Introduction to argoFloats

Dan Kelley (https://orcid.org/0000-0001-7808-5911), Jaimie Harbin (https://orcid.org/0000-0003-3774-3

2020-07-02

Abstract. The `argoFloats` package makes it easy to get, read, and analyze core, biogeochemical(bgc), and deep Argo data. Our goal in making this package was to eliminate the gap between the freely available Argo data and the end user.

Introduction

The `argoFloats` package provides tools for downloading and processing Argo profile data. It allows users to focus on core, biogeochemical (BGC) , or deep Argo profiles, and also allows the user to sift these profiles based on ID, time, geography, variable, and institution. Once downloaded, such data sets can be analysed within `argoFloats` or using other R tools and packages.

Youtube Videos

As an adjunct to the written documentation, the following videos are provided, to introduce concepts and show how to accomplish some every-day tasks. In some cases, sample code is also made available at https://github.com/ArgoCanada/argoFloats/tree/develop/videos.

Video Name	Creators	Date	URL
argoFloats R 01: Introduction	Dan Kelley & Jaimie Harbin	April 9, 2020	https://youtu.be/xeBoFbb66Nk
argoFloats R 02: TS plot near Bermuda	Jaimie Harbin & Dan Kelley	April 24, 2020	https://youtu.be/ZoTrVEMG5Qo
argoFloats R 03: new website	Jaimie Harbin & Dan Kelley	April 30, 2020	https://youtu.be/lOvCrRDTmTs
argoFloats R 04: subset by ocean or polygon	Jaimie Harbin & Dan Kelley	May 7, 2020	https://youtu.be/tcGRB479Udk
argoFloats R 05: TS diagram, colour-coded by oxygen	Jaimie Harbin & Dan Kelley	May 14, 2020	https://youtu.be/Y_SxjcOnW04
argoFloats R 06: trajectory plot, colour coded by time	Jaimie Harbin & Dan Kelley	May 28, 2020	https://youtu.be/7BB3UuwjUqo

Preliminary Setup

To set up a computer to use `argoFloats`, the user should first install the following packages (*Hint:* you may have to use the `Sys.setenv(R_REMOTES_NO_ERRORS_FROM_WARNINGS=TRUE)` command if you receive a Zero Exit Status Error):

```
devtools::install_github("ArgoCanada/oce", ref='develop')
devtools::install_github('ArgoCanada/ocedata', ref='develop')
devtools::install_github('ArgoCanada/argoFloats', ref='develop')
```

Note that the `install_github()` function is provided by the `devtools` package, so the above will fail if that package has not already been installed. To install it, use

```
install.packages(devtools)
```

Work Flow

Figure 1 illustrates the typical workflow with the package, with descriptions of the steps on the left, and names of the relevant functions on the right.

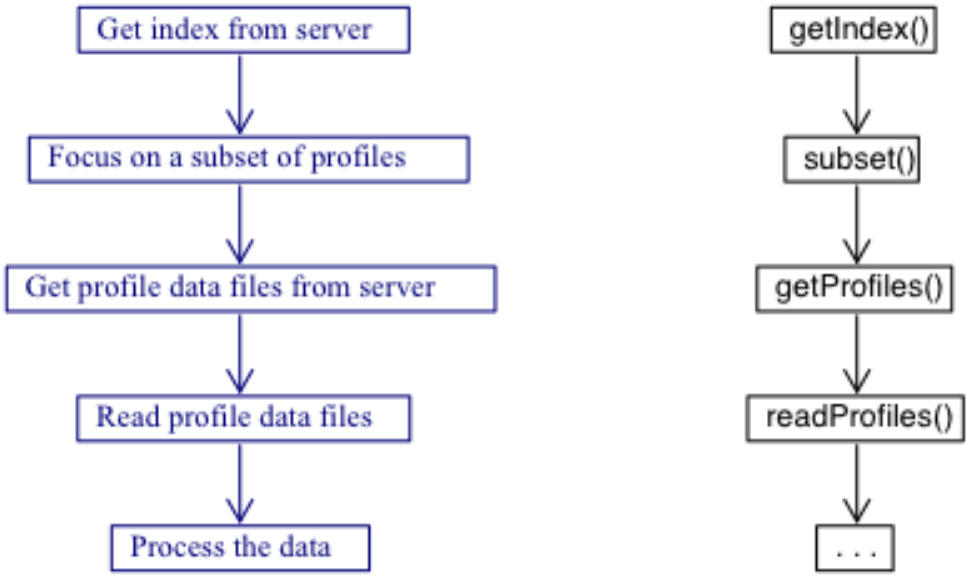

Figure 1: Figure 1. Flow chart

As shown above, the central functions for the argoFloats package are `getIndex()`, `subset()`, `getProfiles()`, and `readProfiles()`.

