ECNU IMAPP WORKSHOP LAB FIVE: MODIS AOD Analysis

Liam.Gumley@ssec.wisc.edu, June 6, 2011

Objective

The objective of this laboratory is to analyze 12 months of Aqua MODIS aerosol optical depth (AOD) retrievals to create a plot of AOD vs. time over the city of Shanghai. You will use the MODIS AOD retrievals created by the Direct Broadcast Processing System at ECNU for input data. IDL will be used to:

1. read the MODIS AOD product files,
2. create global grids of AOD data,
3. plot a time series of AOD over Shanghai.

By completing this laboratory you will learn basic IDL programming skills for data analysis and visualization.

MODIS data and IDL environment

This lab requires the Aqua MODIS Aerosol Optical Depth product files (*.mod04.hdf) from 2010145 to 2011149 that were created on the dbps server at ECNU. A compressed archive of these files is available at

http://dbps.ecnu.edu.cn/data/workshop/day5/data/data.zip

You will also need the IDL program files available at

http://dbps.ecnu.edu.cn/data/workshop/day5/idl/idl.zip

Note: These files have already been downloaded and installed on your computer for the laboratory session.

The following preferences should by selecting **Windows > Preferences > IDL**:

Startup file: **E:\Workshop\Day5\idl\idl_startup.pro**

Initial working directory: **E:\Workshop\Day5\work**

1. Read the AOD product files and create daily data files

The first step in analyzing the MODIS AOD data is to read all the MOD04 product files. Since there are multiple passes from the satellite each day, there are often multiple product files for each day in the time period 2010145 to 2011149.

You will need to read each product file, extract the valid data, and save it to one output file for each day.

Start by creating a new IDL program named read_modis.pro in the IDL Editor. First, get the list of files to be analyzed by entering the following lines:

```
PRO READ_MODIS

COMPILE_OPT IDL2

directory = '../products/'
product = '*.mod04.hdf'

;- Get list of input files
list = file_search(directory + product)
help, list

END
```

Notes:

1. An IDL program begins with a PRO statement and ends with an END statement.
2. COMPILE_OPT IDL2 sets IDL compiler options (recommended in every IDL program).
3. String variables (text variables) in IDL are enclosed in single or double quotes.
4. Comments in IDL start with a semi-colon.
5. The FILE_SEARCH function returns a list of files matching a pattern.
6. The HELP program prints the name, type, and size of a variable.

Save the program by typing Control-S, then compile and run the program by typing the following lines in the command window:

```
IDL> .compile read_modis
IDL> read_modis
```

You should see the following output:

```
LIST            STRING    = Array[587]
```

BUT WHAT IF I MADE A MISTAKE!

Let's say you entered the file search pattern in your program without single quotes as shown below:

```
product = *.mod04.hdf
```

When you compile the program, you will get the following error:

```
IDL> .compile read_modis

product = *.mod04.hdf
```

```
                      ^
% Syntax error.
  At: e:\workshop\day5\work\read_modis.pro, Line 4
% 1 Compilation error(s) in module READ_MODIS.
```

This means you need to fix the program on line 4. When you have fixed the offending line, you should return to the main program level by typing

```
RETALL
```

at the command line, and then compile your program again. You should also type RETALL if your program crashes (i.e. stops unexpectedly) during execution.

IDL now has a list of 587 file names in an array named LIST. The next step is to print the name of each file. Edit your program to include the highlighted lines shown below:

```
PRO READ_MODIS

COMPILE_OPT IDL2

directory = '../products/'
product = '*.mod04.hdf'

;- Get list of input files
list = file_search(directory + product)
help, list

;- Loop over input files
for i = 0, n_elements(list) - 1 do begin

  ;- Get file name
  input_file = list[i]
  print, input_file

endfor

END
```

Notes:

1. The up and down arrow keys can be used to recall the command history.
2. A loop begins with a FOR statement and ends with an ENDFOR statement.
3. The N_ELEMENTS function returns the number of elements in a variable.
4. Elements of an array are accessed in form array[index], where the index is a zero-based scalar or vector.

