues for the resulting $12 \times n$ years time series matrix. The function `ts2df` is useful for expressing a monthly time series in the form needed by `eof`. For example, the following code converts the monthly chlorophyll time series for Station 27 in San Francisco Bay to the appropriate data frame with October, the first month of the local "water year", in the first column, and years with missing data omitted:

```
> chl27b <- interpTs(sfbayChla[, "s27"], gap = 3)
> chl27b <- ts2df(chl27b, mon1 = 10, addYr = TRUE, omit = TRUE)
> head(round(chl27b, 1))
```

```
     Oct Nov Dec Jan Feb Mar  Apr  May Jun Jul Aug Sep
1979 2.1 2.2 1.7 1.9 1.8 2.4  3.8  2.3 4.8 1.6 3.9 2.1
1980 1.2 1.1 1.2 1.3 1.9 2.1 10.2  3.4 2.1 1.1 1.4 1.6
1983 1.8 1.7 1.0 1.2 1.4 7.0 16.4 16.6 5.4 1.4 1.7 2.0
1984 1.5 1.5 1.4 1.9 2.8 3.0  9.8  3.5 1.2 1.7 2.3 2.9
1986 1.5 1.1 1.2 1.2 1.2 4.0 25.5  4.0 1.5 1.5 1.4 1.4
1987 1.3 1.2 1.1 1.4 1.4 5.1  5.9  5.1 2.9 1.7 2.0 2.0
```

The following example plots the EOFs from an analysis of this month $\times$ year data frame for Station 27 chlorophyll. `eofNum` (not shown) suggested retaining up to

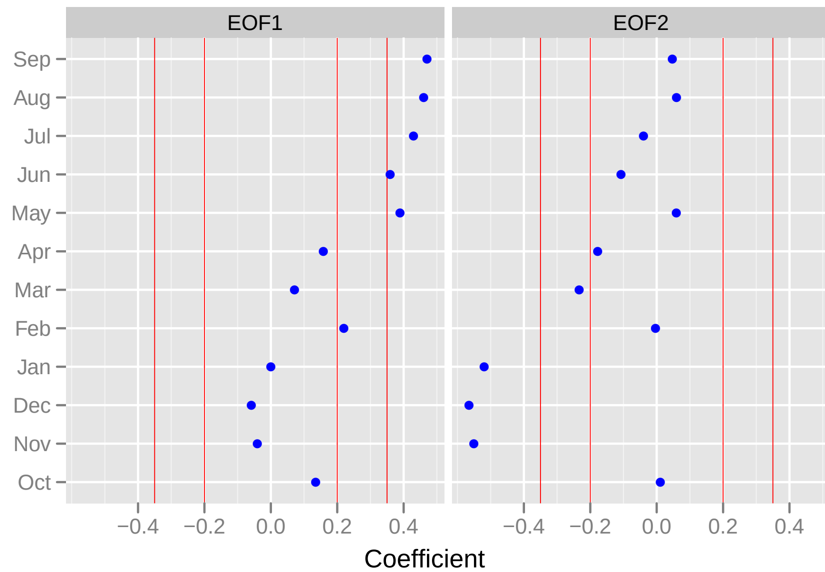

Figure 9: Rotated EOFs for the San Francisco Bay Station 27 month $\times$ year chlorophyll time series.

two EOFs. The resulting rotated EOFs imply two separate modes of variability for further exploration, the first operating during May-Sep and the other during Nov-Jan (Figure 9). The red lines represent an approximate range of thresholds for statistical significance when sufficient data are used:

```
> e2 <- eof(chl27b, n = 2)
> eofPlot(e2, type = "coef")
```

