# How to Open and Work with NetCDF Data in R

NASA Earthdata Webinar

ORNL DAAC https://daac.ornl.gov

December 5, 2023

## Contents



---

This tutorial illustrates how to read spatial data (raster) from netCDF file and import into R for plotting and analysis. The R code is uses the 'ncdf4' package for reading netCDF files, 'terra' package for creating spatial raster objects, and 'ggplot2' for drawing line graphs.

## Install and Load Packages

In addition to the built-in functionality of R, we will use four packages throughout this exercise. Packages are a collection of documentation, functions, and other items that someone has created and compiled for others to use in R. Install the packages, as well as their dependencies, using the function `install.packages()`.

```
install.packages("ggplot2", dependencies = TRUE)
install.packages("ncdf4", dependencies = TRUE)
install.packages("terra", dependencies = TRUE)
```

Reading data stored in netCDF files is quite straightforward in R, provided that you load the appropriate packages. After installing the packages, their libraries need to be loaded to use the functions.

```
library(ggplot2)
library(ncdf4)
library(terra)
```

For package details try `help()` (e.g., `help("ncdf4")`), and to view the necessary arguments of a function try `args()` (e.g., `args(nc_open)`).

## Load Data

First, we need some data to manipulate. As an example, we will use data on trends in vegetation greenness in the Arctic created by Guay et al. (2015) titled "Long-Term Arctic Growing Season NDVI Trends from GIMMS 3g, 1982-2012." Specifically, we will use the file "gimms3g_ndvi_1982-2012.nc4."

This dataset provides normalized difference vegetation index (NDVI) data for the Arctic growing season derived primarily with data from Advanced Very High Resolution Radiometer (AVHRR) sensors on board several NOAA satellites over the years 1982 through 2012. The NDVI data, which show vegetation activity, were averaged annually for the arctic growing season (June, July, and August) in each year. The data are circumpolar in coverage at 8-kilometer resolution and limited to greater than 20 degrees North.

The dataset is available from the ORNL DAAC. An active Earthdata login is needed for download. For new users, an Earthdata account is available from free by completing the registration at https://urs.earthdata.nasa.gov.

Use this link to download the file if needed:

https://daac.ornl.gov/daacdata/global_vegetation/GIMMS3g_NDVI_Trends/data/gimms3g_ndvi_1982-2012.nc4

Be sure to save this *.nc4 file to the current working directory.

## Read the NetCDF File Contents

Use the `nc_open()` function to read the data into a data structure named *nc_data*. Print the metadata about the file to a text file.

```
nc_data <- nc_open("gimms3g_ndvi_1982-2012.nc4")
# print(nc_data)  # uncomment to print summary information for netCDF
```

```
##
File gimms3g_ndvi_1982-2012.nc4 (NC_FORMAT_NETCDF4_CLASSIC):
     2 variables (excluding dimension variables):
        double time_bnds[nv,time]   (Chunking: [2,1])  (Compression: level 4)
            calendar: standard
            units: months since 1982-01-01 00:00:00
        double NDVI[lon,lat,time]   (Chunking: [720,120,1])  (Compression: level 4)
            grid_mapping: crs
            standard_name: normalized_difference_vegetation_index
            long_name: Mean Normalized Difference Vegetation Index in growing season (June, July, and Au
            cell_methods: area: mean time: mean
            _FillValue: -9999
            missing_value: -9999
```

```
     4 dimensions:
        time  Size:31   *** is unlimited ***
            calendar: standard
            standard_name: time
            units: months since 1982-01-01 00:00:00
            bounds: time_bnds
        nv  Size:2 (no dimvar)
        lat  Size:840
            standard_name: latitude
            long_name: latitude
            units: degrees_north
        lon  Size:4320
            standard_name: longitude
            long_name: longitude
            units: degrees_east
    10 global attributes:
        GDAL_AREA_OR_POINT: Area
        GDAL: GDAL 1.10.0, released 2013/04/24
        Conventions: CF-1.6
        title: Mean Normalized Difference Vegetation Index in growing season (June, July, and August)
        source: GIMMGS3g
        contact: Kevin Guay
        institution: Woods Hole Research Center
        email: kguay@whrc.org
        references: Guay, K.C., P.S.A. Beck, L.T. Berner, S.J. Goetz, A. Baccini, and W. Buermann. 2014
        history: Converted to CF-netCDF v4 at Oak Ridge National Labor`
##
```

From this output we see that there are two variables: "time_bnds", which contains the start and end date of each observation, and "NDVI", which is the variable of interest. "NDVI" has three dimensions written as [lon,lat,time].