Save, compile, and run the program. The names of all the files should be printed (one name per line). Partial output is shown below:

```
a1.11148.0550.mod04.hdf
a1.11149.0454.mod04.hdf
a1.11149.0635.mod04.hdf
```

Now you will read the AOD data from each file (the files are in HDF4 format). Edit your program to include the highlighted lines shown below:

```
PRO READ_MODIS

COMPILE_OPT IDL2

directory = '../products/'
product = '*.mod04.hdf'
sdsname = 'Optical_Depth_Land_And_Ocean'
scale = 0.001

;- Get list of input files
list = file_search(directory + product)
help, list

;- Loop over input files
for i = 0, n_elements(list) - 1 do begin

  ;- Get file name
  input_file = list[i]
  print, input_file

  ;- Read the input file
  hdfid = hdf_sd_start(input_file)
  hdf_sd_varread, hdfid, 'Latitude', lat
  hdf_sd_varread, hdfid, 'Longitude', lon
  hdf_sd_varread, hdfid, sdsname, data
  hdf_sd_end, hdfid
  help, lat, lon, data

  ;- Apply scale factor
  data = data * scale

endfor

END
```

Notes:

1. The MODIS AOD product is stored in a 'Scientific Data Set' named 'Optical_Depth_Land_And_Ocean' in each MOD04 HDF file.
2. The HDF_SD_START function opens a HDF file and returns a file identifier.
3. The HDF_SD_VARREAD function reads a scientific data set (SDS) variable from an open HDF file.
4. The HDF_SD_END function closes a HDF file.
5. The AOD values in the HDF file are stored as scaled integers (x1000) to save space.

Save, compile, and run the program. The name of each file, along with the name, size, and type of the LAT, LON, and DATA arrays for each file will be printed. Partial output is shown below:

```
../products/a1.11148.0550.mod04.hdf
LAT             FLOAT     = Array[135, 425]
LON             FLOAT     = Array[135, 425]
DATA            INT       = Array[135, 425]
../products/a1.11149.0454.mod04.hdf
LAT             FLOAT     = Array[135, 465]
LON             FLOAT     = Array[135, 465]
DATA            INT       = Array[135, 465]
../products/a1.11149.0635.mod04.hdf
LAT             FLOAT     = Array[135, 221]
LON             FLOAT     = Array[135, 221]
DATA            INT       = Array[135, 221]
```

The next step is to find the data values that are valid (i.e., not missing values). In the AOD retrievals, the missing value is -9.999, so you will search for data values that are greater than zero. Then you will extract the valid latitude, longitude, and AOD data values, and store them in a table.

Edit your program to include the highlighted lines shown below:

```
PRO READ_MODIS

COMPILE_OPT IDL2

directory = '../products/'
product = '*.mod04.hdf'
sdsname = 'Optical_Depth_Land_And_Ocean'
scale = 0.001

;- Get list of input files
list = file_search(directory + product)
help, list

;- Loop over input files
for i = 0, n_elements(list) - 1 do begin

  ;- Get input file name
  input_file = list[i]
  print, input_file

  ;- Read the input file
  hdfid = hdf_sd_start(input_file)
  hdf_sd_varread, hdfid, 'Latitude', lat
  hdf_sd_varread, hdfid, 'Longitude', lon
  hdf_sd_varread, hdfid, sdsname, data
  hdf_sd_end, hdfid

  ;- Apply scale factor
  data = data * scale

  ;- Find valid data values
  loc = where(data gt 0.0, count)
  print, count

  ;- Save valid data values
  if (count gt 0) then begin
```

```
    table = fltarr(3, count)
    table[0, *] = lat[loc]
    table[1, *] = lon[loc]
    table[2, *] = data[loc]
    help, table
  endif

endfor

END
```

Notes:

1. The WHERE function returns an array of locations where a given expression is true. In this case, it returns the locations (or indices) in DATA where the data values are greater than zero. The COUNT argument returns the number of array indices.
2. The code within the IF statement is executed if the given expression is true.
3. The FLTARR function creates a new array, with dimensions [3, COUNT], or 3 columns and COUNT rows. The array values are set to zero.
4. The array named TABLE is then filled with the valid data values; latitude in the first column; longitude in the second column; and AOD in the third column.

Save, compile, and run the program. The name of each input file, the number of valid data values, and the name, type, and size of the output table will be printed. Partial output is shown below:

```
../products/a1.11148.0550.mod04.hdf
       13874
TABLE           FLOAT     = Array[3, 13874]
../products/a1.11149.0454.mod04.hdf
        9241
TABLE           FLOAT     = Array[3, 9241]
../products/a1.11149.0635.mod04.hdf
        3645
TABLE           FLOAT     = Array[3, 3645]
```

The final task for this program is to write the output table of latitude, longitude, and AOD values to an output file. A new output file will be created for each day (e.g., 2011147). If there is more than one input file for a day, the data from each input file will be written in sequence to the output file for that day.

