

SWMPPr: An R package for the National Estuarine Research Reserve System

Marcus W. Beck, beck.marcus@epa.gov

Oct. 22, 2014

Contents

Overview	1
Installing the package	2
Data retrieval	2
The swmpr object class	4
An overview of methods for swmpr objects	5
Functions	12
Retrieve	12
Organize	12
Analyze	12
Miscellaneous	13

Overview

SWMPPr is an R package that contains functions for retrieving, organizing, and analyzing estuary monitoring data from the System Wide Monitoring Program (SWMP). SWMP was implemented by the National Estuarine Research Reserve System (NERRS) in 1995 to provide continuous monitoring data at over 300 stations in 28 estuaries across the United States. SWMP data are maintained online by the Centralized Data Management Office (CDMO). This R package provides several functions to retrieve, organize, and analyze SWMP data from the CDMO. Information on the CDMO web services are available [here](http://www.omegahat.org/SSOAP/). Your computer's IP address must be registered with the CDMO website to use most of the data retrieval functions, see contact info in the link. All other functions can be used after obtaining data from the CDMO, as described below.

The package has many dependencies, the most important being the SSOAP package for retrieving data from the CDMO using a SOAP client interface. All dependencies are handled automatically when SWMPPr is installed and loaded, excluding SSOAP. The SSOAP package is not required to use SWMPPr but is necessary for using most of the data retrieval functions. The SSOAP package is currently removed from CRAN but accessible at <http://www.omegahat.org/SSOAP/>. Functions that require SSOAP should install the package automatically or it can be installed as follows:

```
install.packages("SSOAP", repos="http://www.omegahat.org/R", dependencies = T,  
                  type = "source")
```

All data obtained from the CDMO should be cited using the format:

National Estuarine Research Reserve System (NERRS). 2012. System-wide Monitoring Program. Data accessed from the NOAA NERRS Centralized Data Management Office website: <http://cdmo.baruch.sc.edu/>; accessed 12 October 2012.

To cite this package:

Beck MW. 2014. SWMPPr: An R package for the National Estuarine Research Reserve System. Version 0.3.2. <https://github.com/fawda123/SWMPPr>

Installing the package

This package is currently under development and has not been extensively tested. The development version can be installed from Github:

```
install.packages('devtools')
require(devtools)
install_github('fawda123/SWMPPr')
require(SWMPPr)
```

Note that the current version of devtools (v1.6.1) was built under R version 3.1.1. The SWMPPr package may not install correctly with older versions of R.

Data retrieval

SWMP data can be used in R after they are obtained directly from the CDMO through an online query or by using the retrieval functions provided in this package. In the latter case, the IP address for the computer making the request must be registered with CDMO. This can be done by following instructions here. The metadata should also be consulted for available data, including the parameters and date ranges for each monitoring station. Metadata are included as a .csv file with data requested from the CDMO and can also be obtained using the `site_codes` (all sites) or `site_codes_ind` (individual site) functions.

```
# retrieve metadata for all sites
site_codes()

# retrieve metadata for a single site
site_codes_ind('apa')
```

Due to rate limitations on the server, the retrieval functions in this package return a limited number of records. The functions are more useful for evaluating short time periods, although these functions could be used iteratively (i.e., with `for` or `while` loops) to obtain longer time series. Data retrieval functions to access the CDMO include `all_params`, `all_params_dtrng`, and `single_param`. These are functions that call the available methods on the CDMO SOAP interface. `all_params` returns the most recent 100 records of all parameters at a station, `all_params_dtrng` returns all records within a date range for all parameters or a single parameter, and `single_param` is identical to `all_params` except that a single parameter is requested.

```
# all parameters for a station, most recent
all_params('hudscwq')

# get all parameters within a date range
```

```
all_params_dtrng('hudscwq', c('09/10/2012', '02/8/2013'))

# get single parameter within a date range
all_params_dtrng('hudscwq', c('09/10/2012', '02/8/2013'), param = 'do_mgl')

# single parameter for a station, most recent
single_param('hudscwq', 'do_mgl')
```

For larger requests, it's easier to obtain data outside of R using the CDMO query system and then importing within R using the `import_local` function. Data can be retrieved from the CDMO several ways. The `import_local` function is designed for data from the zip downloads feature in the advanced query section of the CDMO. The function may also work using data from the data export system, but this feature has not been extensively tested (expect bugs). The zip downloads feature is an easy way to obtain data from multiple stations in one request. The downloaded data will be in a compressed folder that includes multiple .csv files by year for a given data type (e.g., apacpwq2002.csv, apacpwq2003.csv, apacpnut2002.csv, etc.). It is recommended that all stations at a site and the complete date ranges are requested to avoid repeated requests to CDMO. The `import_local` function can be used after the folder is decompressed.

```
# import data for apaebmet that you downloaded

# this is an example path with the csv files, change as needed
path <- 'C:/my_path/'

# import, do not include file extension
import_local(path, 'apaebmet')
```

