robspack An R package for processing atmospheric greenhouse gas observations in NOAA's ObsPack

Sergio Ibarra-Espinosa[a,b,1,*], Lei Hu[b]

[a] *Cooperative Institute for Research in Environmental Sciences, CU Boulder, 216 UCB, University of Colorado Boulder campus, Boulder, 80309, Colorado, United States*
[b] *NOAA Global Monitoring Laboratory, 325 Broadway, 80309, Colorado, United States*

Abstract

In this study, we present a new open-source R package `robspack`, 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 and Srinivasan, 2021). data.table is faster than other Python, Julia and R implementations for data-science, providing a strong basis for robspack. robspack provides functions to perform these tasks in a transparent and efficient way, supporting open-source communities in environmental sciences.

Keywords: ObsPack, NOAA, Greenhouse gases

1. Introduction

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 (U.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 et al., 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.

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[2]. To achieve that goal, GML gather own and other laboratories data, releasing observation in a compendium named ObsPack (Masarie et al., 2014). Specifically, the CH_4 ObsPack GLOBALVIEW+ is a comprehensive product consisting in observations from aircrafts, ships, surface stations, towers and aircores. However, each

*Corresponding author
Email addresses: `sergio.ibarra-espinosa@noaa.gov` (Sergio Ibarra-Espinosa), `lhu@noaa.gov` (Lei Hu)
[1] This is the first author footnote.
[2] https://gml.noaa.gov/about/aboutgml.html

ObsPack product generally contains hundreds of files, each of which has different sampling frequencies, hours, and hundreds of lines of headers. 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.

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., 2020, 2016) 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. Furthermore, Hu et al. (2023) presented trends and cycles of methane over the US.

The NOAA ObsPack data is delivered to the public as NetCDF and text files. The structure of the files including descriptor fields depend on the type of file. For instance, the metadata from aircrfats is different than surface stations, but all the files include concentrations and other critical fields. An important information is the scale of measurement, which isXX. Cite Importance of scale and what is WMOX2014. Given the complexity of ObsPack format, reading and analyzing the data can be cumbersome. The `robspack` 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 `robspack`, an R package to read, process and plot NOAA ObsPack data. For this release, we are focued on the CH_4 ObsPack GLOBALVIEW+ product. The general process consists in create a summary of the ObsPack files, reading them in an iteration process, filter and generating another output and plots.

2. Installation

To install `robspack`, 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 and Wickham, 2011; Ibarra-Espinosa, 2017). Then, we call the libraries to load the function into the environment.

```
remotes::install_github("ibarraespinosa/robspack")
library(robspack)
```

3. Overview

`robspack` is a collection of function organized together to read and process ObsPack files (Masarie et al., 2014). The general process consists in create a summary of the ObsPack files, reading them in an iteration process, filter and generating another output. In other to facilitate this task, we are letting to the public the directory shown below. This directory includes a README file with all the required information and scripts generate process each category. The structure of the directory is:

https://github.com/ibarraespinosa/robspack/tree/main/rscripts

```
|--"README.md"
|-- r
    |-- index.R
    |-- aircore_year.R
    |-- aircraft_year.R
    |-- flask_non_noaa_year.R
    |-- surface_insitu_year.R
    |-- tower_insitu_year.R
    |-- inputs_inv.R
```