There are a total of four dimensions in the file: "lat", "lon", "time", and "nv". The latter is used to record the beginning and end of the time range. In our case, we can ignore "nv" and focus on the three dimensions that are used to organize the NDVI data. There are also 10 global attributes which provide metadata information about the file.

We need to capture the values associated with these dimensions. The following code reads the latitudes, longitudes, and times of each NDVI observation, saves them to memory, and prints the length of these vectors.

```
lon <- ncvar_get(nc_data, "lon", verbose = FALSE)   # read lon variable
lat <- ncvar_get(nc_data, "lat", verbose = FALSE)   # read lat variable
t <- ncvar_get(nc_data, "time", verbose = FALSE)   # read t (time) variable
print( c(length(lon), length(lat), length(t)) )    # show vector lengths
```

```
## [1] 4320  840   31
```

Look at a few of the longtitude and latitude values to ensure they make sense.

```
head(lon)  # print the first few entries in the longitude vector
```

```
## [1] -179.9583 -179.8750 -179.7917 -179.7083 -179.6250 -179.5417
```

```
head(lat)  # and the longitude vector
```

```
## [1] 20.04167 20.12500 20.20833 20.29167 20.37500 20.45833
```

Next, read in the data from the variable "NDVI" and verify the dimensions of the array.

```
ndvi.array <- ncvar_get(nc_data, "NDVI")  # store the data in a 3-dimensional array
dim(ndvi.array)  # print the dimensions to the screen
```

```
## [1] 4320  840   31
```

By comparing these dimensions to the *lon*, *lat*, and *t* vector lengths, we see that there are 4320 longitude, 840 latitude, and 31 time values in the NDVI array.

Another pertinent piece of information about the "NDVI" variable is the value used to designate missing data. This code obtains the "_FillValue" attribute from the NDVI variable, the value which indicates missing data.

```
fillvalue <- ncatt_get(nc_data, "NDVI", "_FillValue")
print(fillvalue$value)
```

```
## [1] -9999
```

The fill value is -9999.

We can close the netCDF file now that the data have been read into an R object.

```
nc_close(nc_data)
```

## Working with the Data

Now, we have the entire array of NDVI values for 4320 x 840 grid cells over each of 31 years. What can we do with it?

First, a little housekeeping. Replace all those pesky fill values with the R-standard "NA" to indicate no data.

```
ndvi.array[ndvi.array == fillvalue$value] <- NA
```

Let's get one year of the NDVI data and plot it.

Time is the third dimension of the *ndvi.array*. The first time slice represents the growing season of 1982. Just to make sure everything is working correctly, we can take a look at the dimensions of this time slice.

```
ndvi.slice.1982 <- ndvi.array[, , 1]
dim(ndvi.slice.1982)
```

```
## [1] 4320  840
```

The dimensions should be 4320 longitudes by 840 latitudes. Everything checks out, so we can save the data as a raster.

## Rasterize and Plot

It may be necessary to rearrange the data array. For example, the rows vary by longitude and columns by latitude. In order to display the data correctly on a map, the data need to be transposed so the values are arranged by latitude (rows) by longitude (columns).

```
r.tmp <- t(ndvi.slice.1982)               # transpose to lat (rows), lon (cols)
      # r.tmp is a temporary object to hold the transposed array
```