For the following example, the `import_local` function is used to load data included with the SWMPPr distribution. To use it, set the path variable using the `system.file` command shown below (you can see this full path by executing `path` at the command line). Execute both lines to import the data.

```
# import data for apaebmet that comes with SWMPPr

# this is the path for csv example files
path <- system.file('zip_ex', package = 'SWMPPr')

# import, do not include file extension
import_local(path, 'apaebmet')
```

In all cases, the imported data need to assigned to an object in the workspace for use with other functions:

```
# import data and assign to dat
dat <- import_local(path, 'apaebmet', trace = F)

# view first six rows
head(dat)
```

```
##           datetimestamp atemp f_atemp rh f_rh    bp f_bp wspd f_wspd maxwspd
## 1 2011-01-01 00:00:00  15.4    <0>  94 <0> 1019 <0>   2.6    <0>    3.4
## 2 2011-01-01 00:15:00  15.2    <0>  95 <0> 1019 <0>   2.7    <0>    4.0
## 3 2011-01-01 00:30:00  15.2    <0>  95 <0> 1019 <0>   2.8    <0>    3.5
## 4 2011-01-01 00:45:00  15.3    <0>  95 <0> 1019 <0>   3.1    <0>    4.2
## 5 2011-01-01 01:00:00  15.3    <0>  95 <0> 1018 <0>   3.2    <0>    4.4
```

```
## 6 2011-01-01 01:15:00 15.3 <0> 95 <0> 1018 <0> 3.6 <0> 4.9
## f_maxwspd wdir f_wdir sdwdir f_sdwdir totpar f_totpar totprcp f_totprcp
## 1 <0> 145 <0> 8 <0> 0.8 <1> (CSM) 0 <0>
## 2 <0> 146 <0> 7 <0> 0.8 <1> (CSM) 0 <0>
## 3 <0> 139 <0> 7 <0> 0.8 <1> (CSM) 0 <0>
## 4 <0> 140 <0> 7 <0> 0.8 <1> (CSM) 0 <0>
## 5 <0> 144 <0> 6 <0> 0.8 <1> (CSM) 0 <0>
## 6 <0> 141 <0> 7 <0> 0.8 <1> (CSM) 0 <0>
## cumprcp f_cumprcp totsorad f_totsorad
## 1 0 <0> NA <-1>
## 2 0 <0> NA <-1>
## 3 0 <0> NA <-1>
## 4 0 <0> NA <-1>
## 5 0 <0> NA <-1>
## 6 0 <0> NA <-1>
```

The swmpr object class

All data retrieval functions return a swmpr object that includes relevant data and several attributes describing the dataset. The data include a datetimestamp column in the appropriate timezone for a station. Note that the datetimestamp is standard time for each timezone and does not include daylight savings. Additional columns include parameters for a given data type (weather, nutrients, or water quality) and corresponding QAQC columns if returned from the initial data request. The attributes for a swmpr object include **names** of the dataset, **class** (swmpr) **station name** (7 or 8 characters), **qaqc_cols** (logical), **date_rng** (POSIX vector), **timezone** (text string in country/city format), **stamp_class** (class of datetimestamp vector, POSIX or Date), and **parameters** (character vector). Attributes of a swmpr object can be viewed as follows:

```
# verify that dat is swmpr class
class(dat)
```

```
## [1] "swmpr" "data.frame"
```

```
# all attributes of dat
names(attributes(dat))
```

```
## [1] "names" "row.names" "class" "station" "parameters"
## [6] "qaqc_cols" "date_rng" "timezone" "stamp_class"
```

```
# a single attribute of dat
attr(dat, 'station')
```

```
## [1] "apaebmet"
```

The swmpr object class was created for use with specific methods and it is suggested that these methods be used for data organization and analysis. A swmpr object also secondarily inherits methods from the data.frame class, such that common data.frame methods also apply to swmpr objects. Available methods for the swmpr class are described below and can also be viewed:

```
# available methods for swmpr class
methods(class = 'swmpr')
```

```
## [1] aggregate.swmpr comb.swmpr      decomp.swmpr      hist.swmpr
## [5] lines.swmpr      na.approx.swmpr  plot.swmpr        qaqc.swmpr
## [9] setstep.swmpr    smoother.swmpr  subset.swmpr
```

An overview of methods for swmpr objects

Three categories of functions are available: retrieve, organize, and analyze. The retrieval functions import the data into R as a swmpr object for use with the organize and analyze functions. Methods defined for swmpr objects can be applied with the organize and analyze functions. These methods are available for generic functions specific to this package, in addition to methods for existing generic functions available from other packages. S3 methods are implemented in all cases.

