Exploring DataForEver data using the SFNRC R package

1. Introduction

The `SFNRC` R package provides a simple, streamlined means of interacting with the DataForEver hydrology and water quality databases from within an analytical programming environment (RStudio). Additional tools are also included in the package, including:

- APIs for the South Florida Water Management District's DBHYDRO database
- A function to process data collected by the Miami-Dade Department of Environmental and Resources Management in preparation for merging it with DataForEver data
- A responsive tool for spatial interpolation of data
- Utility functions to calculate annual geometric means, label seasons and water years, and similar rudimentary tasks.

This vignette demonstrates some of the functionality of the `SFNRC` R package.

Before demonstrating the package's capabilities, it is important to highlight some limitations of the `SFNRC` R package. The DataForEver APIs can only be used on Linux machines connected to the SFNRC's internal network. When DataForEver's data are opened up to the general public, the capabilities in this R package will be adapted to provide broader access to these world-class databases. Another important aspect to draw attention to is the construction and use of BASH scripts by the DataForEver API functions. This dependence on BASH is the reason the APIs only work on Linux operating systems.

---

2. Automated access to DataForEver

Users can search for stations in the DataForEver databases:

```
getStn(query = "s33")
```

Users can also query the parameters available for each station in the DataForEver hydrology database, using parameter and/or station names as constraints:

```
getDataTypes(parameter = "salinity")

getDataTypes(stn = "S333")
```

```
## to search for exact matches, set `exactMatch = TRUE`
getDataTypes(stn = "S333", exactMatch = TRUE)
```

3. Downloading DataForEver data

Using `SFNRC` to download DataForEver data is very straightforward. The water quality database is accessed using the function `getWQ()`:

```
# identify desired stations and water quality parameters
stations   <- c("S333", "S12A")
wqParams   <- c("PHOSPH|NITROGEN|AMMONI|SUSPENDED|TURBIDITY")
```

```
dat <- getWQ(stns = stations, target_analytes = wqParams)
head(dat)
```

The `SFNRC` R package can also be used to download data from the DataForEver hydrology database using a function called `getHydro()`. The `data_shape` argument reshapes data from long to wide form to facilitate a broad range of end-user analyses.

```
hydroParams <- c("flow", "head_water")
```

```
### by default, data are in "long" format, with one row
### for every date-station-parameter combination
hyd.long  <- getHydro(stns = stations, parameter_list = hydroParams)

### but data can also be retrieved in "wide" (one row per date-station) or
### "really_wide" (one row per date) forms, enabling more rapid
### comparison between stations and/or parameters
hyd.wide  <- getHydro(stns = stations, parameter_list = hydroParams, data_shape = "wide")
hyd.vwide <- getHydro(stns = stations, parameter_list = hydroParams, data_shape = "really_wide")
```

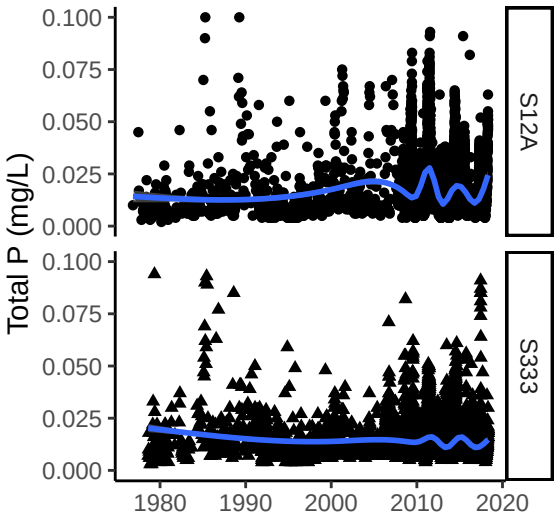
```
head(hyd.long) # one row for every date-station-parameter combination
head(hyd.wide) # one row for every date-station combination
head(hyd.vwide) # one row for every date
```

Merging water quality and hydrology data is a routine task that is automated by `SFNRC`. Instead of navigating the two databases and then figuring out how to merge the two datasets, water quality data can be downloaded at the same time as flow/hydrology data by setting `getWaterQuality = TRUE` in a call to `getHydro()`. To reduce the number of columns in the returned dataset, it is recommended that water quality parameters be specified manually in this workflow, because the returned dataset will have two columns per parameter (returning the MDL and the value).

