OPeNDAP Using Matlab with OPeNDAP

Table of Contents

1. Overview

Who Is This Tutorial For?

This tutorial is for Matlab users who want to access OPeNDAP data services. In the tutorial we will use Matlab's built-in OPeNDAP client to access data from either a server we run just for this purpose or from a server NASA runs as part of its effort to move data into a cloud computing environment.

This tutorial assumes that the reader has a basic grasp of Matlab commands and programming. It also assumes that the reader has a basic grasp of the Matlab netCDF interface.

The NASA data in this document will require that the user configure their client applications to authenticate with the NASA Earthdata Login service. Since the authentication setup is similar for many clients, we have covered it in a separate document

- **Client Authentication Tutorial**

Tutorial Examples Language

The examples are written as Matlab commands. As much as possible, the examples can be copied from the tutorial and run in the Matlab command window.

1.1. Matlab netCDF C details

The Matlab netCDF interface provides a way to read data from both netCDF files on you computer and from remote OPeNDAP servers. The Matlab netCDF interface has two parts: the netCDF C library and the Matlab interface to the netCDF C library. The latter, the Matlab interface to the library is a set of Matlab functions that call the C library and are documented in the Matlab help system. These functions are somewhat easier to use that the netCDF library interface itself, but they are not as flexible as the C library.

In this tutorial we will use both the Matlab netCDF interface and the netCDF library calls.

1.1.1. Environment

You will need Matlab 2020a or later to run the examples in this tutorial.

1.2. The Data

In these examples will use GHRSST Level 3 Sea Surface Temperature data. The URL to the data is given below and in the tutorial. In fact, there are two different URLs to the same data. The first is a URL to a file on a server we run just for tutorial data and testing while the second is a URL to a file on a NASA server running in NASA's Cloud Computing Environment. Both of the URLs use the *dap4://* URL prefix to indicate that the netCDF software should use the DAP4 protocol to access the data. The URL to the OPeNDAP test server can be used without authentication. To use the URL to the NASA Cloud server, you will need to have an Earthdata Login (EDL) account and have configured your client applications to authenticate with EDL.

GHRSST Sub-skin Sea Surface Temperature data, no authentication needed

dap4://test.opendap.org/opendap/hyrax/tutorials/20220812010000-OSISAF-L3C_GHRSST-SSTsubskin-GOES16-ssteqc_goes16_20220812_010000-v02.0-fv01.0.nc

GHRSST Sub-skin Sea Surface Temperature data, NASA Cloud, authentication required

dap4://opendap.earthdata.nasa.gov/collections/C2036877806-POCLOUD/granules/20220812010000-OSISAF-L3C_GHRSST-SSTsubskin-GOES16-ssteqc_goes16_20220812_010000-v02.0-fv01.0

2. Getting Started

First, start Matlab and make sure the dataset is accessible. If you encounter problems at this step, try using the URL in a browser. You will need to replace the *dap4:://* URL prefix with *http://* to make the URL work in a browser.

When the `netcdf.open` function is called, the Matlab netCDF interface will attempt to 'open' the remote dataset. If that works, you will be about to use the `netcdf.inq` function along with the netCDF ID returned by `netcdf.open` to get information about the dataset. These function calls are examples of using the netCDF C library interface to get information.

We do that here and see that this dataset holds 19 variables.

```
>> ghrsst = 'dap4://test.opendap.org/opendap/hyrax/tutorials/20220812010000-OSISAF-
L3C_GHRSST-SSTsubskin-GOES16-ssteqc_goes16_20220812_010000-v02.0-fv01.0.nc'
>> ncid = netcdf.open(ghrsst);
>> [numdims,numvars,numglobalatts,unlimdimid] = netcdf.inq(ncid);
>> numvars
numvars =

    19
```

The ncinfo() function can also be used to retrieve information about the remote dataset. The `ncinfo()` function is a Matlab function that provides a convenient way to access information about a remote dataset. It is a wrapper to the netCDF library calls used in the previous example.

```
% Use the ncinfo function to retrieve information about the remote netCDF file
>> ncinfo_result = ncinfo(ghrsst);
```