The organize functions are used to clean or prepare the data for analysis, including removal of QAQC flags, subsetting, creating a standardized time series vector, and combining data of different types. The `qaqc` function is a simple screen to retain values from the data with specified QAQC flags, described here. Each parameter in the swmpr data typically has a corresponding QAQC column of the same name with the added prefix 'f_'. Values in the QAQC column specify a flag from -5 to 5. Generally, only data with the '0' QAQC flag should be used, which is the default option for the `qaqc` function. Data that do not satisfy QAQC criteria are converted to NA values. Processed data will have QAQC columns removed, in addition to removal of values in the actual parameter columns that do not meet the criteria.

```
# qaqc screen for a swmpr object, retain only '0'
qaqc(dat)

# retain all data regardless of flag
qaqc(dat, qaqc_keep = NULL)

# retain only '0' and '-1' flags
qaqc(dat, qaqc_keep = c(0, -1))
```

A subset method added to the existing `subset` function is available for swmpr objects. This function is used to subset the data by date and/or a selected parameter. The date can be a single value or as two dates to select records within the range. The former case requires a binary operator input as a character string passed to the argument, such as `>` or `<`. The subset argument for the date(s) must also be a character string of the format YYYY-mm-dd HH:MM for each element (i.e., %Y-%m-%d %H:%M in POSIX standards). Finally, the function can be used to remove rows and columns that do not contain data.

```
# select two parameters from dat
subset(dat, select = c('rh', 'bp'))

# subset records greater than or equal to a date
subset(dat, subset = '2013-01-01 0:00', operator = '>=')

# subset records within a date range
subset(dat, subset = c('2012-07-01 6:00', '2012-08-01 18:15'))

# subset records within a date range, select two parameters
subset(dat, subset = c('2012-07-01 6:00', '2012-08-01 18:15'),
       select = c('atemp', 'totsorad'))

# remove rows/columns that do not contain data
subset(dat, rem_rows = T, rem_cols = T)
```

The `setstep` function formats a `swmpr` object to a continuous time series at a given time step. This function is not necessary for most stations but can be useful for combining data or converting an existing time series to a set interval. The first argument, `timestep` specifies the desired time step in minutes starting from the nearest hour of the first observation. The second argument, `differ`, specifies the allowable tolerance in minutes for matching existing observations to user-defined time steps in cases where the two are dissimilar. Values for `differ` that are greater than one half the value of `timestep` are not allowed to prevent duplication of existing data. Likewise, the default value for `differ` is one half the time step. Rows that do not match any existing data within the limits of the `differ` argument are not discarded. Output from the `setstep` function can be used with `subset` and to create a time series at a set interval with empty data removed.

```
# convert time series to two hour intervals
# tolerance of +/- 30 minutes for matching existing data
setstep(dat, timestep = 120, differ = 30)

# convert a nutrient time series to a continuous time series
# then remove empty rows and columns
dat_nut <- import_local(path, 'apacpnut')
dat_nut <- setstep(dat_nut, timestep = 60)
subset(dat_nut, rem_rows = T, rem_cols = T)
```

The `comb` function is used to combine multiple `swmpr` objects into a single object with a continuous time series at a given step. The `timestep` function is used internally such that `timestep` and `differ` are accepted arguments for `comb`. The function requires one or more `swmpr` objects as input as separate, undefined arguments. The remaining arguments must be called explicitly since an arbitrary number of objects can be used as input. In general, the function combines data by creating a master time series that is used to iteratively merge all `swmpr` objects. The time series for merging depends on the value passed to the `method` argument. Passing `union` to `method` will create a time series that is continuous starting from the earliest date and the latest date for all input objects. Passing `intersect` to `method` will create a time series that is continuous from the set of dates that are shared between all input objects. Finally, a seven or eight character station name passed to `method` will merge all input objects based on a continuous time series for the given station. The specified station must be present in the input data. Currently, combining data types from different stations is not possible, excluding weather data which are typically at a single, dedicated station.

```
# get nuts, wq, and met data as separate objects for the same station
# note that most sites usually have one weather station
swmp1 <- import_local(path, 'apacpnut')
swmp2 <- import_local(path, 'apacpwq')
swmp3 <- import_local(path, 'apaebmet')

# combine nuts and wq data by union
comb(swmp1, swmp2, method = 'union')

# combine nuts and wq data by intersect
comb(swmp1, swmp3, method = 'intersect')

# combine nuts, wq, and met data by nuts time series, two hour time step
comb(swmp1, swmp2, swmp3, timestep = 120, method = 'apacpnut')
```