Edit your program to include the highlighted lines shown below:

```
PRO READ_MODIS

COMPILE_OPT IDL2

directory = '../products/'
product = '*.mod04.hdf'
sdsname = 'Optical_Depth_Land_And_Ocean'
scale = 0.001
```

```
;- Get list of input files
list = file_search(directory + product)
help, list

;- Set old file date
old_date = 'a1.99001'

;- Get output file unit
get_lun, lun

;- Loop over input files
for i = 0, n_elements(list) - 1 do begin

  ;- Get input file name
  input_file = list[i]
  print, input_file

  ;- Read the input file
  hdfid = hdf_sd_start(input_file)
  hdf_sd_varread, hdfid, 'Latitude', lat
  hdf_sd_varread, hdfid, 'Longitude', lon
  hdf_sd_varread, hdfid, sdsname, data
  hdf_sd_end, hdfid

  ;- Apply scale factor
  data = data * scale

  ;- Get file date (e.g., 'a1.11210')
  file_name = basename(input_file)
  date_length = strlen(old_date)
  new_date = strmid(file_name, 0, date_length)

  ;- Find valid data values
  loc = where(data gt 0.0, count)

  ;- Save valid data values
  if (count gt 0) then begin

    ;- Open a new output file if needed
    if (new_date ne old_date) then begin
      free_lun, lun
      output_file = new_date + '.dat'
      openw, lun, output_file, /get_lun
      old_date = new_date
      print, ' '
      print, 'Opened new output file', output_file
    endif

    ;- Save the valid data values in a table (lat, lon, data)
    table = fltarr(3, count)
    table[0, *] = lat[loc]
    table[1, *] = lon[loc]
    table[2, *] = data[loc]

    ;- Write the output file in binary format
    print, 'Writing data from ', input_file
    writeu, lun, table

  endif

endfor

END
```

Notes:

1. The OLD_DATE variable stores the date for the last output file. It is initialized to a value of 'a1.99001' so that a new output file will be opened for the first AOD product table.
2. The GET_LUN procedure assigns a logical unit number to be used for writing the output files.
3. The date for each input file is obtained from the input file name. The BASENAME function returns the filename without the path information. The STRLEN function returns the length of the OLD_DATE string variable. The STRMID function returns the first 8 characters from the file name, starting with the first character.
4. The NEW_DATE variable is compared with the OLD_DATE variable, and if it does not match, a new output file must be opened. The FREE_LUN procedure closes the current output file. The OPENW procedure opens the new output file. The OLD_DATE variable is then set to the value of the NEW_DATE variable.
5. The WRITEU procedure writes the table of latitude, longitude, and AOD values to the output file. If more than one input file is found for a day, then the output tables are written sequentially (in order) to the output file.

Save, compile, and run the program. Now the program prints a message very time a new output file is opened. Partial output is shown below:

```
../products/a1.11148.0413.mod04.hdf
       12157
Opened new output file a1.11148.dat
TABLE           FLOAT     = Array[3, 12157]
../products/a1.11148.0550.mod04.hdf
       13874
TABLE           FLOAT     = Array[3, 13874]
../products/a1.11149.0454.mod04.hdf
        9241
Opened new output file a1.11149.dat
TABLE           FLOAT     = Array[3, 9241]
../products/a1.11149.0635.mod04.hdf
        3645
TABLE           FLOAT     = Array[3, 3645]
```

You should find that several hundred output files, with names like a1.11148.dat, have been created in the current directory. The next task is to transform these AOD observations into daily global gridded averages.

2. Read the daily data files and create daily grid average files

The next program you write will read the AOD data files, and create a global average of AOD values on a 1x1 degree equal angle grid.

Create a new IDL program named grid_modis.pro in the IDL Editor, and enter the following lines of code that will (a) get the list of files to be read, and (b) open and read the contents of each data file:

```
PRO GRID_MODIS

COMPILE_OPT IDL2

;- Get list of input files
list = file_search('a1.*.dat')
help, list

;- Loop over input files
for index = 0, n_elements(list) - 1 do begin

  ;- Open input file and get number of rows
  input_file = list[index]
  print, input_file
  openr, lun, input_file, /get_lun
  result = fstat(lun)
  nrows = result.size / (4 * 3)

  ;- Read data table
  table = fltarr(3, nrows)
  readu, lun, table
  free_lun, lun

  ;- Extract, lat, lon, data values
  lat = table[0, *]
  lon = table[1, *]
  data = table[2, *]
  help, lat, lon, data

endfor

END
```

Notes:

1. The list of input file names is returned by the FILE_SEARCH function.
2. A loop over the list of input files is started with a FOR statement.
3. The size of each data file is not known in advance, so each file is opened with the OPENR procedure, and file information is returned by the FSTAT function. The variable RESULT.SIZE contains the size of the file in bytes. To convert this size to the number of rows in the file, you divide by the number of bytes per floating-point number (4) multiplied by the number of columns in the file (3). The variable NROWS then contains the number of rows in the file.