Some built-in data sets are provided for concreteness of illustration and for testing, but actual work always starts with a call to `getIndex()` to download a full index of float data, which we will demonstrate later in this vignette.

To begin to get familiar with how the `argoFloats` package works, we will begin looking at the built in data sets. Built into the `argoFloats` package is the `index`, `indexBgc`, and `indexMerged` indices, referring to core Argo, BGC-Argo, and a combination respectively. For the sake of this vignette we will focus on the `index` data set.

The first step is to access the required packages that will be needed during this tutorial, with

```
library(oce)
library(ocedata)
library(argoFloats)
```

To access the embedded index within `argoFloats`, the following code is used:

```
data('index')
```

It's now possible to process the downloaded index using the `argoFloats` specialized versions of R "generic" functions, `plot()`, `[[`, `summary()`, and `show()` as shown below.

Overview of Processing Steps

The following subsections use built-in data.

Plotting

The specialized `plot()` command within the `argoFloats` package provides simple ways to plot aspects of `argoFloats-class` objects. To produce the built in plot and visualize the coordinates of a section of Argo floats off of the Bahamas, the following code is used:

```
plot(index, bathymetry=FALSE)          # also, try using bathymetry=TRUE
```

Extracting Data

Furthermore, the `[[` command provides a way to extract items from `argoFloats` objects, without getting lost in the details of storage. For example, if the user wanted to extract the `file` within the `index` data set, instead of doing `index@data$index$file`, instead they can simply do `index[['file']]`. (Note that `[[<-` is *not* specialized, since the user is highly discouraged from altering values within `argoFloats` objects).

Summarizing

Additionally, the `summary()` command displays key features of `argoFloats-class` objects such as the type, server, file, URL etc. See `summary,argoFloats-method()` further details.

Printing

Lastly, the `show()` command provides a one-line sketch of `argoFloats-class` objects. This gets used by the `print()` function. For example if the user types in:

```
index
```

The following output occurs:

```
argoFloats object of type "index" with 953 items
```

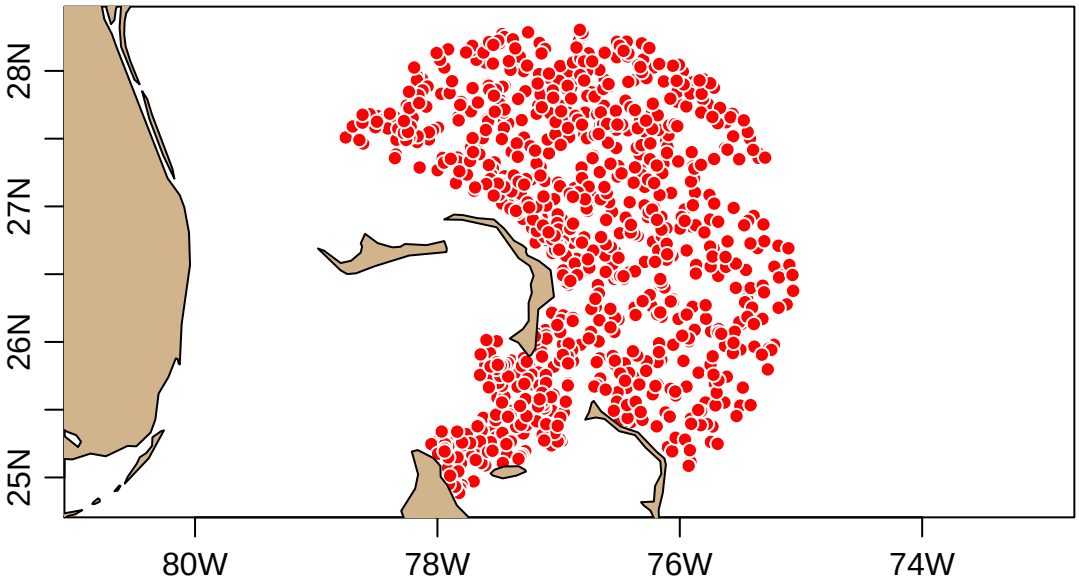

Figure 2: Figure 2: Built in index demonstrating argo floats within 200 km of Bahamas.