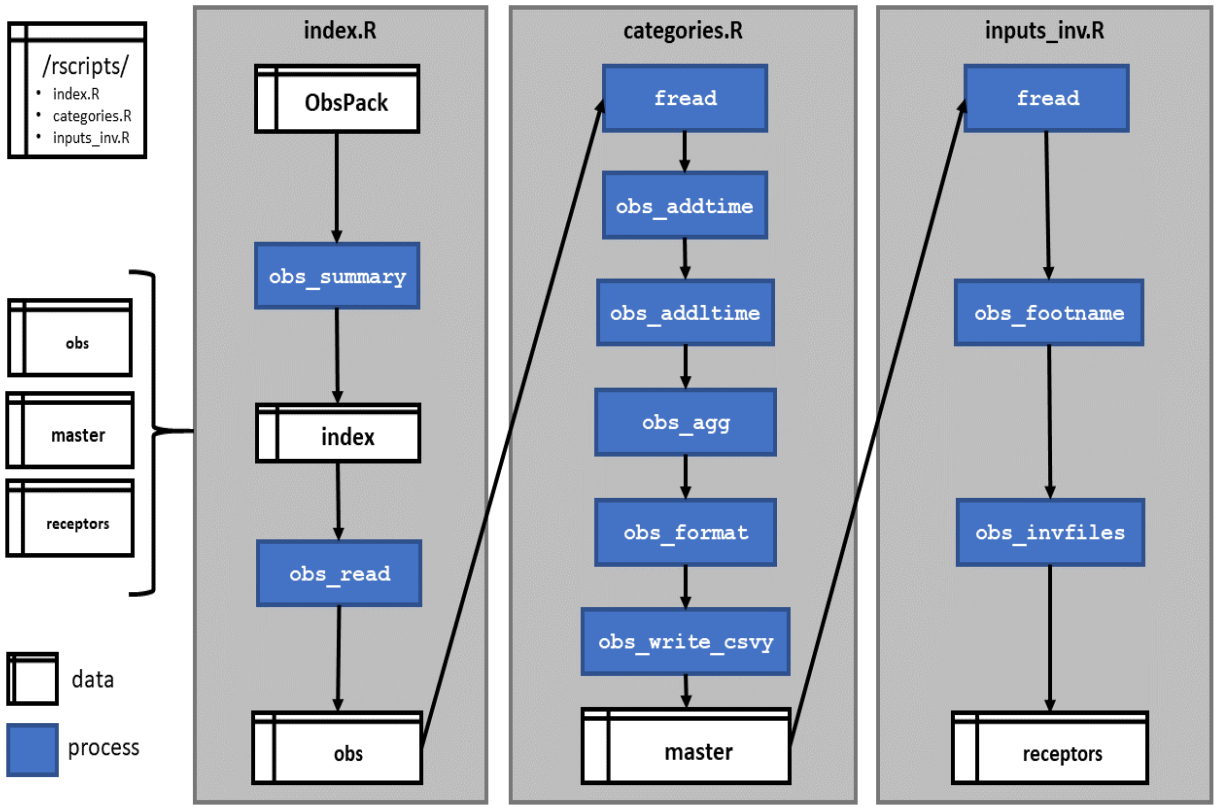

Figure 1: Methane observations (ppb) by towers-insitu

Table 1: Summary of ObsPack

sector	N
aircraft-pfp	40
aircraft-insitu	11
flask	101
surface-insitu	124
aircore	1
surface-pfp	33
tower-insitu	51
shipboard-insitu	1
Total sectors	362

The file `index.R` creates a summary of ObsPack, generates the directories master, receptor and obs, and store the summaries in obs directory for each category. Then, the scripts for each category are run to generate the master and receptor files. The next step consists in running HYSPLIT and obtaining footprints (not in the scope of this manuscript). At last, the script `inputs_inv.R` checks each footprint file for each category and generates the final receptor list files. The process is described in detail in the following example for tower insitu.

The first step consists in constructing a summary for the ObsPack. This is required to read the data, but also, identify agl, which is present in some of the file names. This function returns a data.frame. Optionally, the user can indicate a path to store the data.frame. obs_summary also prints a summary of the data. The second argument is the categories, and by default includes the categories shown below, to account for all the files. Then the summary data.frame contains the columns id as the full path to each file, name which is the name or relative path of the file, n which is just an id, sector such as tower, and the column agl which indicates the agl indicated in the name of the file if available. To read the documentation of this function, the user must run `?obs_summary`.

```
categories <- c("aircraft-pfp","aircraft-insitu","surface-insitu",
  "aircore","surface-pfp","tower-insitu","shipboard-insitu","flask")
obs <- "../../../obspack_ch4_1_GLOBALVIEWplus_v4.0_2021-10-14/data/txt"
index <- obs_summary(obs = obs, verbose = F)
```

There are 362 files in the ObsPack directory. The printed information also shows the total at the bottom, as the sum of the individual file by sector. This is to ensure that the sum of files is equal to the total number of files found, shown at the top. furthermore, the printed information also shows that there are 136 files with the agl explicitly mentioned in the name of the file. Sometimes we need more information about the site. For instance, what do the observations start and end. Then, we added the function `obs_table`, which calculates statistics summary of "time" and other numeric variables by file name, sector, site, altitude and mode. For instance, the observations in the site "SCT" in South Carolina, USA, were between "2015-08-19 21:30:00 UTC" and "2020-12-31 23:30:00 UTC". In figure @ref(fig:tiobs) we see the average of methane concentrations over each tower-insitu site. Higher concentration are found over Russia. Over the United States (U.S.), most of towers are found in the south to east coast.