The analysis functions range from general purpose tools for time series analysis to more specific functions for working with continuous monitoring data in estuaries. The latter category includes a limited number of functions that were developed by myself or others. The general purpose tools are `swmpr` methods that were developed for existing generic functions in the R base installation or relevant packages. These functions include `swmpr` methods for `aggregate`, `filter`, and `approx` to deal with missing or noisy data and more

general functions for exploratory data analysis, such as `plot`, `summary`, and `hist` methods. A `decomp` function is provided for a relatively simple approach for time series decomposition. The analysis functions may or may not return a `swmpr` object depending on whether further processing with `swmpr` methods is possible from the output.

The `aggregate` function aggregates parameter data for a `swmpr` object by set periods of observation. This function is most useful for aggregating noisy data to evaluate trends on longer time scales, or to simply reduce the size of a dataset. Data can be aggregated by years, quarters, months, weeks, days, or hours for a user-defined function, which defaults to the mean. A `swmpr` object is returned for the aggregated data, although the `datetimestamp` vector will be converted to a `date` object if the aggregation period is a day or longer. Days are assigned to the `date` vector if the aggregation period is a week or longer based on the `round` method for `IDate` objects (`data.table` package). This approach was used to facilitate plotting using predefined methods for `Date` and `POSIX` objects. Additionally, the method of treating NA values for the aggregation function should be noted since this may greatly affect the quantity of data that are returned (see the example below). Finally, the default argument for `na.action` is set to `na.pass` for `swmpr` objects to preserve the time series of the input data.

```
# combine, qaqc, remove empty columns
dat <- comb(swmpr1, swmpr2, method = 'union')
dat <- qaqc(dat)
swmpr_in <- subset(dat, rem_cols = T)

# get mean DO by quarters
aggregate(swmpr_in, 'quarters', params = c('do_mgl'))

# get mean DO by quarters, remove NA when calculating means
fun_in <- function(x) mean(x, na.rm = T)
aggregate(swmpr_in, FUN = fun_in, 'quarters', params = c('do_mgl'))
```

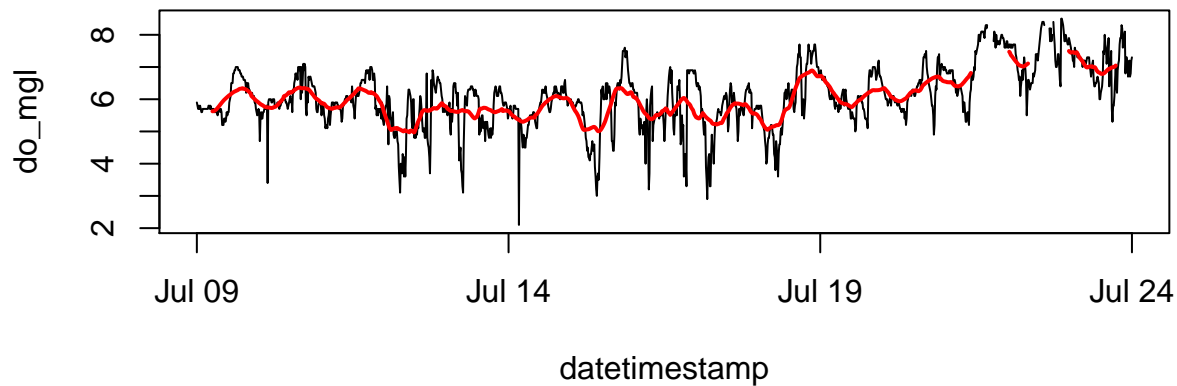
Time series can be smoothed to better characterize a signal independent of noise. Although there are many approaches to smoothing, a moving window average is intuitive and commonly used. The `smoother` function can be used to smooth parameters in a `swmpr` object using a specified window size. This method is a simple wrapper to `filter`. The `window` argument specifies the number of observations included in the moving average. The `sides` argument specifies how the average is calculated for each observation (see the documentation for `filter`). A value of 1 will filter observations within the window that are previous to the current observation, whereas a value of 2 will filter all observations within the window centered at zero lag from the current observation. As before, the `params` argument specifies which parameters to smooth.

```
# import data
swmpr1 <- import_local(path, 'apadbwq')

# qaqc and subset imported data
dat <- qaqc(swmpr1)
dat <- subset(dat, subset = c('2012-07-09 00:00', '2012-07-24 00:00'))

# filter
test <- smoother(dat, window = 50, params = 'do_mgl')

# plot to see the difference
plot(do_mgl ~ datetimestamp, data = dat, type = 'l')
lines(test, select = 'do_mgl', col = 'red', lwd = 2)
```