Hint: This command can be particularly useful when doing the `merge()` command, which will be explained in greater detail further down.

It should be noted that the profile elements within `argoFloats` objects are stored as in the form of `argo` objects as defined by the `oce` package. This means that `argoFloats` users can rely on a wide variety of `oce` functions to analyse their data. The full suite of R tools is also available, and the vastness of that suite explains why `argoFloats` is written in R.

Function Details

getIndex()

Until this point, we have demonstrated how the user can become familiar with embedded indices. As previously described, however, actual work always starts with downloading a full index of float data. As shown by Figure 1, the `getIndex()` command is used to get an index of available Argo float profiles, either by downloading information from a data repository or by reusing an index (stored as an .rda file) that was prepared by a recent call to the function.

The `getIndex()` command works by specifying the server, with first trying the USGODAE server ftp://usgodae.org/pub/outgoing/argo and then the IRREMER server ftp://ftp.ifremer.fr/ifremer/argo if that does not work. The next step is to specify the file name. The table below can be obtained using `?getIndex()`. As shown, the user has the ability to write the specific file name from the server, or to simply use the embedded nicknames within the package: `"argo"`, `"bgc"` or `"bgcargo"`, `"merge"` or `"merged"`, or `"synthetic"`. The following table summarizes the contents of the various files indicated by the `filename` argument.

File Name	Nickname	Contents
`ar_greylist.txt`	-	Suspicious/malfunctioning floats
`ar_index_global_meta.txt.gz`	-	Metadata files
`ar_index_global_prof.txt.gz`	`"argo"`	Argo data
`ar_index_global_tech.txt.gz`	-	Technical files
`ar_index_global_traj.txt.gz`	-	Trajectory files
`argo_bio-profile_index.txt.gz`	`"bgc"` or `"bgcargo"`	Biogeochemical data (without S or T)
`argo_bio-traj_index.txt.gz`	-	Biogeochemical trajectory files
`argo_merge-profile_index.txt.gz`	`"merge"` or `"merged"`	Merged `"argo"` and `"bgc"` data
`argo_synthetic-profile_index.txt.gz`	`"synthetic"`	Synthetic data, successor to `"merge"`

Additionally, the `destdir` argument has a default of `~/data/argo`, where it should be noted that `~` is a short cut for `C:\Users\`. See `?getIndex()` for further description about the additional arguments for this command.

To get the index from the USGODAE server, the following code is used:

```
ai <- getIndex("argo")
```

subset()

As shown by Figure 1, the next step when working with the `argoFloats` package is to use the `subset()` function to focus on a subset of profiles. The `argoFloats` package provides tools to sift through profiles based on ID, time, geography, variable, institution, and ocean.

For geographic subsetting, the user has the ability to subset by `circle`, `rectangle`, or `polygon`.

To subset for specific groups of Argos off the coast of Bahamas, the following code is used:

```
# Subsetting by circle
aiCircle <- subset(ai, circle=list(longitude=-77.5, latitude=27.5, radius=50))
# Subsetting by polygon
lonPoly <- c(-76.5, -76.0, -75.5)
latPoly <- c(25.5, 26.5, 25.5)
aiPoly <- subset(ai, polygon=list(longitude=lonPoly, latitude=latPoly))
# Plotting the subsets together
CP <- merge(aiCircle, aiPoly)
plot(CP, bathymetry=FALSE)              # also, try using bathymetry=TRUE
```

