Downloading and Preparing NOAA OISST Data

Robert W Schlegel and AJ Smit

2021-10-26

Overview

In this vignette we will see how to retrieve and prepare Reynolds optimally interpolated sea surface temperature (OISST) data for calculating marine heatwaves (MHWs). The OISST product is a global 1/4 degree gridded dataset of Advanced Very High Resolution Radiometer (AVHRR) derived SSTs at a daily resolution, starting on 1 September 1981. The source of the data is currently the NOAA NCDC.

Each daily global file, when not compressed, is around 8.3 MB, so they add up to a large amount of data when a time series of the recommended 30 year minimum duration for the detection of MHWs is downloaded. If one were to download all of the data currently available it would exceed 100 GB of total disk space. It is therefore best practice to download only a subset of the data that matches one's study area. Thanks to the `rerddap` package this is incredibly easy to do in `R`.

Should one want to download the full global dataset, each daily global file is available in netCDF format and is roughly 1.6 MB. This means that one full year of global data will be roughly 600 MB, and the full dataset roughly 25 GB. This is however when the data are very compressed. If we were to attempt to load the entire uncompressed dataset into our memory at once it would take more than 200 GB of RAM. That is well beyond the scope of any current laptop so in the second half of this vignette we will see how to download the full OISST dataset before then seeing how we can load only a subset of the data into the R environment for use with further analyses.

This vignette may appear very long and complex but it has been written in an attempt to keep the process of downloading and working with satellite data as straight-forward and easy to follow as possible. Before we begin with all of the code etc. please note that for almost all applications it is only necessary to use the first method outlined below. For most users the second download method in this vignette can simply be skipped.

Setup

For this vignette we will be accessing the NOAA OISST dataset on this ERDDAP server for the subsetted data, while the global data are indexed here. One may download the data on both servers manually by using the ERDDAP UI or clicking on each indexed file individually. But programming languages like R are designed to prevent us from needing to experience that sort of anguish. Below we will load the libraries we need in order to have R download all of the data that we may need. If any of the lines of code in the following chunk do not run it means that we will need to first install that package. Uncomment the line of code that would install the problem package and run it before trying to load the library again.

```
# The packages we will need
# install.packages("dplyr")
# install.packages("lubridate")
# install.packages("ggplot2")
# install.packages("tidync")
# install.packages("doParallel")
# install.packages("rerddap")
# install.packages("plyr") # Note that this library should never be loaded, only installed

# The packages we will use
library(dplyr) # A staple for modern data management in R
library(lubridate) # Useful functions for dealing with dates
library(ggplot2) # The preferred library for data visualisation
library(tidync) # For easily dealing with NetCDF data
library(rerddap) # For easily downloading subsets of data
library(doParallel) # For parallel processing
```

With our packages loaded we may now begin downloading and preparing our data for further use. Please use the table of contents on the right side of the screen to jump between the different download methods as desired. We will break each different method down into smaller steps in order to keep this process as clear as possible. Before we begin I need to stress that this is a very direct and unrestricted method for accessing these data and I urge responsibility in only downloading as much data as are necessary. Please do not download the entire dataset unless you have a specific need for it.

Downloading subsetted data

File information

Before we begin downloading the subsetted data for our study area we need to make sure that they are currently available on an ERDDAP server. The location of the NOAA OISST data has changed in the past so it should not be assumed that the current location will exist in perpetuity. Finding the server on which these data are located can be a cup game at times.

```
# The information for the NOAA OISST data
rerddap::info(datasetid = "ncdcOisst21Agg_LonPM180", url =
         "https://coastwatch.pfeg.noaa.gov/erddap/")

# Note that there is also a version with lon values from 0 yo 360
rerddap::info(datasetid = "ncdcOisst21Agg", url = "https://coastwatch.pfeg.noaa.gov/erddap/")
```

With our target dataset identified we may now begin the download with the `griddap()` function. While putting this vignette together however I noticed one little hiccup in the work flow. It seems that the ERDDAP server does not like it when one tries to access more than nine consecutive years of data in one request, regardless of the spatial extent being requested. So before we download our data we are going to make a wrapper function that helps us control the range of times we want to download. This will reduce the amount of redundant coding we would otherwise need to do.