In this case, the `ncinfo()` function retrieves information about the remote netCDF file located at the specified URL and stores it in the `ncinfo_result` structure. Information about the Dimensions, Variables, and Global Attributes can all be accessed from the object returned by `ncinfo()`. This same information can also be accessed using the ncid returned by `netcdf.open()`, but `ncinfo()` packages the result conveniently.

```
ncinfo_result =

  struct with fields:

      Filename: 'dap4://test.opendap.org/opendap/hyrax/tutorials/20220812010000-
OSISAF-L3C_GHRSST-SSTsubskin-GOES16-ssteqc_goes16_20220812_010000-v02.0-fv01.0.nc'
          Name: '/'
    Dimensions: [1x3 struct]
     Variables: [1x19 struct]
    Attributes: []
        Groups: []
        Format: 'netcdf4'
     Datatypes: []
```

2.1. Getting more detailed information about the dataset

To see all the values of the fields of an attribute struct, you can loop over the attributes and display their names and values using the Matlab `disp()` function. Here’s an example:

```
% Extract the global attributes from the ncinfo result
>> global_atts = ncinfo_result.Attributes;

% Loop over the attributes and display their names and values
>> disp('Global attributes:')
>> for i = 1:length(global_atts)
    att_name = global_atts(i).Name;
    att_val = global_atts(i).Value;
    disp([att_name, ': ', att_val])
end
```

In this case, we first extract the global attributes from the Attributes field of the `ncinfo_result` structure. We then loop over the attributes using a for loop, and for each attribute, we extract its name and value using the Name and Value fields of the attribute struct. We then display the attribute name and value using the disp function. This will display all the global attributes and their values.

```
Global attributes:
Conventions: CF-1.4
title: Sea Surface Temperature
summary: The L3C product derived from GOES16/ABI brightness temperatures.
references: Geostationary Sea Surface Temperature Product User Manual, http://www.osi-
saf.org
institution: OSISAF
comment:
license: All intellectual property rights of the Ocean & Sea Ice SAF products belong
to EUMETSAT. The use of these products is granted to every user, free of charge. If
users wish to use these products, EUMETSAT's copyright credit must be shown by
displaying the words 'Copyright EUMETSAT' under each of the products shown. EUMETSAT
offers no warranty and accepts no liability in respect of the Ocean & Sea Ice SAF
products. EUMETSAT neither commits to nor guarantees the continuity, availability, or
quality or suitability for any purpose of, the Ocean & Sea Ice SAF products.
id: GOES16-OSISAF-L3C-v1.0
product_id: OSI-207-b
naming_authority: org.ghrsst
product_version: 1.0
gds_version_id: 2.0
file_quality_level: ▯
spatial_resolution: 0.05 degree
northernmost_latitude: <
southernmost_latitude:
easternmost_longitude:
westernmost_longitude:
source: GOES_ABI
platform: GOES16
sensor: GOES_ABI
Metadata_Conventions: Unidata Dataset Discovery v1.0
metadata_link: N/A
keywords: Oceans > Ocean Temperature > Sea Surface Temperature
keywords_vocabulary: NASA Global Change Master Directory (GCMD) Science Keywords
standard_name_vocabulary: NetCDF Climate and Forecast (CF) Metadata Convention
geospatial_lat_units: degrees_north
geospatial_lat_resolution:
geospatial_lon_units: degrees_east
geospatial_lon_resolution:
acknowledgment: In case SAF data (pre-operational or operational) has been used for
the study described in a paper the following sentence would be an appropriate
reference to the funding coming from EUMETSAT: The data from the EUMETSAT Satellite
```

```
Application Facility on Ocean & Sea Ice  used in this study are accessible through the
SAF's homepage http://www.osi-saf.org
creator_name: O&SI SAF
creator_email: osi-saf.helpdesk@meteo.fr
creator_url: http://www.osi-saf.org
project: Group for High Resolution Sea Surface Temperature
publisher_name: The GHRSST Project Office
publisher_url: http://www.ghrsst.org
publisher_email: ghrsst-po@nceo.ac.uk
processing_level: L3C
cdm_data_type: grid
history: METEO-FRANCE GEOSAFO v1.1.8
uuid: DF556788-19E1-11ED-A08A-48DF370DAD10
date_created: 20220812T015542Z
start_time: 20220812T004042Z
time_coverage_start: 20220812T004042Z
stop_time: 20220812T011929Z
time_coverage_end: 20220812T011929Z
netcdf_version_id: 4.6.3
DODS_EXTRA.Unlimited_Dimension: time
```

We would like to look at the names of the variables in this dataset so that we can use netcdf command to read the data values into Matlab.