3.1. Functions

The `robspack` functions are shown in table 1.

Table 1. Functions and classes in `robspack`.

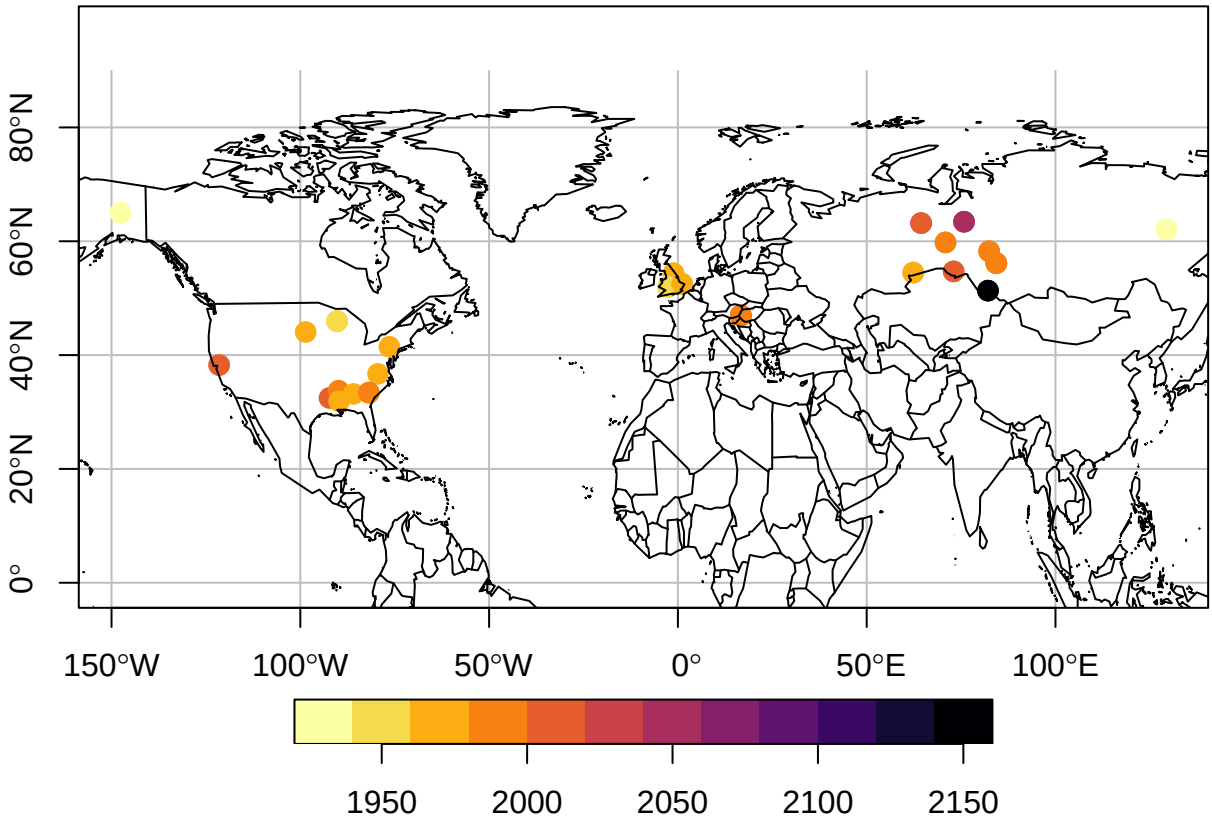

Figure 2: Methane observations (ppb) by towers-insitu

Function	Description
`invfile`	Class with `print`, `summary` and `plot` methods
`obs_addltime()`	Add local time based on metadata and longitude
`obs_addtime()`	Add UTC time
`obs_agg()`	Aggregates ObsPack by time
`obs_find_receptors()`	Find expected receptors and NetCDF files
`obs_format()`	Format for some columns of data.table
`obs_freq()`	Return numeric vector in intervals
`obs_invfiles()`	Construct `invfile` objects
`obs_list.dt()`	Rbind list of data.frames with different names
`obs_meta()`	Reads ObsPack metadata
`obs_out()`	Outersect, opposed as intersect
`obs_rbind()`	Rbind data.frames with different names
`obs_read()`	Read files, and add metadata as columns
`obs_read_csvy()`	Read csvy file and prints yaml header
`obs_roundtime()`	Round seconds from "POSIXct" "POSIXt" classes
`obs_summary()`	Construct summary of ObsPack as a data.frame
`obs_table()`	Return a data.frame with summary of data
`obs_trunc()`	Trunc numbers with a desired number of decimals
`obs_write()`	Write CSVY to disk, YAML followed by tabulated