A common issue with any statistical analysis is the treatment of missing values. Missing data can be excluded from the analysis, included but treated as true zeroes, or interpolated based on similar values. In either case, an analyst should have a strong rationale for the chosen method. A common approach used to handle missing data in time series analysis is linear interpolation. A simple curve fitting method is used to create a continuous set of records between observations separated by missing data. A challenge with linear interpolation is an appropriate gap size for fitting missing observations. The ability of the interpolated data to approximate actual trends is a function of the gap size. Interpolation between larger gaps are less likely to resemble patterns of an actual parameter, whereas interpolation between smaller gaps are more likely to resemble actual patterns. An appropriate gap size limit depends on the unique characteristics of specific datasets or parameters. The `na.approx` function can be used to interpolate gaps in a `swmpr` object. A required argument for the function is `maxgap` which defines the maximum gap size for interpolation.

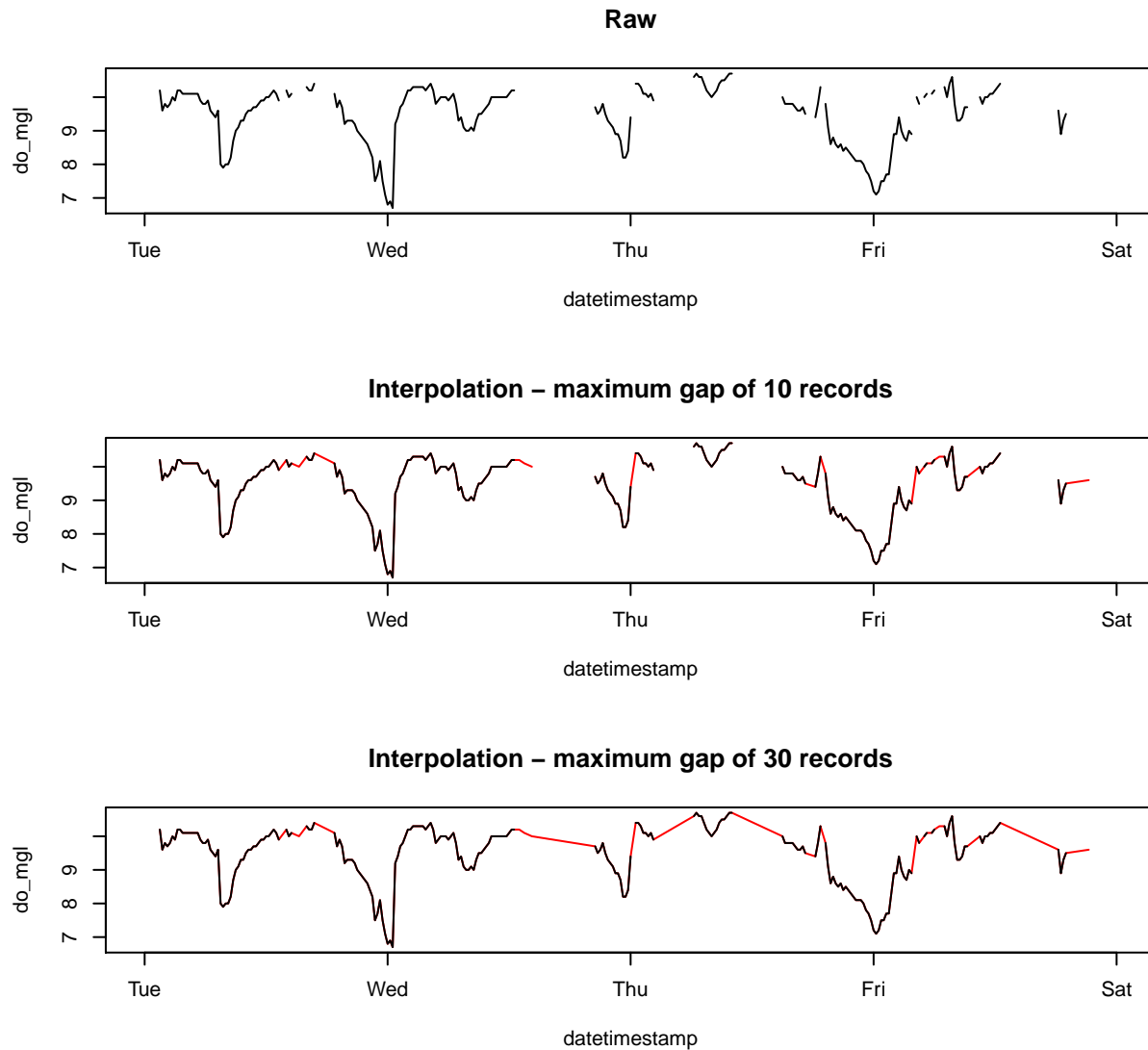
```
# get data
swmp1 <- import_local(path, 'apadbwq')

# qaqc and subset imported data
dat <- qaqc(swmp1)
dat <- subset(dat, subset = c('2013-01-22 00:00', '2013-01-26 00:00'))

# interpolate, maxgap of 10 records
test <- na.approx(dat, params = 'do_mgl', maxgap = 10)

# interpolate maxgap of 30 records
test2 <- na.approx(dat, params = 'do_mgl', maxgap = 30)

# plot for comparison
par(mfrow = c(3, 1))
plot(do_mgl ~ datetimestamp, dat, main = 'Raw', type = 'l')
plot(do_mgl ~ datetimestamp, test, col = 'red',
     main = 'Interpolation - maximum gap of 10 records', type = 'l')
lines(dat, select = 'do_mgl')
plot(do_mgl ~ datetimestamp, test2, col = 'red',
     main = 'Interpolation - maximum gap of 30 records', type = 'l')
lines(dat, select = 'do_mgl')
```