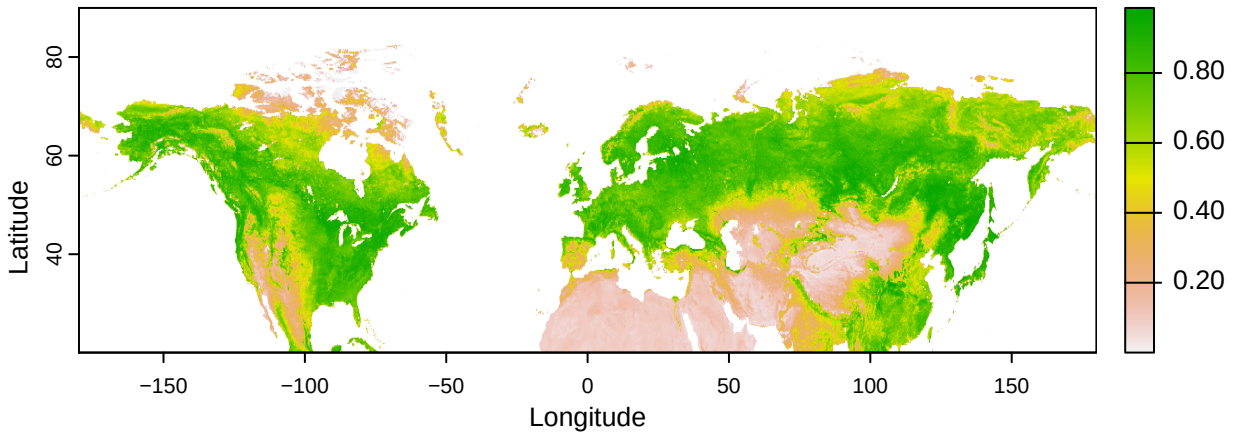
Now, create a SpatialRaster object using the terra package. Note that the W,E,S,N geographic boundaries are specified in the 'extent' portion of the `rast()` function. In addition, the coordinate reference system (CRS) is specified in proj4 format. This dataset uses the common "WGS 84" system (EPSG:4326) for geographic coordinates.

```
r <- rast(r.tmp, extent=ext( min(lon), max(lon), min(lat), max(lat) ),
            crs = "+proj=longlat +ellps=WGS84 +datum=WGS84 +no_defs +towgs84=0,0,0")
rm(r.tmp)        # a bit of clean up
```

Finally, we can plot the raster to take a look at NDVI for 1982.

Most netCDF files record spatial data beginning at the **bottom** left corner, but R stores and plots the data from the **upper** left corner. Therefore, the data must be flipped on the vertical axis to correctly orient it for R.

```
r <- flip(r, direction = "vertical")              # flip on vertical axis
plot(r, xlab="Longitude", ylab="Latitude")      # verify correct orientation
```



Remember that these data are cut-off below 20 degrees North latitude.

For these netCDF data, we needed to transpose and flip the raster to orient the data correctly in R. The best way to figure this out is through trial and error. First plot the spatial raster to check orientation, then transpose and flip as needed.

## Save as a GeoTIFF File

Save the raster as a GeoTIFF file for use by other GIS software.

```
writeRaster(r, filename="GIMMS3g_1982.tif", filetype="GTiff", overwrite = TRUE)
```

## Extract Data at a Study Site

Suppose you want a timeseries of NDVI at a study location, such as the Toolik Lake Field Station in Alaska (Latitude: 68.6275, Longitude: -149.5975).

First, convert the entire 3D array of data to a multilayer SpatialRaster object. It will be necessary to transpose and flip this array as above, but this code is a bit different because we are dealing with three dimensions rather than a 2-D slice. The transpose and flip operations are done before converting to raster object. In the code below, transposing is done using the `aperm()` function (array permutation), and flipping is accomplished by reversing the array's row dimension index.

```
ndvi.x <- aperm(ndvi.array, c(2,1,3))        # transpose the 3D array
y <- dim(ndvi.x)[1]                        # y = number of rows (latitude)
ndvi.f <- ndvi.x[(y:1),,]                    # flip by row index (reversed)
r_br <- rast(ndvi.f, extent=ext(min(lon),max(lon),min(lat),max(lat)),
            crs = "+proj=longlat +ellps=WGS84 +datum=WGS84 +no_defs+ towgs84=0,0,0")
rm(y, ndvi.x, ndvi.f)                        # clean up
```

Extract a timeseries of data at the Toolik Lake study location from the raster brick using the `extract()` function. This function reads the NDVI values for each of the 31 timesteps at the location designated by the lon, lat coordinates. The next four lines create a SpatialVector point object "toolik_pt" with the Toolik coordinates that is passed to `extract()` in the fifth line.