4. Application for towers in situObsPack summary

4.1. Read data

Once the summary is built, the function obs_read will read the files available in the index file previously generated. Here we selected the category "tower-insitu". The argument verbose prints which files are being read each time, by default. At the end, this function prints the total number of observations by type of altitude (agl or asl).

```
df <- obs_read(index = index,
               categories = "tower-insitu",
               verbose = FALSE)
```

We added a function to plot the data read from ObsPack. The y-axis is the field value and the x-axis is by default time. The data illustrated sorted by color is the field site_code, with the default number of 3 sites. The argument pal is to define the color palette, used by the internally imported function cptcity::cpt.

```
obs_plot(dt = df, time = "time", yfactor = 1e+09, cex = 0.5)
```

```
## Found the following sites:
##  [1] AZV   BRZ   BSD   CRV   DEM   DVV   GCI01 GCI02 GCI03 GCI04 HUN   IGR
## [13] KRS   LEF   MRC   NOY   RGL   SCT   SVV   TAC   VGN   WGC   WSD   YAK
## Plotting the following sites:
## [1] AZV BRZ
```

Before sub setting the data, tower-insitu has about 2.32 million observations. These observations are made between 2004 and 2020. The identification of the altitude and type is critical, then we developed an approach based on the availability of data:

1. Identify agl from the name of the tile.
2. If agl not present, search fill_values used in elevation and transform them into NA (not available)