The **decomp** function is a simple wrapper to **decompose** that separates a time series into additive or multiplicative components describing a trend, cyclical variation (e.g., daily or seasonal), and the remainder. The additive decomposition assumes that the cyclical component of the time series is stationary (i.e., the variance is constant), whereas a multiplicative decomposition accounts for non-stationarity. By default, a moving average with a symmetric window is used to filter the seasonal component. Alternatively, a vector of filter coefficients in reverse time order can be supplied (see the help documentation for **decompose**).

The **decompose** function requires a ts object with a specified frequency as input. The **decomp** function converts the input swmpr vector to a ts object prior to **decompose**. This requires an explicit input defining the frequency of the parameter in the time series. For example, the frequency of a parameter with diurnal periodicity would be 96 if the time step is 15 minutes ($4 * 24$). The frequency of a parameter with seasonal periodicity would be 35040 ($4 * 24 * 365$). For simplicity, character strings of 'daily' or 'seasonal' can be supplied in place of numeric values. A starting value of the time series must be supplied in the latter case. Use of the **setstep** function is also required to standardize the time step prior to decomposition.

Note that the **decompose** function is a relatively simple approach and alternative methods should be investigated if a more sophisticated decomposition is desired.

```

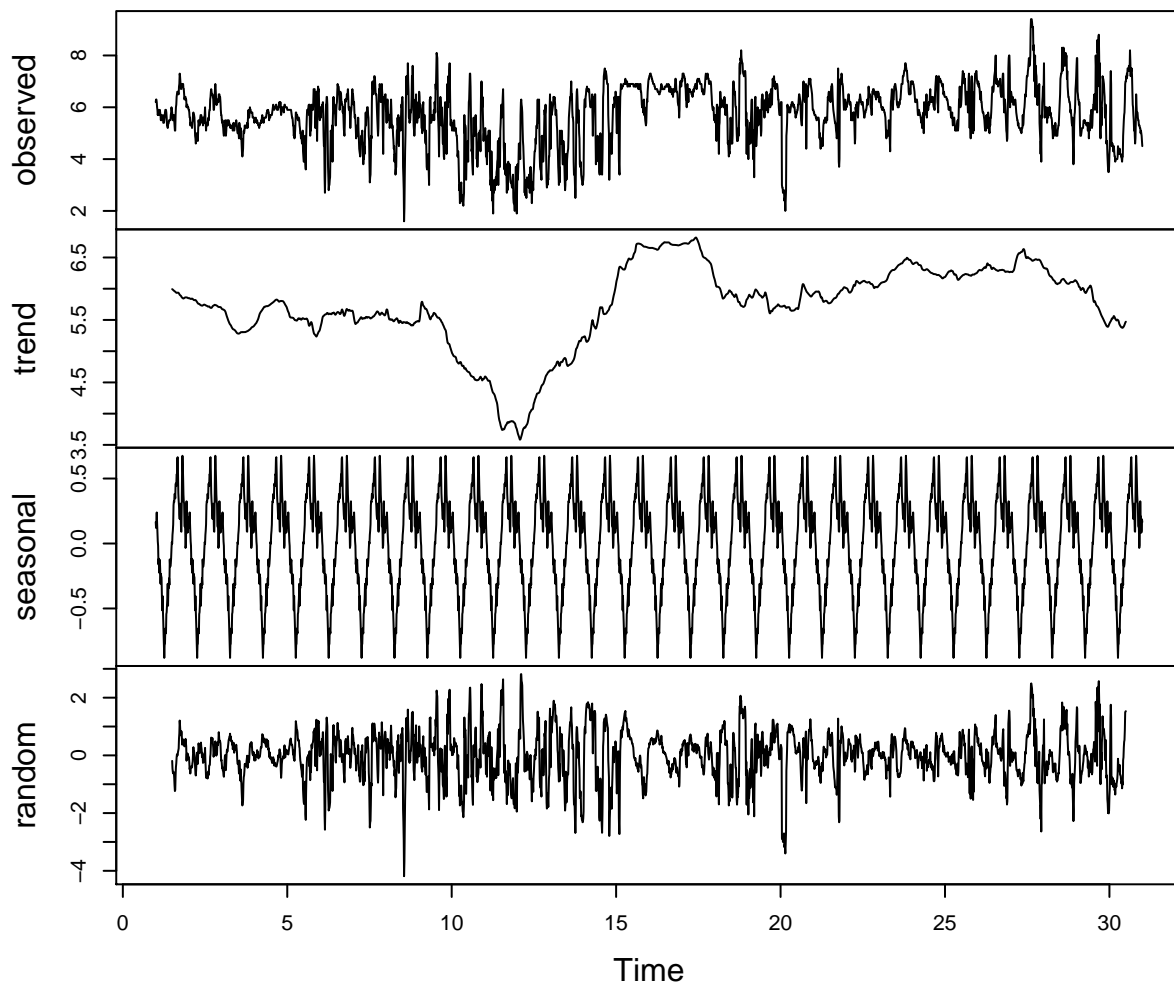
# get data
swmp1 <- import_local(path, 'apadbwq')

# subset for daily decomposition
dat <- subset(swmp1, subset = c('2013-07-01 00:00', '2013-07-31 00:00'))

# decomposition and plot
test <- decomp(dat, param = 'do_mgl', frequency = 'daily')
plot(test)

