The Journal of Open Source Software

R Tools for ObsPack, Receptors and Footprints (rtorf) for processing atmospheric observations in NOAA-GML ' ObsPack

Sergio Ibarra-Espinosa[1, 2] **and Lei Hu**[2]

1 Cooperative Institute for Research in Environmental Sciences, CU Boulder **2** NOAA Global Monitoring Laboratory

DOI:

Software

- Review
- Repository
- Archive

Submitted:
Published:

Summary

In this study, we present a new open-source R package `rtorf`, to read, process, select, and plot NOAA Observation Package (ObsPack) data products. We use a methane ObsPack data product as an example in this code base, but it can be easily modified to analyze ObsPack products for other greenhouse gasses. The R package starts with creating a catalog of all ObsPack files in each product. It then reads all files and creates one database. While reading each ObsPack file, it extracts site elevation and time zone information from the file header and calculates sampling altitude in meters above ground level and local time for individual samping events. Finally, it processes and selects observations for inverse modeling purposes. This package imports functions from data.table R package, which contains C bindings with parallel implementation via Open-MP (Dowle & Srinivasan, 2021). rtorf provides functions to perform these tasks in a transparent and efficient way, supporting open-source communities in environmental sciences.

The world is experiencing an accelerated global warming due to the accumulation of greenhouse gases (GHG) since the industrial revolution (Reidmiller et al., 2018). Greenhouse gas observations are critical to monitor the state of the atmosphere, quantify present and historical emissions, and understand global climate change. During the 21th Conference of Parties (COP21), it was established the Paris Accord, a multilateral effort reduce greenhouse emissions in order to limit the temperature increment of 1.5 degrees (Rhodes, 2016). Methane is a greenhouse gas responsible for half of the temperature increase since preindustrial levels. Furthermore, methane has a 9 years lifetime and a global warming potential of 30 over 100 years (S.EPA, 2023), with a current global radiative forcing of 0.650 Wm^{-2} (NOAA GML, 2023). Hence, in the 26 version of COP conference (Hunter, Salzman, & Zaelke, 2021), it was signed the Global Methane Pledge aiming reduce at least methane emissions 30% from 2020 levels by 2030, with U.S. as one of the parties (U.S. White House, 2021). Therefore, monitoring CH_4 observations, emissions and sinks has become critical. NOAA ObsPack data has been used to support many studies. For instance, the global methane budget for the year 2017 was 596 Tgy^{-1}, in agreement with other studies (Saunois et al., 2016, 2020) Lu et al. (2021), characterized global methane emissions in between 2014 and 2017, including a comparison with Greenhouse gases Observing SATellite (GOSAT) data. Saunois et al. (2016). At regional scale, Lu et al. (2022) performed another studied focused on north america using as priors local emissions inventories.

The National Oceanic and Atmospheric Administration (NOAA) and its Global Monitoring Laboratory (GML) has the mission of acquire, evaluate and make available long-term records of atmospheric gases[1]. To achieve that goal, GML gather own and other

[1]https://docs.r-wasm.org/webr/latest/

laboratories data, releasing observation in a compendium named ObsPack (Masarie, Peters, Jacobson, & Tans, 2014). Specifically, the CH_4 ObsPack GLOBALVIEW+ is a comprehensive product consisting in observations from aircrafts, ships, surface stations, towers and aircores. ObsPack include a descriptor named `datasetid` covering: aircraft-pfp, aircraft-insitu, aircraft-flask, surface-insitu, surface-flask, surface-pfp, tower-insitu, aircore, shipboard-insitu, and shipboard-flask. ObsPack product generally contains hundreds of files, each of which has different sampling frequencies, hours, and attributes. It takes time and effort to develop tools to read and process each ObsPack product and select observations of interest for specific modeling and data analysis purposes.

The NOAA ObsPack data is delivered to the public as NetCDF and text files (Masarie et al., 2014). The structure of the files including descriptor fields depend on the type of file. For instance, the metadata from aircrafts is different than surface stations, but all the files include concentrations and other critical fields. Given the complexity of ObsPack format, reading and analyzing the data can be cumbersome. The `rtorf` package provides the GHG science and research community a transparent and efficient tool to process ObsPack products for GHG modeling and analyses. In this manuscript we present `rtorf`, an R package to read, process and plot NOAA ObsPack data, a software useful and needed for the community (R Core Team, 2024). For this release, we are focused on the CH_4 ObsPack GLOBALVIEW+ product. The general process consists in creating a summary of the ObsPack files, reading them in an iteration process, filtering, and generating another output and plots.

Installation

To install `rtorf`, the user must have installed the R package remotes and run the following script. This process will install all the required dependencies, such as data.table, `cptcity`, an R package with more than 7000 color palettes, and `lubridate`, a package to manage time and dates (Grolemund & Wickham, 2011; Ibarra-Espinosa, 2017). Then, we call the libraries to load the function into the environment. `rtorf` is hosted at GitHub, which allows the implementation of checking the package installation in a variety OS.

```r
remotes::install_github("noaa-gml/rtorf")
library(rtorf)
```

Overview

`rtorf` is a collection of function organized together to read and process ObsPack files. The general process consists in create a summary of the ObsPack files, reading them in an iteration process, filter and generating another output. As CH_4 ObsPACKGLOBALViewplus 5.1, the product used in this manuscript, includes dataid, we produced a guide for each of of them available at https://noaa-gml.github.io/rtorf. Then, in this manuscript we present the processing of `aircraft-insitu`. The obspack product in this case is `obspack_ch4_1_GLOBALVIEWplus_v5.1_2023-03-08`.