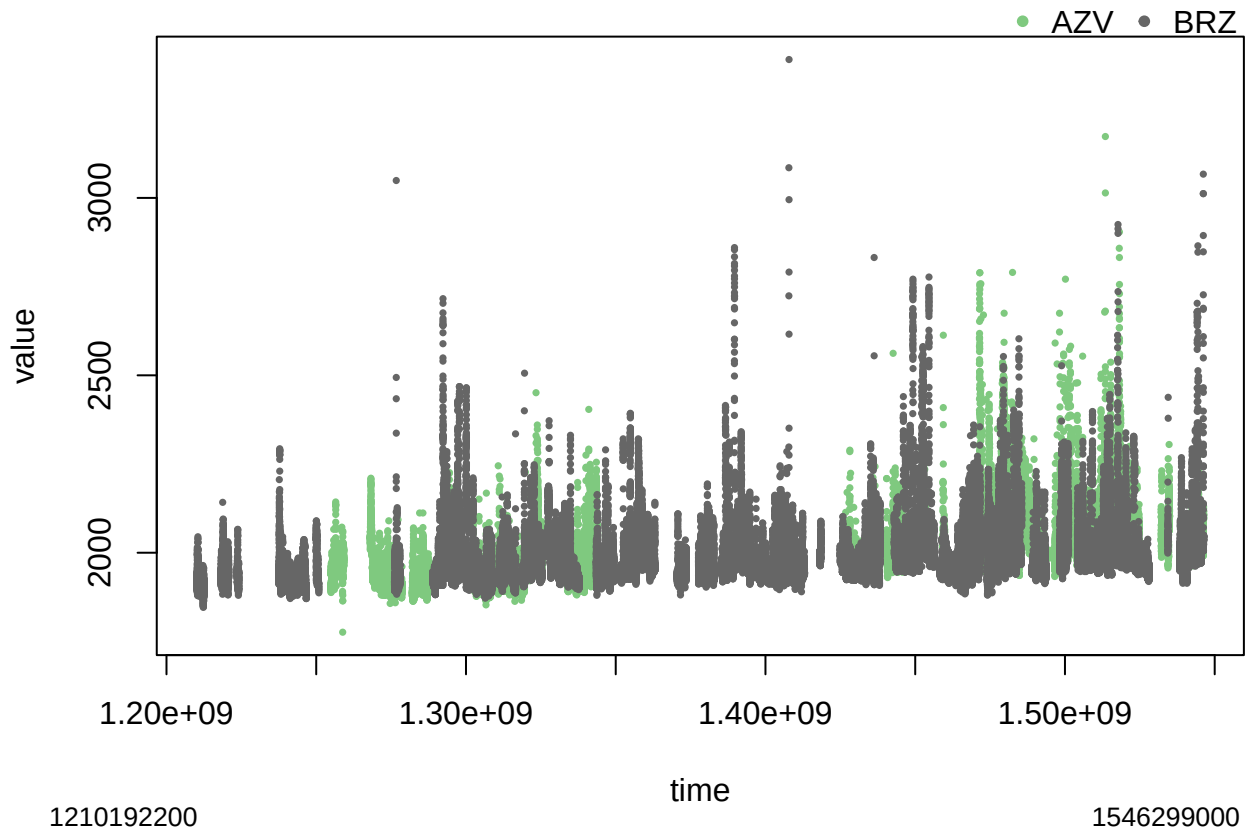

Figure 3: First two sites in ObsPack

3. If agl is not present, agl = altitude - elevation.
4. If there are some NA in elevation, will result some NA in agl
5. A new column is added named altitude_final to store agl or asl
6. Another column named type_altitude is added to identify agl or asl.
7. If there is any case NA in altitude_final, type_altitude is "not available"

4.2. Filtering

ObsPack includes global observations and sometimes we need to extract data for a specific region and periods of time. In this part we include spatial and temporal parameters to filter data. The year of interest is 2020, but we also included December of 2019 and January of 2021. The spatial filtering is done by using the coordinates, in this case covering North America. After filtering by space and time, we have 2.322369×10^6 million observations.

```
north <- 80
south <- 10
west <- -170
east <- -50
max_altitude <- 8000
evening <- 14:15

yy <- 2020
df <- rbind(df[year == yy - 1 & month == 12],
            df[year == yy],
            df[year == yy + 1 & month == 1])

df <- df[altitude_final < max_altitude &
           latitude < north &
           latitude > south &
           longitude < east &
           longitude > west]
```

4.3. Time

The function obs_addtime adds time columns timeUTC, timeUTC_start which shows the start time of each observation and timeUTC_end which shows the end time for each observation. Then we need to identify the local time with the function add_ltime. This is important because to identify observations in the evening in local time for modeling purposes. add_ltime uses two methods, first identifying the time difference with utc by identifying the metadata column "site_utc2lst". If this information is not available, with the aircrafts for instance, the local time is calculated with an approximation based on longitude:

$$lt = UTC + longitude/15 * 60 * 60$$

Where lt is the local time, UTC the time, and longitude is the coordinate. Then, the time is cut every two hours. We also identify the local time to select evening hours.

```
df2 <- obs_addtime(df)
```

```
## Adding timeUTC
## Adding timeUTC_start
## Adding timeUTC_end
```

```
df2$timeUTC <- cut(x = df2$timeUTC+3600,
                   breaks = "2 hour") |>
  as.character() |>
  as.POSIXct(tz = "UTC")
df3 <- obs_addltime(df2)
df3 <- df3[lh %in% evening]
```

Now there are 8391 observations. At this point we can calculate the averages of several columns by the cut time. The function obs_agg does this aggregation as shown in the following lines of code. The argument gby establish the function used to aggregate cols, in this case the function "mean" by time and altitude. Finally, we add local time again.

```
df4 <- obs_agg(dt = df3,
               gby = "mean",
               cols = c("value", "latitude", "longitude", "type_altitude",
                        "dif_time", "year_end", "site_utc2lst"),
               verbose = FALSE,
               byalt = TRUE)
```

```
## Detecting dif_time. Adding ending times
```

```
df5 <- obs_addltime(df4)
```

Now there are 4394 observations. Towers can have observations at different heights. Here we need to select one site with the observations registered at the highest height. The column with the height is named altitude_final and the max altitude was named max_altitude. Then, we print the altitudes of each site.

```
df5[,
    max_altitude := max(altitude_final),
    by = site_code]
df5[,
    c("site_code",
      "altitude_final",
      "max_altitude")] |> unique()
```

```
##     site_code altitude_final max_altitude
##  1:       CRV           17.0           32
##  2:       CRV           32.0           32
##  3:       CRV            4.9           32
##  4:       LEF          122.0          396
##  5:       LEF           30.0          396
##  6:       LEF          396.0          396
##  7:       SCT          305.0          305
##  8:       SCT           31.0          305
##  9:       SCT           61.0          305
## 10:       WGC           30.0          483
## 11:       WGC          483.0          483
## 12:       WGC           91.0          483
```