```

Decomposition of additive time series



The next example illustrates how to handle missing values using the `decomp` function. The `decompose` function used internally within `decomp` currently cannot process time series with missing values. A recommended approach is to use `na.approx` to interpolate the missing values prior to `decompose`.

```

# get data
dat <- subset(swmp1, subset = c('2013-06-01 00:00', '2013-07-31 00:00'))

```

```
# this returns an error
# test <- decomp(dat, param = 'do_mgl', frequency = 'daily')

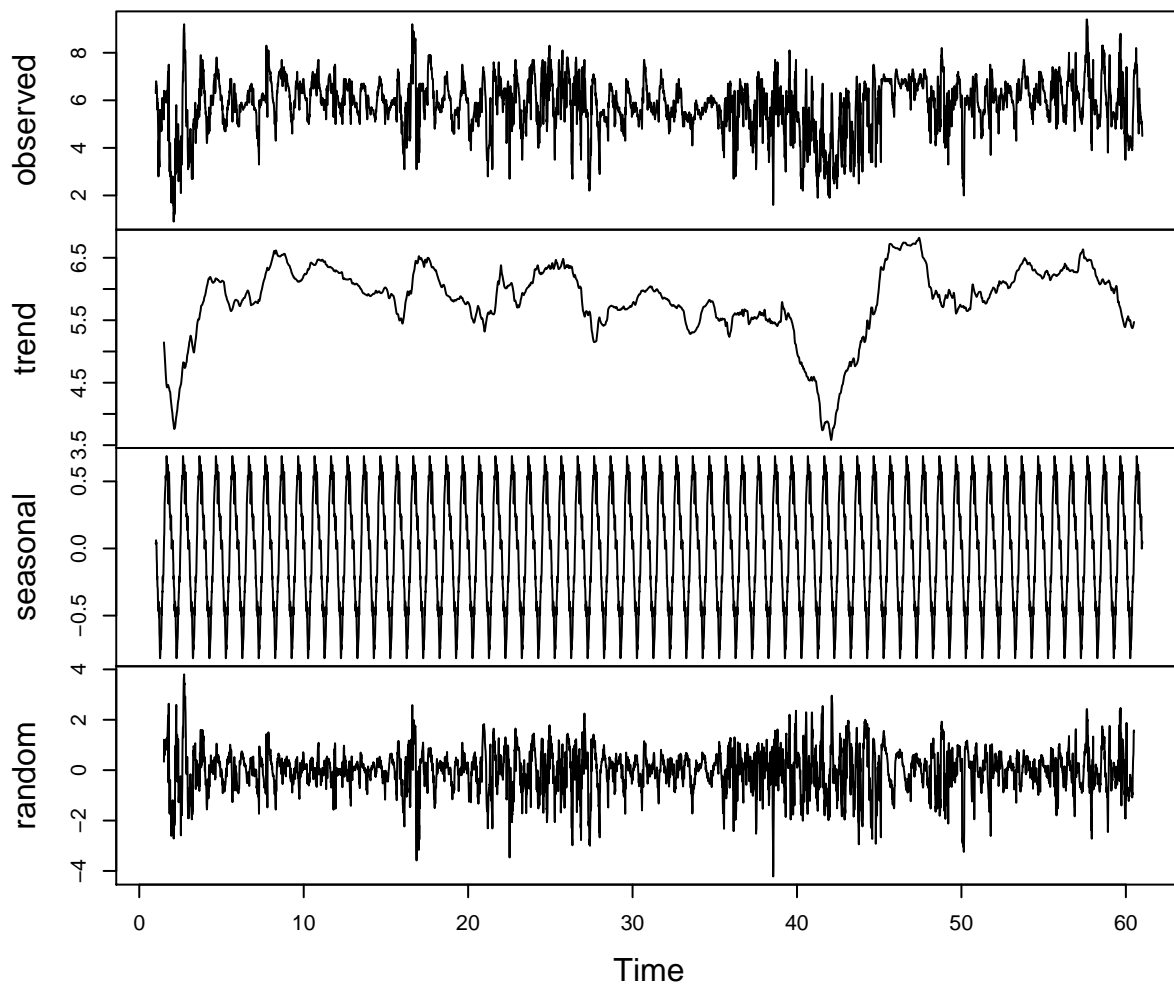
# how many missing values?
sum(is.na(dat$do_mgl))
```

```
## [1] 3
```

```
# use na.approx to interpolate missing data
dat <- na.approx(dat, params = 'do_mgl', maxgap = 10)

# decomposition and plot
test <- decomp(dat, param = 'do_mgl', frequency = 'daily')
plot(test)
```