To display the name, size, and dimensions of each variable in a netCDF file, you can loop over the variables in the Variables field of the ncinfo_result structure and display their names, sizes, and dimensions using the `disp()` function. Here's an example:

```
% Loop over the variables and display their names and sizes
>> disp('Variables:')
>> for i = 1:length(ncinfo_result.Variables)
    var_name = ncinfo_result.Variables(i).Name;
    var_size = ncinfo_result.Variables(i).Size;
    var_dims = ncinfo_result.Variables(i).Dimensions;
    disp([var_name, ': ', mat2str(var_size), ' (', strjoin({var_dims.Name}, ', '),
')'])
end
```

We loop over the variables using a for loop, and for each variable, we extract its name and size using the Name and Size fields of the variable struct. We then display the variable name and size using the `disp()` function. The `mat2str()` function is used to convert the variable size from a numeric array to a string for display, and the `strjoin()` function is used to concatenate the dimension names into a comma-separated string. This will display the names, sizes, and dimensions of all the variables in the netCDF file. This will display the names and sizes of all the variables in the netCDF file.

```
Variables:
time: 1 (time)
```

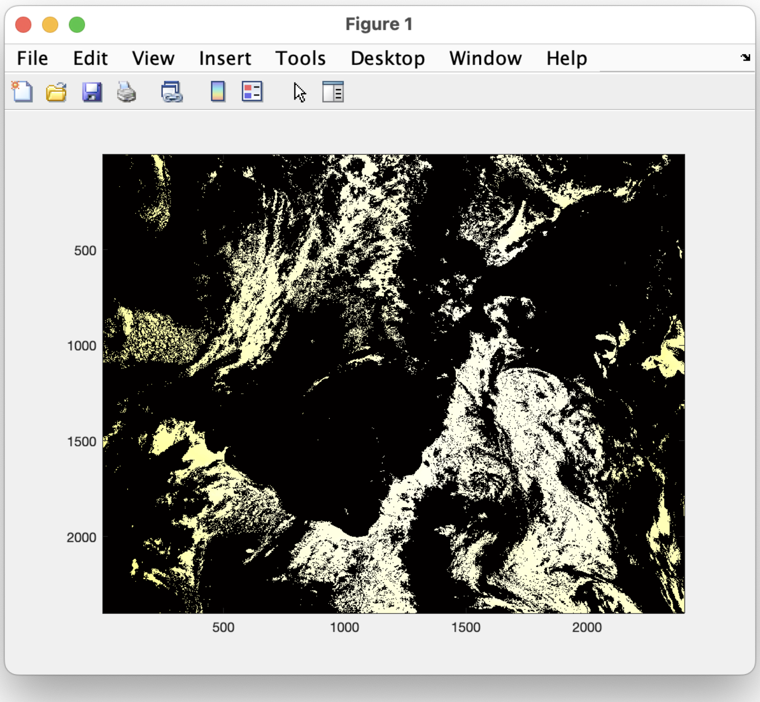
```
lat: 2400 (lat)
lon: 2400 (lon)
sea_surface_temperature: [2400 2400 1] (lon, lat, time)
sst_dtime: [2400 2400 1] (lon, lat, time)
sses_bias: [2400 2400 1] (lon, lat, time)
sses_standard_deviation: [2400 2400 1] (lon, lat, time)
dt_analysis: [2400 2400 1] (lon, lat, time)
wind_speed: [2400 2400 1] (lon, lat, time)
sea_ice_fraction: [2400 2400 1] (lon, lat, time)
aerosol_dynamic_indicator: [2400 2400 1] (lon, lat, time)
adi_dtime_from_sst: [2400 2400 1] (lon, lat, time)
sources_of_adi: [2400 2400 1] (lon, lat, time)
l2p_flags: [2400 2400 1] (lon, lat, time)
quality_level: [2400 2400 1] (lon, lat, time)
satellite_zenith_angle: [2400 2400 1] (lon, lat, time)
solar_zenith_angle: [2400 2400 1] (lon, lat, time)
or_latitude: [2400 2400 1] (lon, lat, time)
or_longitude: [2400 2400 1] (lon, lat, time)
```

2.2. Get a quick look at the data

We might want to get a quick look at the 'sea_surface_temperature' array before going further, so let’s do that. This will be far from 'publication ready,' but given that the array is quite large, it will give us a look at the data.

```
>> sst_varid = netcdf.inqVarID(ncid, 'sea_surface_temperature');
>> sst_data = netcdf.getVar(ncid, sst_varid);
>> imagesc(sst_data);
>> colormap(hot);
```

This shows a plot in a popup window. It’s pretty rough, but we can manipulate the data later.