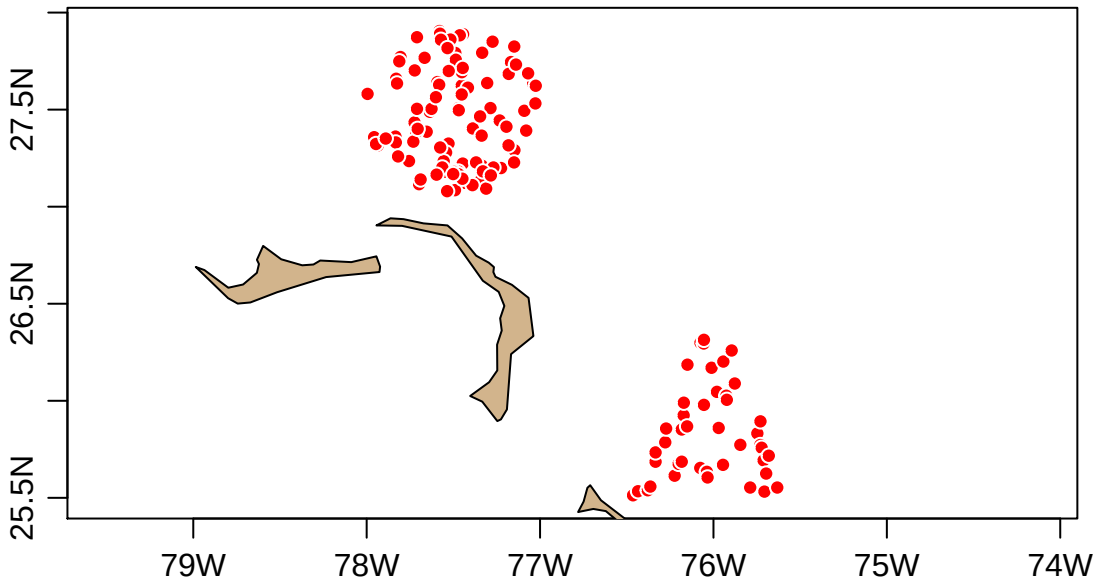

Figure 3: Figure 3: 50 km radius and polygon subset of argo floats found off the coast of Bahamas

Exercise 1: Produce the same plot as shown above, however, only include data from 2013.

Additionally, a practical use of `rectangle` subset can be shown while analyzing the section data found within the oce package. This data includes a westward transect from the Mediterranean outflow region across to North America. To view this data the following code is used:

```
data(section, package='oce')
plot(section)
```

Exercise 2: Create a rectangle subset for this transect and plot the positions for the year 2017.