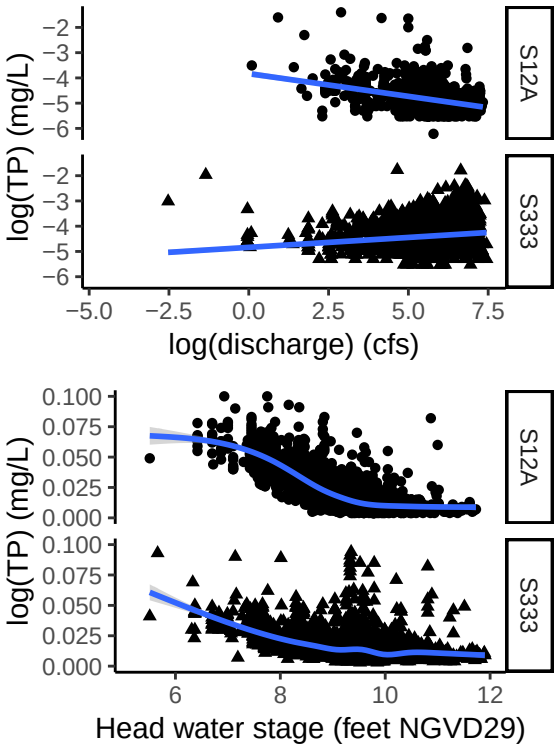
```
### simultaneously download hydrology and water quality data:
hyd.wq <- getHydro(stns = stations,
                   parameter_list = hydroParams, # hydrology database parameters
                   getWaterQuality = TRUE,
```

```
                target_analytes = wqParams) # water quality database parameters
```

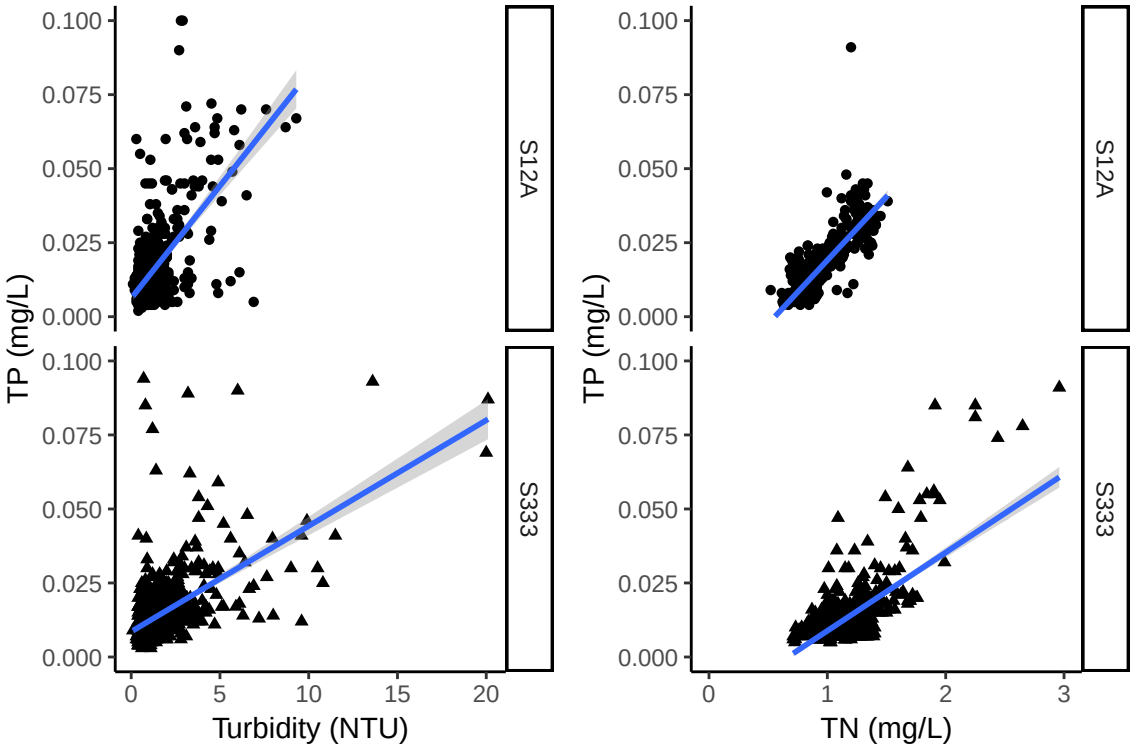
At this point the data can be plotted or analyzed. For example, simple time series plots:



Relationships between hydrologic and water quality parameters can be explored:



And relationships between water quality parameters can be evaluated:



4. Spatial interpolation wrapper

Spatial interpolation of data is streamlined by the `interp()` function. The function takes a user-specified SpatialPolygonDataframe and interpolates point data. The interpolation method applied to the data can be `ordinary kriging` or `nearest neighbor`. Ordinary kriging estimates data using a variogram model of similarity as a function of distance. By default, a broad range of models are attempted and the best-fitting model is used to produce the output. If `ordinary kriging` is selected but the variogram does not converge (typically indicative of substantial variation over small distances), the `nearest neighbor` approach will instead be used.

Nearest neighbor interpolation generates Voronoi polygons for the sampling locations and applies the value from each sampling point uniformly within its respective polygon.

An example of a complete workflow is below. Stations in Biscayne Bay are selected programatically, data undergo some initial processing and an interpolated map of salinity is produced.

```
# find stations in Biscayne Bay
bsc.stns <- getDataTypes(stn = "BISC|BB")
stns     <- as.character(unique(bsc.stns[bsc.stns$parameter %in% "salinity", "stn"]))

# get data
paramsToGet   <- c("salinity")
```

```
bsc           <- getHydro(stns = stns, parameter_list = paramsToGet)
names(bsc)[4] <- "SALINITY"
```

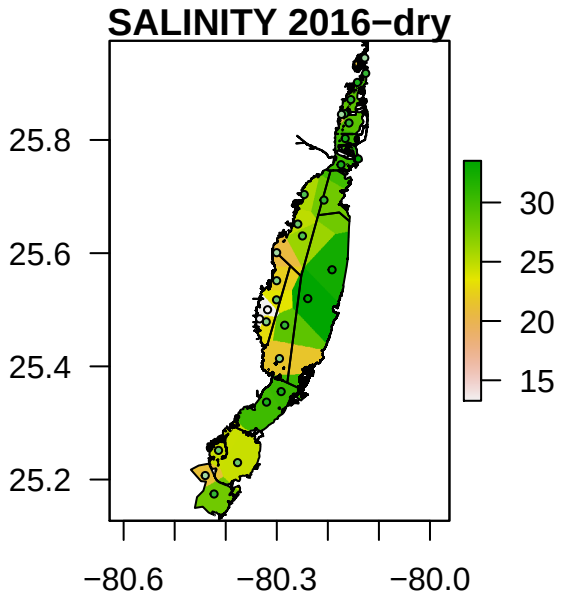
```
### assign wet/dry seasons
bsc        <- seas(bsc, wetSeas = c("May", "Sep"), waterYearBegin = "Oct", timeCol = "date")
bsc$seas2  <- paste0(bsc$waterYr,"-", bsc$seas)

### calculate annual geometric means
agm        <- ddply(bsc, .(stn, seas2), numcolwise(geoMean))

### add coordinates to make point data spatial
names(agm)    <- gsub(x = names(agm), pattern = " |,", replacement = "")
names(agm)    <- gsub(x = names(agm), pattern = "-|[+]", replacement = ".")
agmWithCoords <- join_all(list(agm, as.data.frame(masterCoords)), by = "stn")
sitesInBay    <- agmWithCoords[!is.na(agmWithCoords$long), ]
coordinates(sitesInBay) <- c("long", "lat")
proj4string(sitesInBay) <- CRS("+proj=longlat +datum=WGS84 +no_defs +ellps=WGS84 +towgs84=0,0,0")


# interpolate!
interp(inputData = sitesInBay, mapLayer = bnpMod,
       paramCol = "SALINITY", year = "2016-dry", yearCol = "seas2",
       interpMethod = "ordinary kriging")
```

```
30 stations with SALINITY data in 2016-dry
```



In the case above, ordinary kriging was unsuccessful because of a lack of convergence in the variogram, so the function used a nearest-neighbor approach to spatial interpolation.