6.3 Time series decomposition

An analysis of chlorophyll *a* time series from many coastal and estuarine sites around the world demonstrates that the standard deviation of chlorophyll is approximately proportional to the mean, both among and within sites, as well as at different time scales (Cloern and Jassby 2010). One consequence is that these monthly time series are well described by a multiplicative seasonal model: $c_{ij} = Cy_i m_j \epsilon_{ij}$, where c_{ij} is chlorophyll concentration in year i and month j; C is the long-term mean; y_i is the annual effect; m_j is the average seasonal (in this case monthly) effect; and ϵ_{ij} is the residual series, which we sometimes refer to as the "events" component. The annual effect is simply the annual mean $Y_i = (1/12)\sum_{j=1}^{12} c_{ij}$ divided by the long-term mean: $y_i = Y_i/C$. The average monthly effect is given by $m_j = (1/N)\sum_{i=1}^{N} M_{ij}/Y_i$, where

M_{ij} is the value for month j in year i, and N is the total number of years. The events component is then obtained by $\epsilon_{ij} = c_{ij}/Cy_i m_j$. This simple approach is motivated partly by the observation that many important events for estuaries (e.g., persistent dry periods, species invasions) start or stop suddenly. Smoothing to extract the annualized term, which can disguise the timing of these events and make analysis of them unnecessarily difficult, is not used.

The `decompTs` listed here accomplishes this multiplicative decomposition (an option allows additive decomposition as an alternative). It requires input of a time series matrix in which the columns are monthly time series. It allows missing data, but it is up to the user to decide how many data are sufficient and if the pattern of missing data will lead to bias in the results. If so, it would be advisable to eliminate problem years beforehand by setting all month values to `NA` for those years. There are two cases of interest here: one in which the seasonal effect is held constant from year to year, and another in which it is allowed to vary by not distinguishing a separate events component. The choice is made by setting `event = TRUE` or `event = FALSE`, respectively, in the input. If no specific starting or ending year is given, the input data will be extended to cover January of the earliest or December of the latest year, respectively. The output of this function is a matrix time series containing the original time series and its multiplicative model components.

The average seasonal pattern may not resemble observed seasonality in a given year. Patterns that are highly variable from year to year will result in an average seasonal pattern of relatively low amplitude (i.e., low range of monthly values) compared to the amplitudes in individual years. An average seasonal pattern with high amplitude therefore indicates both high amplitude and a recurring pattern for individual years. The default time series `plot` again provides a quick illustration of the result (Figure 10):

```
> chl27 <- sfbayChla[, "s27"]
> d1 <- decompTs(chl27)
> plot(d1, nc = 1, main = "Station 27 Chl-a decomposition")
```

The average seasonal pattern does not provide any information about potential secular trends in the pattern. A solution is to apply the decomposition to a moving time window. The window should be big enough to yield a meaningful average of interannual variability but short enough to allow a trend to manifest. This may be different for different systems, but a decadal window can be used as a starting point. A more convenient, albeit restrictive, way to examine changing seasonality is with the dedicated function `plotSeason`. It divides the time period into equal intervals and plots a composite of the seasonal pattern in each interval. It also warns of months that may not be represented by enough data by colouring them red (Figure 11). `plotSeason` is an easy way to decide on the value for the `event` option in `decompTs`.