4.4. Saving master as text and csvy

Now that we have all the required information, we can save the files. Here, we name the data.frame as master, because it contains all the information. This is important because some fields can be used in the

future, and for traceability. For convenience, time variables are transformed into character before writing into the disk. The separation is space " ".

```
master <- df5
master$timeUTC <- as.character(master$timeUTC)
master$timeUTC_end <- as.character(master$timeUTC_end)
master$local_time <- as.character(master$local_time)

fwrite(master,
       file = "tower_insitu_2020.txt",
       sep = " ")
```

The format Comma Separated Value with YAML (CSVY)[3] consists in a typical CSV with a YAML header. The function obs_write includes the argument notes which allows adding custom notes at the header of the file. Below the notes, obs_write adds the output of the R function **str**, which provides a vertical summary of the data, known as structure.

```
obs_write_csvy(dt = master,
               notes = "tower 2020",
               out = "tower_insitu_2020.csvy")
```

To check the YAML header we read the first 38 lines of the files that were generated. Here we can see the column names, type of data and first observations. The YAML header is delimited by the characters "- - -" (not shown here).

```
readLines("tower_insitu_2020.csvy")[1:38]
```

4.5. Saving receptors

We need to filter some columns from the master files in a new object called receptors. This is needed because internally we run HYSPLIT (Stein et al., 2015) using the information from the receptors. In the case of a tower, we need to select observations with the highest altitude. The specific columns are selected as shown on the following code. We are selecting the ending times, because later HYSPLIT is run backwards based on the time of measurement, between ending and starting times. The columns about time are formatted to have two characters. For instance, the month 1, is formatted as "01". We also need to filter for type_altitude equal 0, representing agl observations , or equal to 1, asl.

```
receptor <- master[altitude_final == max_altitude,
                   c("site_code",
                     "year", "month", "day",
                     "hour", "minute", "second",
                     "latitude", "longitude",
                     "altitude_final", "type_altitude",
                     "year_end", "month_end", "day_end", "hour_end",
                     "minute_end", "second_end")]
receptor$altitude_final <- round(receptor$altitude_final)
receptor <- obs_format(receptor)

if(nrow(receptor_agl) > 0) {
```

[3] https://csvy.org/

```
  fwrite(x = receptor_agl,
         file = "paper/receptor_tower_insitu_2020_AGL.txt"),
  sep = " ")}

if(nrow(receptor_asl) > 0) {
  fwrite(x = receptor_asl,
         file = "paper/receptor_tower_insitu_2020_ASL.txt"),
  sep = " ")}
```

4.6. Recommendation for other applications

The approach to generate receptors depends on each type of observation and other considerations. For instance, aircraft with continuous observations at each second can be filtered and averaged every 20 seconds. In that way, the footprints are still representative and it would not be necessary to run HYSPLIT every second. Of course, it depends on the application and objective of the study. For this manuscript, we are presenting the generation of receptors based on tower observations.

5. Conclusion

In this manuscript we presented an robskpack, an R package to read and process CH4 ObsPack GLOBALVIEW+ published by the Global Monitoring Laboratory (GML) from the National Oceanographic and Atmospheric Administration (NOAA). robspack reads the text data which have different headers and organizes them in a common format. Then, this software applies calculations to filter observations by time and space. Finally, this software generates receptors in a suitable format that allows it to run HYSPLIT and generate footprints. This software does not provide methods to run HYSPLIT, but the user can follow the site https://www.ready.noaa.gov/HYSPLIT.php.

6. Acknowledgements

Funding: This project is funded by the NOAA Climate Program Office AC4 and COM programs (NA21OAR4310233 / NA21OAR4310234). This research was supported by the NOAA cooperative agreement NA22OAR4320151.

References

Matt Dowle and Arun Srinivasan. *data.table: Extension of 'data.frame'*, 2021. URL `https://CRAN.R-project.org/package=data.table`. R package version 1.14.2.

Garrett Grolemund and Hadley Wickham. Dates and times made easy with lubridate. *Journal of Statistical Software*, 40(3): 1–25, 2011. URL `https://www.jstatsoft.org/v40/i03/`.