4. A variable named TABLE is created to hold the data, and the data is read from the input file by the READU procedure. The input file is closed by the FREE_LUN procedure.
5. The values latitude, longitude, and AOD are extracted from the table into individual arrays named LAT, LON, and DATA, respectively.

Now save, compile, and run the program. The name of each data file, and the name, type, and size of the LAT, LON, and DATA arrays will be printed. Partial output is shown below:

```
a1.11147.dat
LAT             FLOAT     = Array[1, 9487]
LON             FLOAT     = Array[1, 9487]
DATA            FLOAT     = Array[1, 9487]
a1.11148.dat
LAT             FLOAT     = Array[1, 26031]
LON             FLOAT     = Array[1, 26031]
DATA            FLOAT     = Array[1, 26031]
a1.11149.dat
LAT             FLOAT     = Array[1, 12886]
LON             FLOAT     = Array[1, 12886]
DATA            FLOAT     = Array[1, 12886]
```

Now you will create a global equal angle grid containing average values of AOD. The grid will be at 1x1 degree resolution, and will have 360 columns (for 360 degrees of longitude from 180W to 180E) and 180 rows (for 180 degrees of latitude from 90S to 90N).

A key concept in the design of this program is the use of one-dimensional indices to access the elements of a two-dimensional array. Arrays in IDL are stored in memory in column-major order (i.e., column 1 row 1, column 2 row 1, column 3 row 1, etc.), and the elements of the array can be accessed using 1D indices. For example, if you had an array in IDL with 5 columns and 4 rows, then array index 16 would point to the second column on the fourth row, as shown in the figure below.

Columns

Rows	0	1	2	3	4
	5	6	7	8	9
	10	11	12	13	14
	15	**16**	17	18	19

This gives you a convenient way to access the locations of the cells in the global grid. Now edit your program to include the highlighted lines shown below:

```
PRO GRID_MODIS

COMPILE_OPT IDL2

;- Get list of input files
list = file_search('a1.*.dat')
```

```
help, list

;- Create equal angle grid
;- (lower left corner of grid is at 180W, 90S, i.e. -180.0, -90.0)
resolution = 1.0
ncol = long(360.0 / resolution)
nrow = long(180.0 / resolution)
grid = fltarr(ncol, nrow)

;- Loop over input files
for index = 0, n_elements(list) - 1 do begin

  ;- Open input file and get number of rows
  input_file = list[index]
  print, input_file
  openr, lun, input_file, /get_lun
  result = fstat(lun)
  nrows = result.size / (4 * 3)

  ;- Read data table
  table = fltarr(3, nrows)
  readu, lun, table
  free_lun, lun

  ;- Extract, lat, lon, data values
  lat = table[0, *]
  lon = table[1, *]
  data = table[2, *]
  help, lat, lon, data

  ;- Get list of grid indices
  lonlat_to_index, lon, lat, resolution, grid_index

  ;- Reset the grid array
  grid[*] = -999.9

  ;- Loop over grid cells
  for cell_index = 0, n_elements(grid) - 1 do begin

    ;- Find the data points within this grid cell
    loc = where(grid_index eq cell_index, count)

    ;- Save the mean for this cell
    if (count gt 0) then begin
      mean = total(data[loc]) / count
      grid[cell_index] = mean
      print, cell_index, mean
    endif

  endfor

endfor

END
```

Notes:

1. The number of columns and rows in the grid is computed from the variable named RESOLUTION, which is the grid resolution in degrees. Then the GRID array is created to hold the average AOD values.
2. The one-dimensional indices (or array locations) of the MODIS AOD values in the global grid are computed by the LONLAT_TO_INDEX procedure, and are stored in the array named GRID_INDEX.
3. The contents of the GRID array are set to -999.9 as a missing value flag.
4. A loop over all the cells in the global grid is started with a FOR statement.
5. For each grid cell, the WHERE function returns an array named LOC which contains the locations in the GRID_INDEX array where the value matches the index of the current grid cell. This is how you find the AOD retrieval locations within each grid cell. The number of matches is returned in the COUNT variable (it will be zero if there are no matches).
6. If AOD values are found within the current grid cell, then the mean AOD value is computed from the sum of the AOD values (returned by the TOTAL function) divided by the number of values. The mean value (or average) is stored in the MEAN variable.
7. The mean value for the grid cell is then stored in the GRID array.