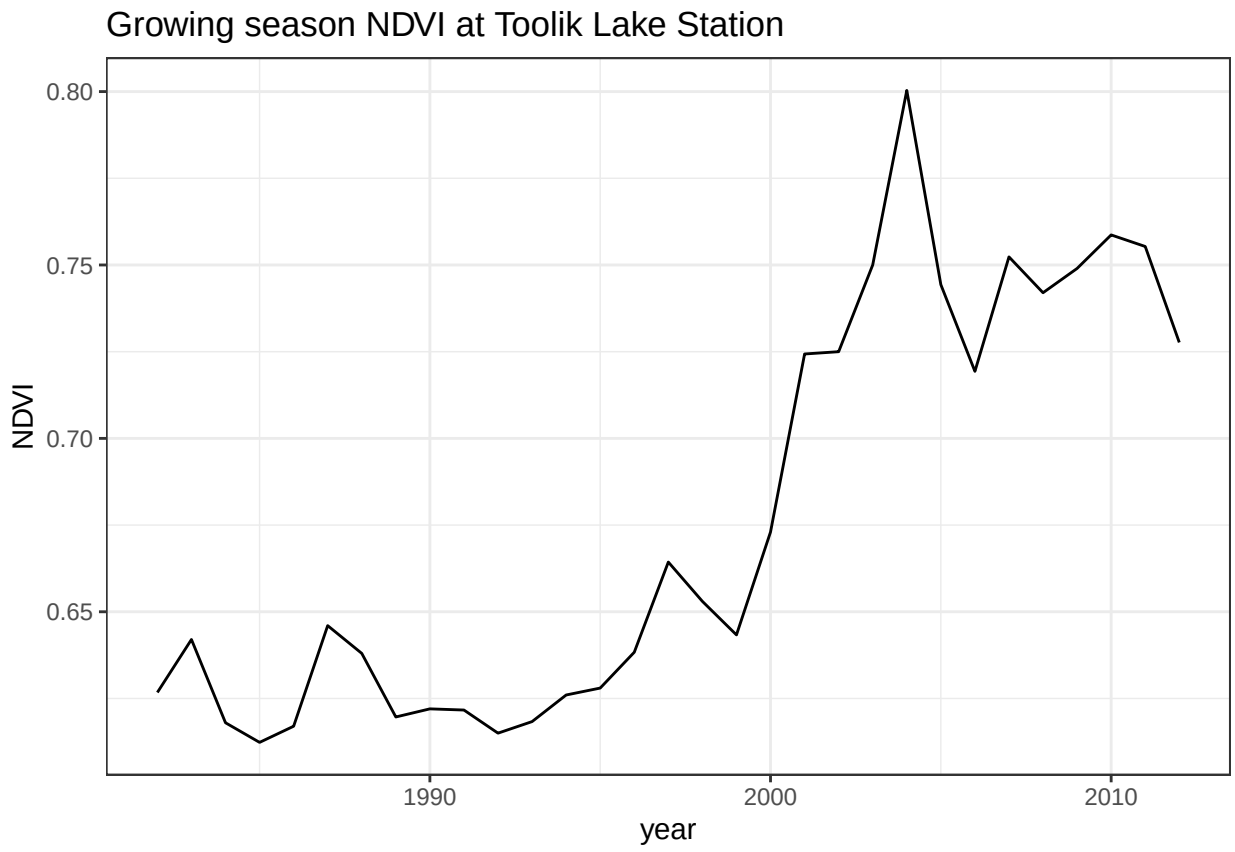
```
toolik_lon <- -149.5975
toolik_lat <- 68.6275
toolik_xy <- matrix( c(toolik_lon, toolik_lat), nrow=1 )            # 1-row matrix
toolik_pt <- vect(toolik_xy, crs="+proj=longlat +datum=WGS84")  # SpatialVector point
toolik_series <- extract(r_br, toolik_pt, ID=FALSE)
colnames(toolik_series) <- 1982:2012
print(format(toolik_series, digits=4))                               # 31 values
```

```
##     1982  1983  1984   1985  1986  1987  1988   1989  1990   1991  1992   1993
## 1 0.6267 0.642 0.618 0.6123 0.617 0.646 0.638 0.6197 0.622 0.6217 0.615 0.6183
##    1994  1995   1996   1997  1998   1999  2000   2001  2002 2003   2004   2005
## 1 0.626 0.628 0.6383 0.6643 0.653 0.6433 0.673 0.7243 0.725 0.75 0.8003 0.7443
##     2006   2007  2008  2009   2010   2011   2012
## 1 0.7193 0.7523 0.742 0.749 0.7587 0.7553 0.7277
```

This timeseries is stored as a simple vector indexed only by the raster layer ID, so let's put it in an easier-to-use dataframe form and then plot the timeseries.

```
toolik_df <- data.frame(year = seq(from = 1982, to = 2012, by = 1), NDVI = t(toolik_series))

ggplot(data = toolik_df, aes(x = year, y = NDVI, group = 1)) +
  geom_line() +  # make this a line plot
  ggtitle("Growing season NDVI at Toolik Lake Station") +
  theme_bw()  # use the black and white theme
```




Wow! The Toolik Lake site really is getting greener over time.