Download function

```
# This function downloads and prepares data based on user provided start and end dates
OISST_sub_dl <- function(time_df){
  OISST_dat <- griddap(x = "ncdcOisst21Agg_LonPM180",
                               url = "https://coastwatch.pfeg.noaa.gov/erddap/",
                               time = c(time_df$start, time_df$end),
                               zlev = c(0, 0),
                               latitude = c(-40, -35),
                               longitude = c(15, 21),
                               fields = "sst")$data %>%
    mutate(time = as.Date(stringr::str_remove(time, "T00:00:00Z"))) %>%
    dplyr::rename(t = time, temp = sst) %>%
    select(lon, lat, t, temp) %>%
    na.omit()
}
```

In the wrapper function above we see that we have chosen to download only the ‘sst’ data out of the several variables (‘fields’) available to us. We also see that we have chosen the spatial extent of latitude -40 to -35 and longitude 15 to 21. This a small window over some of the Agulhas Retroflection to the south west of South Africa. A larger area is not being chosen here simply due to the speed constraints of downloading the data and detecting the events therein. One may simply change the longitude and latitude values above as necessary to match the desired study area. The function will also be re-labelling the ‘time’ column as ‘t’, and the ‘sst’ column as ‘temp’. We do this so that they match the default column names that are expected for calculating MHWs and we won’t have to do any extra work later on.

One must note here that depending on the RAM available on one’s machine, it may not be possible to handle all of the data downloaded at once if they are very large (e.g. > 5 GB). The discussion on the limitations of the R language due to its dependence on virtual memory is beyond the scope of this vignette, but if one limits one’s downloads to no more than several square pixels at a time that should be fine. Were one to try to download the whole Indian Ocean, for example, that may cause issues if being run on a laptop or computer of a similar power.

Date range

With our wrapper function written we would now need to run it several times in order to grab all of the OISST data from `1982-01-01` to `2019-12-31`. Even though each year of data for the extent used in this vignette is only ~360 KB, the server does not like it when more than 9 years of consecutive data are requested. The server will also end a users connection after ~17 individual files have been requested. Because we can’t download all of the data in one request, and we can’t download the data one year at a time, we will need to make requests for multiple batches of data. To accomplish this we will create a dataframe of start and end dates that will allow us to automate the entire download while meeting the aforementioned criteria.

```
# Date download range by start and end dates per year
dl_years <- data.frame(date_index = 1:5,
                       start = as.Date(c("1982-01-01", "1990-01-01",
                                         "1998-01-01", "2006-01-01", "2014-01-01")),
                       end = as.Date(c("1989-12-31", "1997-12-31",
                                       "2005-12-31", "2013-12-31", "2019-12-31")))
```

Download/prep data

One could also use the `plyr` suite of functions to automate the process of downloading and processing multiple files, but I’ve chosen here to stick with the `tidyverse` native approach. If the below chunk of code fails or times out, simply re-run it until all of the data have been downloaded.

It is worth pointing out here that these data are downloaded as cached files on the users computer by using the `hoardr` package. This means that if one runs the same command again, it will not re-download the data because it first looks in the folder where it has automatically cached the data for you and sees that it may simply draw the data from there. No need to change anything or write a second script for loading data.

```r
# Download all of the data with one nested request
# The time this takes will vary greatly based on connection speed
system.time(
  OISST_data <- dl_years %>%
    group_by(date_index) %>%
    group_modify(~OISST_sub_dl(.x)) %>%
    ungroup() %>%
    select(lon, lat, t, temp)
) # 38 seconds, ~8 seconds per batch
```

If the above code chunk is giving errors it is likely due to one's Internet connection timing out. There are also rare instances where the NOAA server is not responding due to an issue on their end. Any connection based issues may be resolved by simply waiting for a few minutes, or by ensuring a stable connection.

Visualise data

Before we save our data for later use it is good practice to visualise them.

```r
OISST_data %>%
  filter(t == "2019-12-01") %>%
  ggplot(aes(x = lon, y = lat)) +
  geom_tile(aes(fill = temp)) +
  # borders() + # Activate this line to see the global map
  scale_fill_viridis_c() +
  coord_quickmap(expand = F) +
  labs(x = NULL, y = NULL, fill = "SST (°C)") +
  theme(legend.position = "bottom")
```

Save data

With the data downloaded and prepared for further use (and a test visual run), all that's left to do is save them.

```r
# Save the data as an .Rds file because it has a much better compression rate than .RData
saveRDS(OISST_data, file = "~/Desktop/OISST_vignette.Rds")
```

Note above that I have chosen to save the file to my desktop. This is not normally where one (hopefully!) would save such a file. Rather one would be saving these data into the project folder out of which one is working. In the next vignette we will see how to detect MHWs in gridded data using the data downloaded here.

Downloading global data

The method for downloading and preparing NOAA OISST data outlined in the first half of this vignette should be considered best practice for all applications except those that specifically need to look at the entire globe. If one needs to download the global dataset then it is preferable to go straight to the source. Note that one may still download the full global dataset using the methods above by setting the lon/lat extent to be the full width and height of the globe. The method outlined below will download over 13,000 individual files. This makes dealing with individual files very easy, but agglomerating them into one file can be very time consuming.