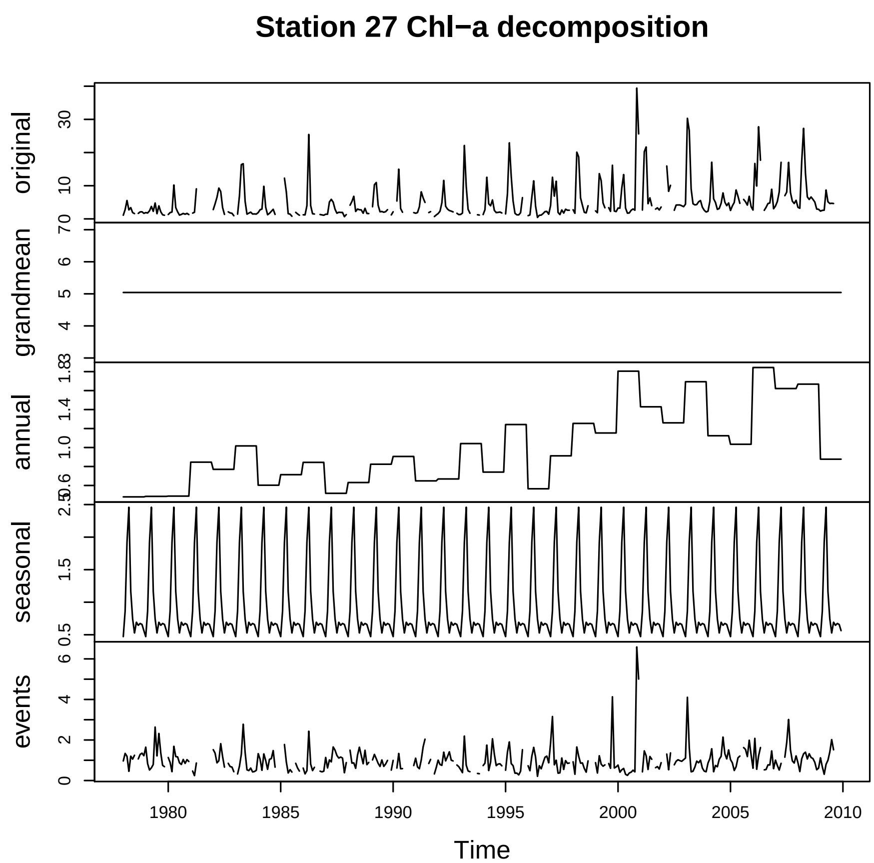

Figure 10: Multiplicative decomposition of chlorophyll at Station 27 in San Francisco Bay.

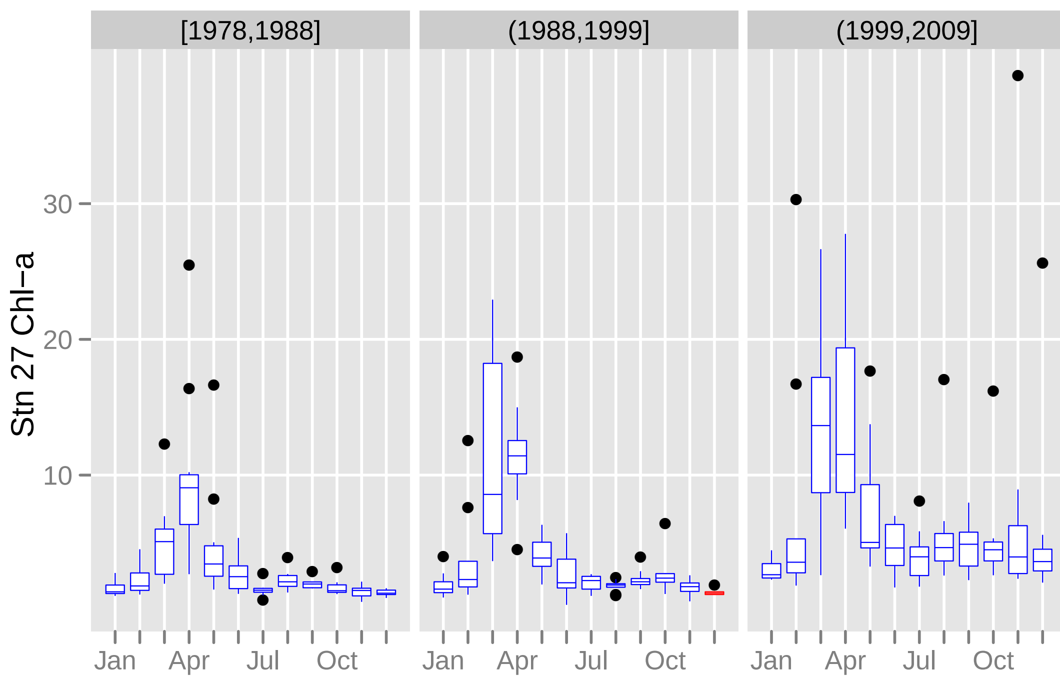

Figure 11: Composites of seasonal pattern in `chl27` for three multi-year intervals in different plots.

```
> plotSeason(chl27, num = 3, same.plot = FALSE, ylab = "Stn 27 Chl-a")
```

The same boxplots can also be combined in one plot, with boxplots for the same month grouped together (Figure 12):

```
> plotSeason(chl27, num = 3, same.plot = TRUE, ylab = "Stn 27 Chl-a")
```

`plotSeason` also has an option to plot all individual months separately for the entire record (Figure 13).

```
> plotSeason(chl27, "by.month", ylab = "Stn 27 Chl-a", scales = "free_y")
```

With all types of seasonal plots, it is often helpful to adjust the device aspect ratio and size manually to get the clearest information.