Decomposition of additive time series



Functions

See help documentation for more details on each function (e.g., `?all_param`).

Retrieve

all_params Retrieve up to 100 records starting with the most recent at a given station, all parameters. Wrapper to **exportAllParamsXMLNew** function on web services.

all_params_dtrng Retrieve records of all parameters within a given date range for a station. Optional argument for a single parameter. Maximum of 1000 records. Wrapper to **exportAllParamsDateRangeXMLNew**.

single_param Retrieve up to 100 records for a single parameter starting with the most recent at a given station. Wrapper to **exportSingleParamXMLNew** function on web services.

import_local Import files from a local path. The files must be in a specific format, specifically those returned from the CDMO using the zip downloads option for a reserve.

Organize

qaqc.swmpr Remove QAQC columns and remove data based on QAQC flag values for a swmpr object. Only applies if QAQC columns are present.

subset.swmpr Subset by dates and/or columns for a swmpr object. This is a method passed to the generic 'subset' function provided in the base package.

setstep.swmpr Format data from a swmpr object to a continuous time series at a given timestep. The function is used in **comb.swmpr** and can also be used with individual stations.

comb.swmpr Combines swmpr objects to a common time series using setstep, such as combining the weather, nutrients, and water quality data for a single station. Only different data types can be combined.

Analyze

aggregate.swmpr Aggregate swmpr objects for different time periods - years, quarters, months, weeks, days, or hours. Aggregation function is user-supplied but defaults to mean.

smoother.swmpr Smooth swmpr objects with a moving window average. Window size and sides can be specified, passed to **filter**.

na.approx.swmpr Linearly interpolate missing data (NA values) in a swmpr object. The maximum gap size that is interpolated is defined as a maximum number of records with missing data.

plot.swmpr Plot a univariate time series for a swmpr object. The parameter name must be specified.

lines.swmpr Add lines to an existing swmpr plot.

hist.swmpr Plot a histogram for a swmpr object.

decomp.swmpr Decompose a swmpr time series into trend, seasonal, and residual components. This is a simple wrapper to **decompose**.

Miscellaneous

swmpr Creates object of `swmpr` class, used internally in retrieval functions.

parser Parses html returned from CDMO web services, used internally in retrieval functions.

time_vec Converts time vectors to POSIX objects with correct time zone for a site/station, used internally in retrieval functions.

site_codes Metadata for all stations, wrapper to `exportStationCodesXMLNew` function on web services.

site_codes_ind Metadata for all stations at a single site, wrapper to `NERRFilterStationCodesXMLNew` function on web services.

param_names Returns column names as a list for the parameter type(s) (nutrients, weather, or water quality). Includes QAQC columns with 'f_' prefix. Used internally in other functions.