File information

The first step in downloading the full global dataset is to tell you computer where they are. There is an automated way to do this but it requires a couple of additional packages and we aim to keep this vignette as simple and direct as possible. For our purposes today we will manually create the URLs of the files we want to download.

```
# First we tell R where the data are on the interwebs
OISST_base_url <- "https://www.ncei.noaa.gov/data/sea-surface-temperature-optimum-
         interpolation/v2.1/access/avhrr/"
# Note that one may go to this URL in any web browser to manually inspect the files

# Now we create a data.frame that contains all of the dates we want to download
  # NB: In order to change the dates download changes the dates in the following line
OISST_dates <- data.frame(t = seq(as.Date("2019-12-01"), as.Date("2019-12-31"), by = "day"))

# To finish up this step we add some text to those dates so they match the OISST file names
OISST_files <- OISST_dates %>%
  mutate(t_day = gsub("-", "", t),
         t_month = substr(t_day, 1, 6),
         t_year = year(t),
         file_name = paste0(OISST_base_url, t_month, "/", "oisst-avhrr-v02r01.", t_day ,".nc"))
```

Download data

Now that we have a dataframe that contains all of the URLs for the files we want to download we'll create a function that will crawl through those URLs and download the files for us.

```
# This function will go about downloading each day of data as a NetCDF file
# Note that this will download files into a 'data/OISST' folder in the root directory
  # If this folder does not exist it will create it
  # If it does not automatically create the folder it will need to be done manually
  # The folder that is created must be a new folder with no other files in it
  # A possible bug with netCDF files in R is they won't load correctly from
  # existing folders with other file types in them
# This function will also check if the file has been previously downloaded
  # If it has it will not download it again
OISST_url_daily_dl <- function(target_URL){
  dir.create("~/data/OISST", showWarnings = F)
  file_name <- paste0("~/data/OISST/",sapply(strsplit(target_URL, split = "/"), "[[", 10))
  if(!file.exists(file_name)) download.file(url = target_URL, method = "libcurl", destfile =
         file_name)
}
```

```r
# The more cores used, the faster the data may be downloaded
  # It is best practice to not use all of the cores on one's machine
  # The laptop on which I am running this code has 8 cores, so I use 7 here
doParallel::registerDoParallel(cores = 7)

# And with that we are clear for take off
system.time(plyr::l_ply(OISST_files$file_name, .fun = OISST_url_daily_dl, .parallel = T)) # ~15
        seconds

# In roughly 15 seconds a user may have a full month of global data downloaded
# This scales well into years and decades, and is much faster with more cores
# Download speeds will also depend on the speed of the users internet connection
```

Load data

The following code chunk contains the function we may use to load and prepare our OISST data for further use in R.

```r
# This function will load and subset daily data into one data.frame
# Note that the subsetting by lon/lat is done before the data are loaded
  # This means it will use much less RAM and is viable for use on most laptops
  # Assuming one's study area is not too large
OISST_load <- function(file_name, lon1, lon2, lat1, lat2){
      OISST_dat <- tidync(file_name) %>%
        hyper_filter(lon = between(lon, lon1, lon2),
                     lat = between(lat, lat1, lat2)) %>%
        hyper_tibble() %>%
        select(lon, lat, time, sst) %>%
        dplyr::rename(t = time, temp = sst) %>%
        mutate(t = as.Date(t, origin = "1978-01-01"))
      return(OISST_dat)
}

# Locate the files that will be loaded
OISST_files <- dir("~/data/OISST", full.names = T)

# Load the data in parallel
OISST_dat <- plyr::ldply(.data = OISST_files, .fun = OISST_load, .parallel = T,
                         lon1 = 270, lon2 = 320, lat1 = 30, lat2 = 50)

# It should only take a few seconds to load one month of data depending on the size of the lon/lat
        extent chosen
```

In the code chunk above I have chosen the spatial extent of longitude 270 to 320 and latitude 30 to 50. This a window over the Atlantic Coast of North America. One may simply change the lon/lat values above as necessary to match the desired study area. The function also re-labels the ‘time’ column as ‘t’, and the ‘sst’ column as ‘temp’. We do this now so that they match the default column names that are expected for calculating MHWs so we won’t have to do any extra work later on.

Again, please note that trying to load too much data at once may be too much for the RAM on one’s machine. If running the above code causes one’s machine to hang, try loading a smaller subset of data. Or make friends with someone with a server sized machine.

Visualise data

It is always good to visualise data early and often in any workflow. The code pipeline below shows how we can visualise a day of data from those we’ve loaded.

```
OISST_dat %>%
  filter(t == "2019-12-01") %>%
  ggplot(aes(x = lon, y = lat)) +
  geom_tile(aes(fill = temp)) +
  scale_fill_viridis_c() +
  coord_quickmap(expand = F) +
  labs(x = NULL, y = NULL, fill = "SST (°C)") +
  theme(legend.position = "bottom")
```

In the next vignette we will see how to detect MHWs in gridded data.