6.4 Phenological parameters

`phenoPhase` and `phenoAmp` act on monthly time series or dated observations (`"zoo"` objects) and produce measures of the phase and amplitude, respectively, for each year. `phenoPhase` finds the month containing the maximum value, the *fulcrum* or center of gravity, and the weighted mean month. `phenoAmp` finds the range, the range divided by mean, and the coefficient of variation. Both functions can be confined to only part of the year, for example, the months containing the spring phytoplankton bloom. This feature can also be used to avoid months with chronic missing-data problems.

Illustrating once again with chlorophyll observations from Station 27 in San Francisco Bay:

```
> chl27 <- sfbayChla[, "s27"]
> p1 <- phenoPhase(chl27)
> head(p1)

  year max.time fulcrum mean.wt
1 1978       NA      NA      NA
2 1979       NA      NA      NA
3 1980        4    4.52    5.54
4 1981       NA      NA      NA
5 1982       NA      NA      NA
6 1983       NA      NA      NA

> p2 <- phenoPhase(chl27, c(1, 6))
> head(p2)
```

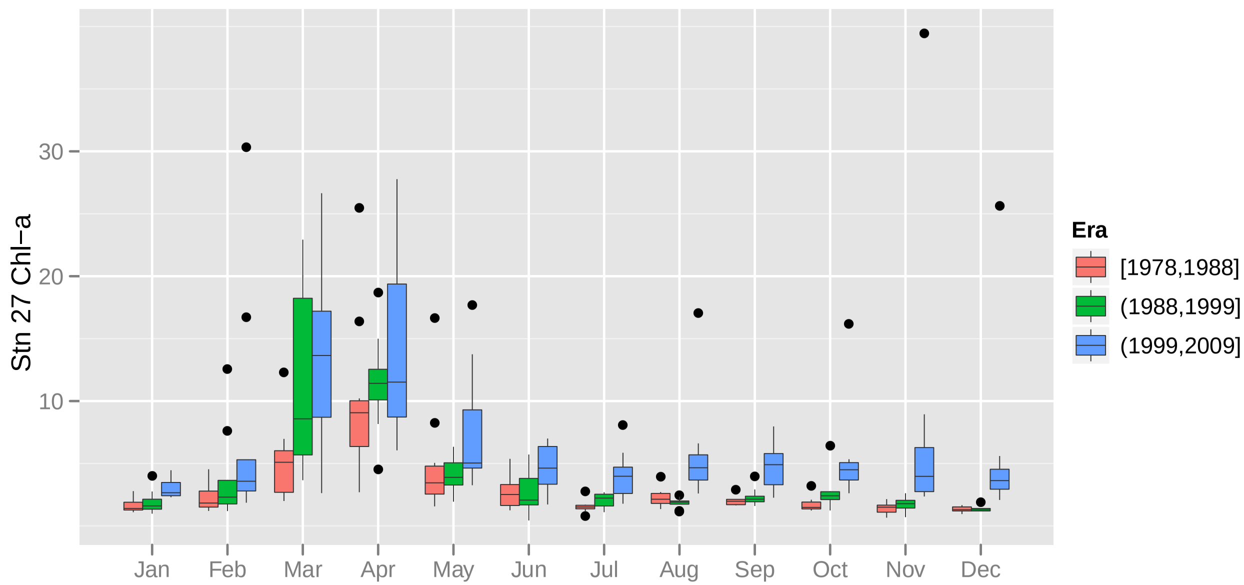

Figure 12: Composites of seasonal pattern in `chl27` for three multi-year intervals in one plot.