3. Using the data

To plot the data in a more publication-ready way, we will need to get the vectors that hold the Latitude and Longitude coordinate information for the Sea Surface Temperature data.

To read the data values for the 'lat' and 'lon' variables, we can first ask for their variable IDs and then use those to read the values.

```
>> % Get the IDs of the lat and lon variables
lat_varid = netcdf.inqVarID(ncid, 'lat');
lon_varid = netcdf.inqVarID(ncid, 'lon');

% Read the data for the lat and lon variables
lat_data = netcdf.getVar(ncid, lat_varid);
lon_data = netcdf.getVar(ncid, lon_varid);
```

3.1. Data wrangling

In many cases, data stored in files are not quite in teh form needed for actual use in a toll like Matlab. Transforming the values so they are ready for analysis is often called 'data wrangling'. We will need to do some data wrangling because the data in `sst_data` are neither scaled nor are the missing data values replaced with *NaN*.

To find out how to transform the data, let’s look at the attributes of the dataset and see if there is any information there that will help us.

To get the attributes of a specific variable we can use the ncinfo() function as follows:

```
% Get the variable's attributes
>> varinfo = ncinfo(ghrsst, 'sea_surface_temperature');
>> varinfo
```

The information returned is:

```
varinfo = 

  struct with fields:

        Filename: 'dap4://test.opendap.org/opendap/hyrax/tutorials/20220812010000-
OSISAF-L3C_GHRSST-SSTsubskin-GOES16-ssteqc_goes16_20220812_010000-v02.0-fv01.0.nc'
            Name: 'sea_surface_temperature'
      Dimensions: [1×3 struct]
            Size: [2400 2400 1]
        Datatype: 'int16'
      Attributes: [1×12 struct]
       ChunkSize: []
       FillValue: 'disable'
    DeflateLevel: []
         Shuffle: 0
          Format: 'netcdf4'
```

As before with the dataset's global attributes, loop over the attributes and display their names and values. This version of the loop is slightly more complex because some attributes are strings and some are numeric. The ischar() function is used along with fprintf() to display the values correctly (note that in the format string used with fprintf(), the %g format specifier is used to display numeric values, e.g.).

```
>> disp('sea_surface_temperature attributes:')
>> for i = 1:length(varinfo.Attributes)
    attr = varinfo.Attributes(i);
    name = attr.Name;
    value = attr.Value;

    if ischar(value)
        fprintf('%s = ''%s''\n', name, value);
    else
        fprintf('%s = %g\n', name, value);
    end
end
```

The output shows the numerical values correctly:

```
_FillValue = -32768
```

```
long_name = NaN
standard_name = NaN
units = NaN
add_offset = 273.15
scale_factor = 0.01
valid_min = -300
valid_max = 4500
depth = NaN
source = NaN
comment = NaN
_edu.ucar.maps = NaN
/lat = NaN
```

The variable attributes *_FillValue*, *add_offset*, and *scale_factor* indicate how the values will need to be modified to get the correct values.

```
% Get the scale factor and add offset
scale_factor = varinfo.Attributes(strcmp({varinfo.Attributes.Name},'scale_factor'))
.Value;
add_offset = varinfo.Attributes(strcmp({varinfo.Attributes.Name},'add_offset')).Value;

% Get the fill value
fill_value =
varinfo.Attributes(strcmp({varinfo.Attributes.Name},'_FillValue')).Value
```

Before we go further, lets look at those values:

```
scale_factor =

    0.0100

>> add_offset

add_offset =

  273.1500

>> fill_value

fill_value =

  int16

   -32768

>> sst_data(1:150:2400, 1:150:2400)

ans =
  16x16 int16 matrix

...
```