Summary

We first call the libraries `rtorf` and `data.table`. Most of objects returned by `rtorf` are of class `data.table`. Then, we define the datasetid to be identified in the name of the files inside the directory with data. This process print a summary of the data and if the logical argument verbose is `TRUE`, print the file being read in each iteration, default is `FALSE`.

```r
library(rtorf)
library(data.table)
```

```
cate = c("aircraft-pfp",
         "aircraft-insitu",
         "aircraft-flask",
         "surface-insitu",
         "surface-flask",
         "surface-pfp",
         "tower-insitu",
         "aircore",
         "shipboard-insitu",
         "shipboard-flask")

obs <- "Z:/obspack/obspack_ch4_1_GLOBALVIEWplus_v5.1_2023-03-08/data/nc/"
index <- obs_summary(obs = obs,
                     categories = cate)
```

```
## Number of files of index: 429
##               sector     N
##               <char> <int>
##  1:     aircraft-pfp    40
##  2:  aircraft-insitu    15
##  3:    surface-flask   106
##  4:   surface-insitu   174
##  5:   aircraft-flask     4
##  6:          aircore     1
##  7:      surface-pfp    33
##  8:     tower-insitu    51
##  9:  shipboard-flask     4
## 10: shipboard-insitu     1
## 11:    Total sectors   429
## Detected 190 files with agl
## Detected 239 files without agl
```

Once the index of file is built, we can read each file. As we are directing to the nc directory in ObsPack, with NetCDF files inside, we use the function `obs_read_nc`. This function dumps the NetCDF information into a data.table with `long` format (Silge & Robinson, 2016). As the global attributes is attributes in the NetCDF would result in a data.table with too many columns, we used the argument `att` equals to FALSE, which is default. In ground-based datasetid the `solar_time` array is available. This is useful to select specific observations, for more information check the site documentation. At the moment, this array is not available for aircraft observations, hence we select `FALSE`. In this case, we select `verbose` equal to `TRUE`, to see the name of the files being read.

```
datasetid <- "aircraft-insitu"
df <- obs_read_nc(index = index,
                  categories = datasetid,
                  att = FALSE,
                  solar_time = FALSE,
                  verbose = TRUE)
```

```
## Searching aircraft-insitu...
## 1: ch4_above_aircraft-insitu_1_allvalid.nc
## 2: ch4_act_aircraft-insitu_428_allvalid-b200.nc
## 3: ch4_act_aircraft-insitu_428_allvalid-c130.nc
## 4: ch4_cob2003b_aircraft-insitu_59_allvalid.nc
## 5: ch4_eco_aircraft-insitu_1_allvalid.nc
## 6: ch4_hip_aircraft-insitu_59_allvalid.nc
## 7: ch4_iagos-caribic_aircraft-insitu_457_allvalid.nc
## 8: ch4_korus-aq_aircraft-insitu_428_allvalid-dc8.nc
## 9: ch4_man_aircraft-insitu_1_allvalid.nc
## 10: ch4_orc_aircraft-insitu_3_allvalid-merge10.nc
## 11: ch4_seac4rs_aircraft-insitu_428_allvalid-ER2.nc
## 12: ch4_seac4rs_aircraft-insitu_428_allvalid-dc8.nc
## 13: ch4_start08_aircraft-insitu_59_allvalid.nc
## 14: ch4_tom_aircraft-insitu_1_allvalid.nc
## 15: ch4_ugd_aircraft-insitu_1_allvalid.nc
```

The resulting `data.table` contains 59 columns and 2041758 observations. Furthermore, the size of `data.table` is 1.4 Gb. The data includes observations between 2003 and 2021. Now we can define some parameters to filter our data, like the year 2020 and spatially data below 8000 meters above sea level (masl) and focused over north America.

```r
df <- df[year == 2020 &
           altitude_final < 8000 &
           latitude < 80 &
           latitude > 10 &
           longitude < -50 &
           longitude > -170]
```

Now we have a `data.table` that contains 59 columns and 236 observations. The size of `data.table` is 0 Gb. We can add a column of time in format "POSIXct" and cut the seconds every 20 seconds. Usually, aircraft observations every 1 second. Then, We can simplify the data by calculating averages every 20 seconds. We perform this task by cutting time very 20 seconds. Then, we add a new column with the mandatory name `key_time`, which will be used to aggregate data with "POSIXct" class, but every 20 seconds.

```r
df <- obs_addtime(df)
```

```
## Adding timeUTC
## Adding timeUTC_start
## Adding timeUTC_end
## Found time_interval
```

```r
df$sec2 <- obs_freq(x = df$second,
                    freq = seq(0, 59, 20))
df$key_time <- ISOdatetime(year = df$year,
                           month = df$month,
                           day = df$day,
                           hour = df$hour,
                           min = df$minute,
                           sec = df$sec2,
                           tz = "UTC")
df[1, c("timeUTC", "key_time")]
```

```
##                 timeUTC            key_time
##                  <POSc>              <POSc>
## 1: 2020-01-08 22:59:55 2020-01-08 22:59:40
```

now we can aggregate the data using the function `obs_agg`. The argument cols indicate which columns will be averaged. Then, we add local time with the function `obs_addltime` and we re order the data.table.

```r
df2 <- obs_agg(df,
               cols =  c("year", "month", "day", "hour", "minute", "second",
```