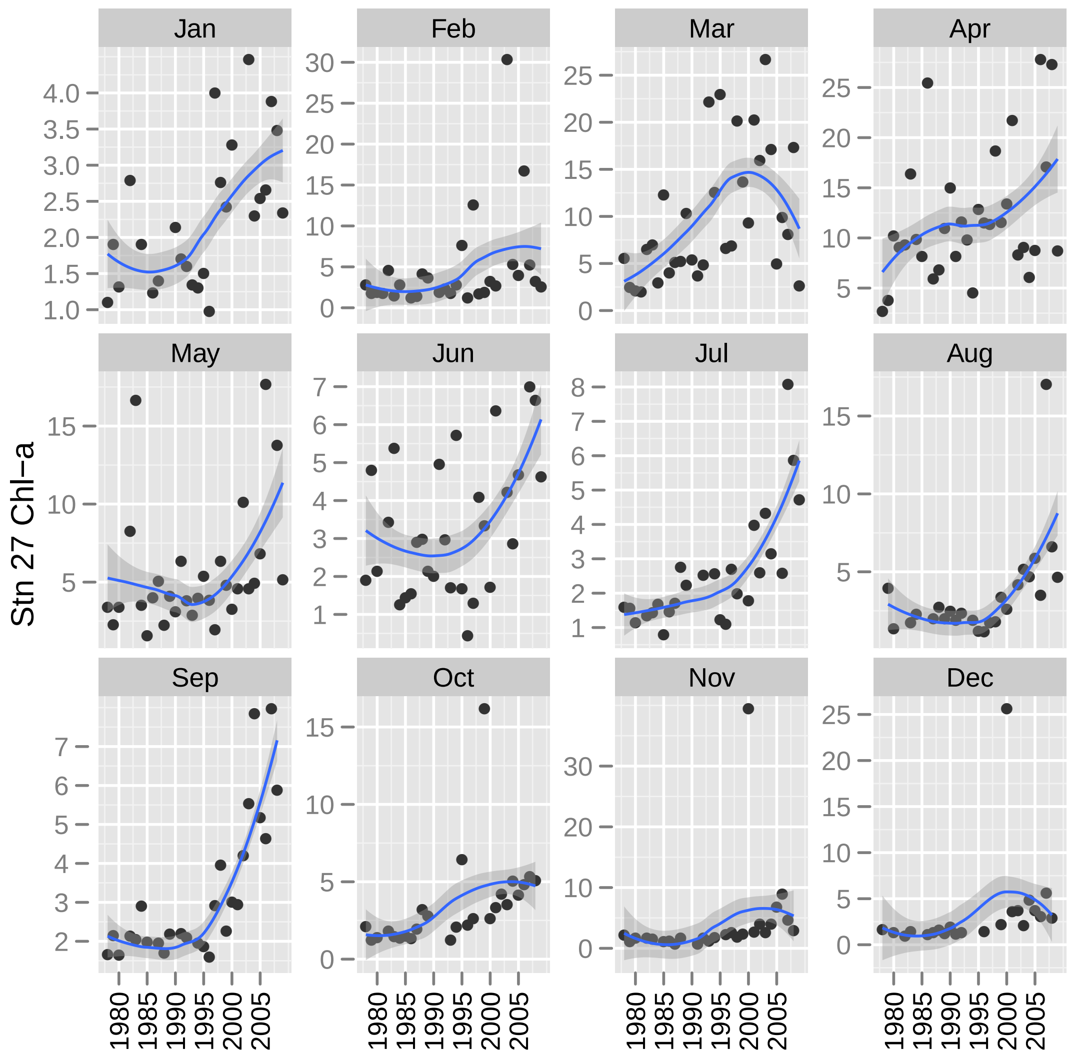

Figure 13: Changing seasonal pattern in `chl27` described by the time series for individual months.

```
  year max.time fulcrum mean.wt
1 1978        3    3.37    3.58
2 1979        6    3.94    4.01
3 1980        4    3.99    3.90
4 1981       NA      NA      NA
5 1982        4    3.86    3.75
6 1983       NA      NA      NA
```

```
> p3 <- phenoAmp(chl27, c(1, 6))
> head(p3)
```

```
  year    range range.mean        cv
1 1978 4.450000   1.530086 0.5228641
2 1979 3.033333   1.074803 0.4260272
3 1980 8.900000   2.538827 0.9578382
4 1981       NA         NA        NA
5 1982 6.509444   1.122560 0.4564730
6 1983       NA         NA        NA
```