Lei Hu, Arlyn Andrews, Stephen Montzka, Ed Dlugokencky, Scot Miller, Sergio Ibarra-Espinosa, Colm Sweeney, Lori Bruwhiler, Natasha Miles, and Kenneth Davis. Trend and seasonal cycle of us methane emissions. Technical report, Copernicus Meetings, 2023.

David B Hunter, James E Salzman, and Durwood Zaelke. Glasgow climate summit: Cop26. *UCLA School of Law, Public Law Research Paper*, (22-02), 2021.

Sergio Ibarra-Espinosa. *cptcity: incorporating the cpt-city archive into R*, 2017. URL `https://CRAN.R-project.org/package=cptcity`. Colour gradients from http://soliton.vm.bytemark.co.uk/pub/cpt-city/.

Xiao Lu, Daniel J Jacob, Yuzhong Zhang, Joannes D Maasakkers, Melissa P Sulprizio, Lu Shen, Zhen Qu, Tia R Scarpelli, Hannah Nesser, Robert M Yantosca, et al. Global methane budget and trend, 2010–2017: complementarity of inverse analyses using in situ (globalviewplus ch< sub> 4</sub> obspack) and satellite (gosat) observations. *Atmospheric Chemistry and Physics*, 21(6):4637–4657, 2021.

Xiao Lu, Daniel J Jacob, Haolin Wang, Joannes D Maasakkers, Yuzhong Zhang, Tia R Scarpelli, Lu Shen, Zhen Qu, Melissa P Sulprizio, Hannah Nesser, et al. Methane emissions in the united states, canada, and mexico: evaluation of national methane emission inventories and 2010–2017 sectoral trends by inverse analysis of in situ (globalviewplus ch< sub> 4</sub> obspack) and satellite (gosat) atmospheric observations. *Atmospheric Chemistry and Physics*, 22(1):395–418, 2022.

KA Masarie, W Peters, AR Jacobson, and PP Tans. Obspack: a framework for the preparation, delivery, and attribution of atmospheric greenhouse gas measurements. *Earth System Science Data*, 6(2):375–384, 2014.

NOAA GML. The noaa annual greenhouse gas index (aggi), 2023. URL https://gml.noaa.gov/aggi/aggi.html. accessed 2023-07-10.

David R Reidmiller, Christopher W Avery, David R Easterling, Kenneth E Kunkel, Kristin LM Lewis, Thomas K Maycock, and Brooke C Stewart. Impacts, risks, and adaptation in the united states: Fourth national climate assessment, volume ii. 2018. doi: doi:10.7930/NCA4.2018. URL https://nca2018.globalchange.gov/downloads/NCA4_Ch00_Front-Matter.pdf. accesed 6/27/2023.

Christopher J Rhodes. The 2015 paris climate change conference: Cop21. *Science progress*, 99(1):97–104, 2016.

Marielle Saunois, Philippe Bousquet, Ben Poulter, Anna Peregon, Philippe Ciais, Josep G Canadell, Edward J Dlugokencky, Giuseppe Etiope, David Bastviken, Sander Houweling, et al. The global methane budget 2000–2012. *Earth System Science Data*, 8(2):697–751, 2016.

Marielle Saunois, Ann R Stavert, Ben Poulter, Philippe Bousquet, Josep G Canadell, Robert B Jackson, Peter A Raymond, Edward J Dlugokencky, Sander Houweling, Prabir K Patra, et al. The global methane budget 2000–2017. *Earth system science data*, 12(3):1561–1623, 2020.

AF Stein, Roland R Draxler, Glenn D Rolph, Barbara JB Stunder, MD Cohen, and Fong Ngan. Noaa's hysplit atmospheric transport and dispersion modeling system. *Bulletin of the American Meteorological Society*, 96(12):2059–2077, 2015.

U.S. White House. Joint us-eu press release on the global methane pledge, Sep 2021. URL https://www.whitehouse.gov/briefing-room/statements-releases/2021/09/18/joint-us-eu-press-release-on-the-global-methane-pledge/. accessed 2023-07-10.

U.S.EPA. Understanding global warming potentials, 2023. URL https://www.epa.gov/ghgemissions/understanding-global-warming-potentials#:~:text=Methane%20(CH4)%20is%20estimated,more%20energy%20than%20CO2. accessed 2023-07-10.