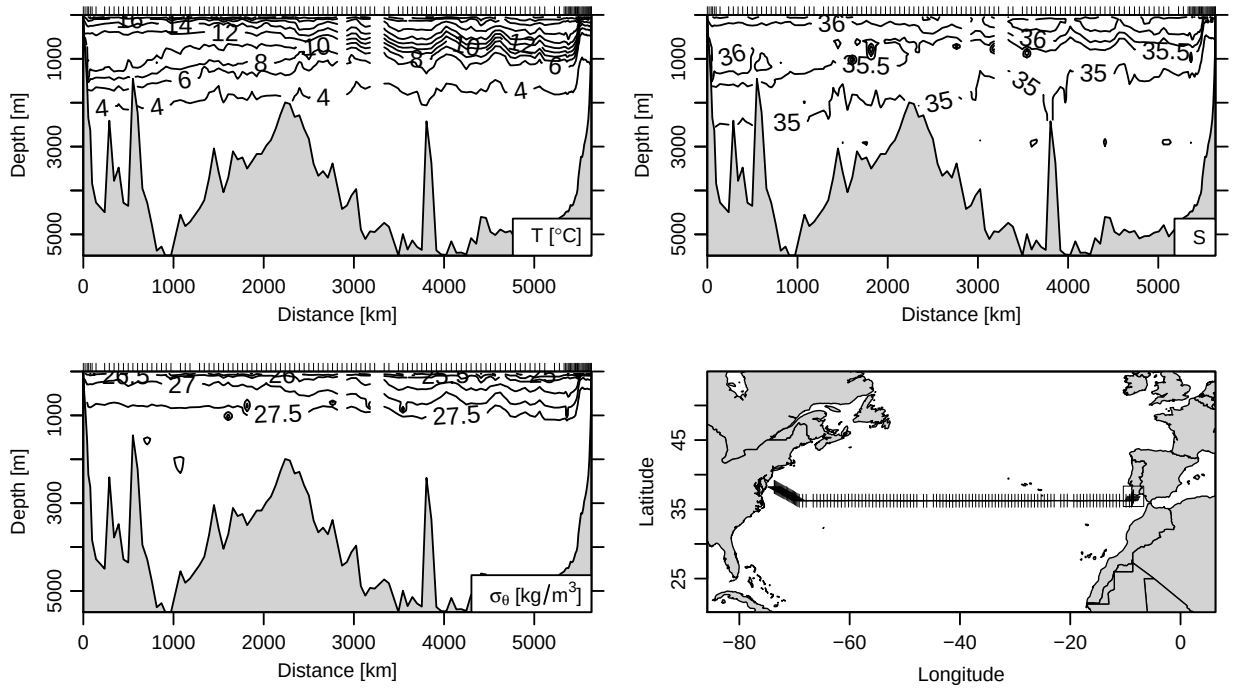

Figure 4: Figure 4: Line A03 section plot made by oce package.

getProfiles()

The next step within the `argoFloats` package is the `getProfiles()` function. This takes an index constructed with `getIndex()`, possibly after focusing with `subset,argoFloats-method()`, and creates a list of files to download from the server named in the index. Then these files are downloaded to the `destdir` directory, using filenames inferred from the source filenames. The value returned by `getProfiles()` is suitable for use by `readProfiles()` function.

readProfiles()

The `readProfiles()` command works with either a list of local netCDF files, or a `argoFloats` object type `profiles`, as created by `getProfiles()`.

The command can be useful for analyzing individual profiles, for example:

```
sub <- subset(ai, 1:2) # To subset for profiles
profiles <- getProfiles(sub)
argos <- readProfiles(profiles)
argosClean <- applyQC(argos)
pressure <- argosClean[['argos']][[1]][['pressure']]
temperature <- argosClean[['argos']][[1]][['temperature']]
plot(temperature, pressure, ylim=rev(range(pressure, na.rm=TRUE)),
     xlab='Temperature (C)', ylab='Depth (dbar)')
```

Exercise 3: Using the profile in the previous example, plot a TS diagram.

Solutions to Exercises

Exercise 1: Produce the same plot as shown above, however, only include data from 2013.

```
library(argoFloats)
ai <- getIndex("argo")
# Subsetting by circle
aiCircle <- subset(ai, circle=list(longitude=-77.5, latitude=27.5, radius=50))
# Subsetting by polygon
lonPoly <- c(-76.5, -76.0, -75.5)
latPoly <- c(25.5, 26.5, 25.5)
aiPoly <- subset(ai, polygon=list(longitude=lonPoly, latitude=latPoly))
#Subset by time
from <- as.POSIXct("2013-01-01", tz="UTC")
to <- as.POSIXct("2013-12-31", tz="UTC")
aic <- subset(aiCircle, time=list(from=from, to=to))
aip <- subset(aiPoly, time=list(from=from, to=to))
# Plotting the subsets together
cp <- merge(aic, aip)
plot(cp, bathymetry=FALSE)              # also, try using bathymetry=TRUE
```

Exercise 2: Create a rectangle subset for this transect and plot the positions for the year 2017.

```
library(argoFloats)
library(oce)
data(section, package='oce')
lat0 <- median(section[['latitude']])
lon0 <- median(section[['longitude']])
# Subset by rectangle
ai <- getIndex("argo")
latRect <- lat0 + c(-2,2)
lonRect <- lon0 + c(-30,30)
air <- subset(ai, rectangle=list(longitude=lonRect, latitude=latRect))
# Subset the rectangle by time
from <- as.POSIXct("2017-01-01", tz="UTC")
to <- as.POSIXct("2017-12-31", tz="UTC")
aiTime <- subset(air, time=list(from=from, to=to))
#Plot this subset
plot(aiTime, bathymetry=FALSE)          # also, try using bathymetry=TRUE
```

Exercise 3: Using the profile in the previous example, plot a TS diagram.

```
library(argoFloats)
ai <- getIndex("merge")
sub <- subset(ai, 1:2) # To subset for profiles
profiles <- getProfiles(sub)
argos <- readProfiles(profiles)
argosClean <- applyQC(argos)
plot(argosClean, which='TS')
```