Using the actual dated observations:

```
> zchl <- tsMake(sfb, focus = "chl", layer = c(0, 5), type = "zoo")
> head(zchl)
```

```
              s21      s24       s27       s30       s32   s36
1985-01-23  4.500  5.90000       NaN  1.300000  2.650000  6.25
1985-02-27    NaN      NaN       NaN  1.600000  5.550000   NaN
1985-03-07  4.800  3.90000  5.200000  5.033333  5.166667   NaN
1985-03-13  2.600  9.35000  7.066667  5.066667  4.500000   NaN
1985-03-21    NaN  7.70000 13.300000 10.200000  4.700000   NaN
1985-03-29 10.175 21.66667 23.600000 30.850000 33.100000 40.10
```

```
> zchl27 <- zchl[, 3]
> head(phenoPhase(zchl27))
```

```
  year   max.time    fulcrum    mean.wt  n
1 1985 1985-03-29 1985-03-31 1985-04-19 17
2 1986 1986-04-29 1986-04-25 1986-04-27 21
3 1987 1987-04-16 1987-05-13 1987-05-18 20
4 1988 1988-04-14 1988-04-27 1988-06-09 16
5 1989 1989-03-01 1989-04-12 1989-04-12 25
6 1990 1990-04-12 1990-04-30 1990-04-21 13
```

```
> head(phenoPhase(zchl27, c(1, 6), out = "doy"))
```

```
  year max.time fulcrum mean.wt  n
1 1985       88      85      94 11
2 1986      119     111     109 15
3 1987      106     107     107 12
4 1988      105      84      98  7
5 1989       60      86      87 18
6 1990      102     106      98 10
```

```
> head(phenoPhase(zchl27, c(1, 6), out = "julian"))
```

```
  year max.time fulcrum mean.wt  n
1 1985     5566    5563    5572 11
2 1986     5962    5954    5952 15
3 1987     6314    6315    6315 12
4 1988     6678    6657    6671  7
5 1989     6999    7025    7026 18
6 1990     7406    7410    7402 10
```

6.5 Miscellaneous plotting functions

`plotTsTile` plots a monthly time series as a month × year grid of tiles, with color representing magnitude. The data can be binned in either of two ways. The first is simply by deciles. The second, which is intended for log-anomaly data, is by four categories: Positive numbers higher or lower than the mean positive value, and negative numbers higher or lower than the mean negative value. In this version of `plotTsTile`, the anomalies are calculated with respect to the overall mean month.

```
> chl27 <- sfbayChla[, "s27"]
> plotTsTile(chl27)
```

This plot shows clearly the change in autumn-winter chlorophyll magnitude after 1999 (Figure 14).

The `layOut` function is a convenient way to create a graph consisting of two or more plots produced by the **ggplot2** package. The position of each plot is determined by the beginning row and column numbers. The relative size of each plot is determined by the sequence lengths of row and column numbers. The total grid size of the graph is determined automatically by the grid numbers used for individual plots. Manual adjustment of the graphics window may be useful to get proper aspect ratios and prevent text from overlapping (Figure 15):

```
> chl27 = sfbayChla[, "s27"]
> g1 <- plotTsTile(chl27, legend.title = "Chl log-anomaly", square = FALSE)
```