## Difference in NDVI Between Time Periods

The authors of this dataset identified long-term trends in NDVI over the 31 year period from 1982 to 2012. Their paper describes some interesting patterns. Read more about it in:

> Guay, K.C., P.S.A.Beck, L.T. Berner, S.J. Goetz, A. Baccini, and W. Buermann. 2014. Vegetation productivity patterns at high northern latitudes: a multi-sensor satellite data assessment. Global Change Biology 20:3147-3158. https://doi.org/10.1111/gcb.12647

Let's look at the difference in NDVI between 1982 and 2012.

The *ndvi.slice.1982* array has the data from 1982. Let's get the data from 2012, the 31st time slice and call save it to *ndvi.slice.2012*.

```
ndvi.slice.2012 <- ndvi.array[, , 31]
dim(ndvi.slice.2012)  # display dimensions (lon, lat)
```

```
## [1] 4320  840
```

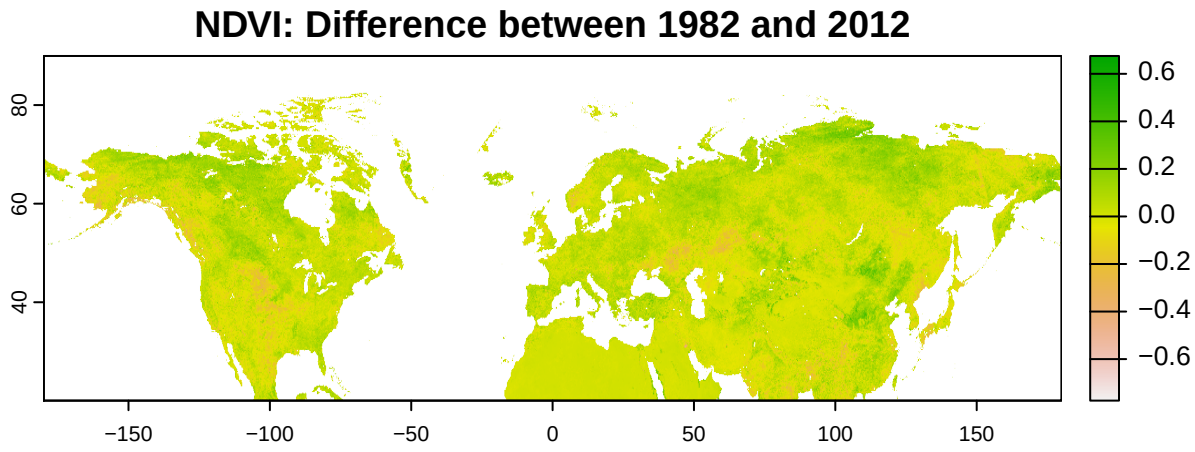
Now take the difference between the slices and save the difference as a raster.

```
ndvi.diff <- ndvi.slice.2012 - ndvi.slice.1982
```

Convert to a SpatialRaster object and plot.

```
ndvi.diff <- t(ndvi.diff)                          # transpose rows & cols
r_diff <- rast(ndvi.diff, extent=ext(min(lon),max(lon),min(lat),max(lat)),
           crs = "+proj=longlat +ellps=WGS84 +datum=WGS84 +no_defs+ towgs84=0,0,0")
r_diff <- flip(r_diff, direction = "vertical")  # flip vertically

plot(r_diff, main="NDVI: Difference between 1982 and 2012")
```




The areas that were greener in 2012 compared to 1982 are represented by positive numbers (green in this color scheme). Note that this is not a proper time series analysis like the authors did in their paper; it only shows the difference between two particular years.

Congratulations, now you know how to open and read data from a netCDF file using R!