The sst_data array in an Int16 array, but we would like an array of double values. Once we have that, we can replace the fill_value cells with NaN and scale the data.

```
% Convert to double
>> data = double(sst_data);

% Set fill values to NaN
>> fv_mask = data == fill_value;
>> data(fv_mask) = NaN;

% Look at a sub-sample of the values
>> data(1:150:2400, 1:150:2400)

ans =

  16x16 double matrix

...

>> data = data * scale_factor + add_offset;
>> data(1:150:2400, 1:150:2400)

ans =

       NaN       NaN       NaN       NaN       NaN       NaN       NaN
       299.2100  297.1900       NaN       NaN  296.9300       NaN
       NaN       NaN       NaN

       NaN       NaN       NaN       NaN       NaN       NaN
       NaN  299.4000  297.0700       NaN       NaN       NaN  294.8500
       NaN       NaN       NaN
```

3.2. Plotting

The data values are rotated 90 degrees (because netCDF uses C notion of row-major order but Matlab uses column-major order).

```
% Use the apostrophe (') operator to transpose the data
>> data_t = data';
```

The data store negative latitude at the top and positive at the bottom - we need those flipped for a north-up plot.

```
>> imagesc(lon_mesh(1,:), flip(lat_mesh(:,1)), data_t);
```

However, that leaves the Y-axis labels still inverted; use this 'set' command to flip tha Y-axis labels.

```
>> set(gca,'YTickLabel',flip(get(gca,'YTickLabel')));
% Add x and y axis labels
>> xlabel('Longitude');
>> ylabel('Latitude');
```

Set the range of the color bar and the colormap

```
>> caxis([270, 310]);

% Try out various color maps
>> colormap(hot);
>> colormap(cool);
>> colormap(parula);
>> colorbar;
```

Here’s the plot. It lacks a coastline because we want to show access without requiring any of the optional Matlab packages that provide coastlines. However, the North and South American continents are clearly visible. It *is* possible to download the coastlines and plot them, but that is beyond the scope of this tutorial.

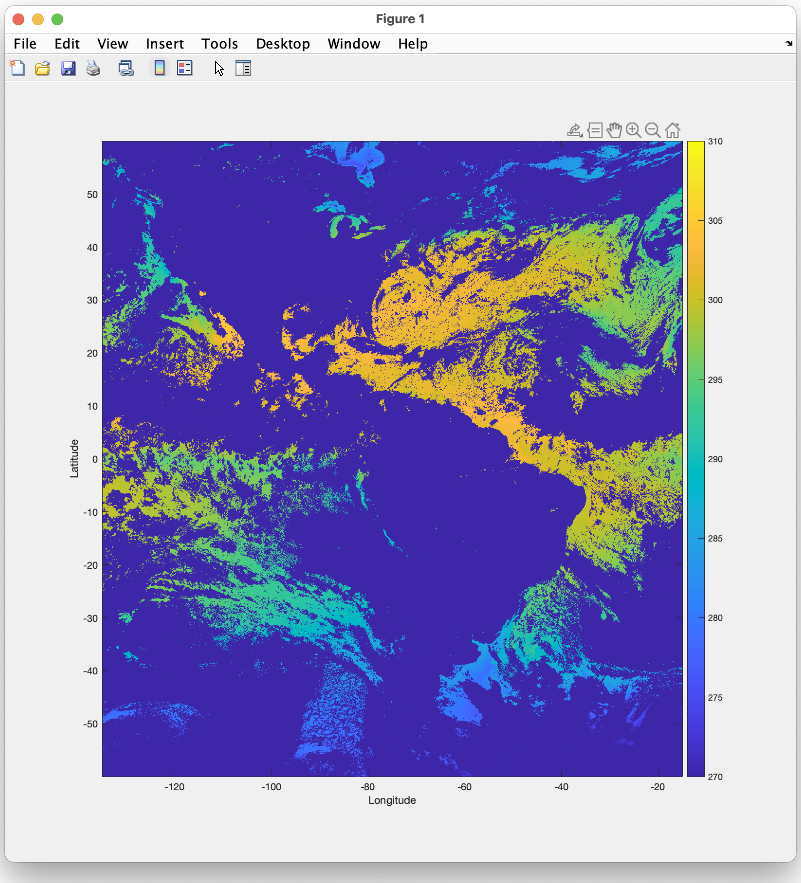


Figure 1
File Edit View Insert Tools Desktop Window Help
Latitude
Longitude
50
40
30
20
10
0
-10
-20
-30
-40
-50
-120
-100
-80
-60
-40
-20
310
305
300
295
290
285
280
275
270

fini