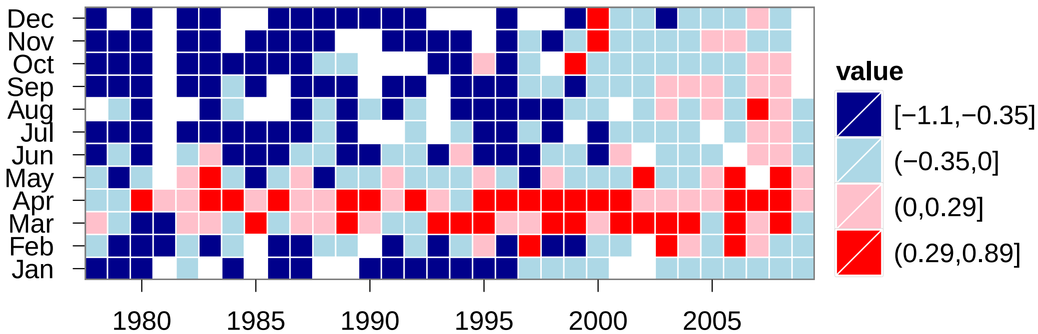

Figure 14: Image plot of monthly log-anomaly time series for Station 27 chlorophyll.

```
> g2 <- seasonTrend(chl27, plot = TRUE, legend = TRUE)
> g3 <- plotSeason(chl27, num = 3, ylab = expression(paste("Chl-",
+     italic(a), ", ", mu * g ~ L^{
+         -1
+     })))
> layOut(list(g1, 1:2, 1:6), list(g2, 1:2, 7:10), list(g3, 3:5,
+     1:8))
```

7 Concluding Remarks

In the near future, this package will remain focused on typical data sets that have accumulated in long-term coastal water quality monitoring programs, namely, those collected at a frequency of about 10^1 to 10^2 times per year at 10^1 to 10^2 sites. Aside from incremental revision and addition of specific functions, the main structural change envisioned is in the class definition for data objects.

In this regard, it is helpful to examine what constitutes a water quality observation, i.e., the essential components of this class. The minimum information typically needed is of four kinds: the location, the time, the analyte and the observed value. As discussed in Section 3, additional information about the location and the analytical method is inherent in the unique codes used for each location and analyte. Sometimes, however, it may be more convenient to include additional information explicitly with the actual observations, such as censoring limits that may change throughout a project. Other complications are introduced by the different ways in which location, time, and even observed values can be recorded. For example, surface location information can come in the form of site names, latitude-longitude coordinates or distance

```
NULL
```

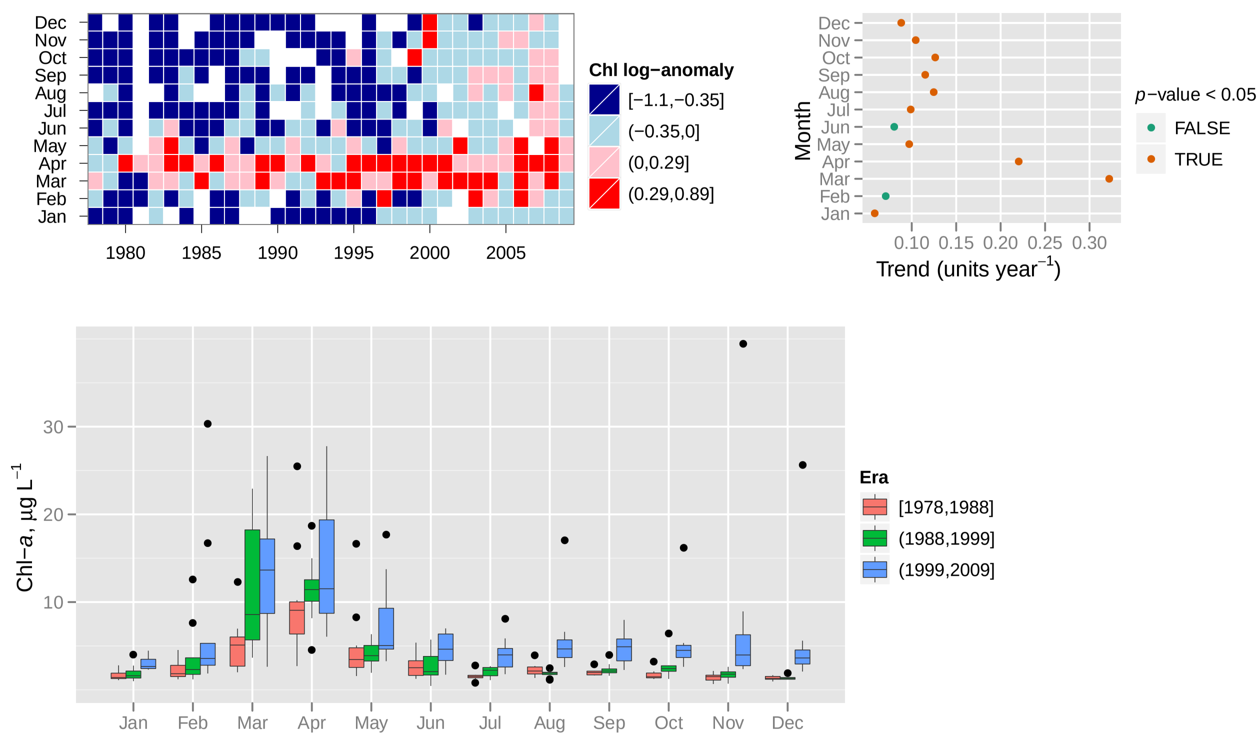

Figure 15: Example of use of `layOut` function for the chlorophyll-a monthly time series at station 27 in San Francisco Bay.

along the axis of a channel from some fixed point. Observed values may be numbers, numeric ranges or discrete classifications. Ideally, one wants each of the basic four kinds of information to accommodate all forms in common use.

An obvious extension of the `WqData` object would be to include additional slots for site and variable metadata, so that there is no ambiguity about the availability of this information. A more significant change would be to define classes for the fields described above as superclasses of basic classes, as already done for time (the `"DateTime"` class). For example, a `site` class could accommodate factors, numeric vectors or matrices. Location could then be given by discrete site name, x position as distance from a fixed point, or x and y positions as latitude and longitude. Similarly, depth could accommodate factors or numeric vectors, the former as names of depth layers ("top 5 m") or as non-numeric depths ("just below surface" or "bottom"). Currently, non-numeric depths can be coded as negative numbers.

Ultimately, the package direction will be driven by the needs of people actually using it. Suggestions for revisions and additions are therefore welcome.

References

Cloern, J. E., and A. D. Jassby. 2010. Patterns and scales of phytoplankton variability in estuarine-coastal ecosystems. Estuaries and Coasts **33**: 230–241.

Grothendieck, G., and T. Petzoldt. 2004. R help desk: Date and time classes in R. R News **4**: 29–32.

Hannachi, A., I. T. Jolliffe, and D. B. Stephenson. 2007. Empirical orthogonal functions and related techniques in atmospheric science: A review. International Journal of Climatology **27**: 1119–1152.

Helsel, D., and L. Frans. 2006. Regional Kendall test for trend. Environmental Science and Technology **40**: 4066–4073.

Helsel, D., and R. Hirsch. 2002. Statistical methods in water resources, Techniques of Water-Resources Investigations of the United States Geological Survey. Book 4, Hydrologic Analysis and Interpretation. Chapter A3. U.S. Geological Survey.

Hirsch, R., J. Slack, and R. Smith. 1982. Techniques of trend analysis for monthly water quality data. Water Resources Research **18**: 107–121.

Jassby, A. D. 1999. Uncovering mechanisms of interannual variability in ecological time series, pp. 285–306. *In* K. Scow, G. Fogg, D. Hinton, and M. Johnson [eds.], Integrated assessment of ecosystem health. CRC Press.

Schertz, T. L., R. B. Alexander, and D. J. Ohe. 1991. The computer program EStimate TREND (ESTREND), a system for the detection of trends in water-quality data. Water-Resources Investigations Report 91-4040, U.S. Geological Survey.

van Belle, G., and J. P. Hughes. 1984. Nonparametric tests for trend in water quality. Water Resources Research **20**: 127–136.