Save the changes to your program, then compile and run it. Partial output is shown below:

```
a1.10146.dat
LAT             FLOAT     = Array[1, 8177]
LON             FLOAT     = Array[1, 8177]
DATA            FLOAT     = Array[1, 8177]
       36303     0.434750
       36304     0.408000
       36663     0.411667
       36664     0.278000
       36665     0.178722
```

Note that this program will now take several minutes to run, as it processes each of the daily data files. After you have seen several of the daily files processed successfully, you can stop the program by typing Control-C. Note that you will need to enter RETALL at the command line before you compile or run the program again.

Now that you have created a global grid for each day, you will need to save the grids to disk for further analysis. It would also be helpful to visualize the grids to make sure they look correct. Add the highlighted lines shown below to your program:

```
PRO GRID_MODIS

COMPILE_OPT IDL2

;- Get list of input files
list = file_search('a1.*.dat')
help, list
```

```
;- Create equal angle grid
;- (lower left corner of grid is at 180W, 90S, i.e. -180.0, -90.0)
resolution = 1.0
ncol = long(360.0 / resolution)
nrow = long(180.0 / resolution)
grid = fltarr(ncol, nrow)

;- Loop over input files
for index = 0, n_elements(list) - 1 do begin

  ;- Open input file and get number of rows
  input_file = list[index]
  print, input_file
  openr, lun, input_file, /get_lun
  result = fstat(lun)
  nrows = result.size / (4 * 3)

  ;- Read data table
  table = fltarr(3, nrows)
  readu, lun, table
  free_lun, lun

  ;- Extract, lat, lon, data values
  lat = table[0, *]
  lon = table[1, *]
  data = table[2, *]
  help, lat, lon, data

  ;- Get list of grid indices
  lonlat_to_index, lon, lat, resolution, grid_index

  ;- Reset the grid array
  grid[*] = -999.9

  ;- Loop over grid cells
  for cell_index = 0, n_elements(grid) - 1 do begin

    ;- Find the data points within this grid cell
    loc = where(grid_index eq cell_index, count)

    ;- Save the mean for this cell
    if (count gt 0) then begin
      mean = total(data[loc]) / count
      grid[cell_index] = mean
      print, cell_index, mean
    endif

  endfor

  ;- Display the grid
  map_set, 30, 130, /orthographic, /isotropic, title=input_file
  image = map_image(grid, x0, y0, compress=1)
  loadct, 39, /silent
  tv, bytscl(image, min=0.0, max=5.0), x0, y0
  map_continents

  ;- Save the output grid
  output_file = strmid(input_file, 0, 8) + '.grd'
  print, 'Writing ' + output_file
  openw, lun, output_file, /get_lun
  writeu, lun, grid
  free_lun, lun

endfor

END
```

Notes:

1. The MAP_SET procedure creates a global orthographic map projection, centered at 30N, 130E.
2. The MAP_IMAGE function is used to create a 2D image array which has been resampled from the global grid to the orthographic map projection.
3. A rainbow color table is loaded by the LOADCT procedure.
4. The resampled image is displayed by the TV procedure, and is scaled over the range 0-5 by the BYTSCL function.
5. Continental boundaries are plotted by the MAP_CONTINENTS procedure.
6. A new output file is created for each day, with a name similar to a1.10154.grd. The OPENW procedure is used to open each file, and the WRITEU procedure is used to write the final in binary format. Each output file contains a 360x180 array (360 columns and 180 rows) of 32-bit floating point data, representing MODIS AOD averages. The missing value -999.9 is used for grid cells where no AOD values are present.

You will find that several hundred global grid files, with names like a1.10154.grd, have been created in your IDL work directory. The last programming task is to create a time averaged plot of AOD over Shanghai.

3. Read the daily grid average files and create time series plot

Now it is time to read the daily grid average AOD data files and create a time series plot at a specific location.

Create a new IDL program named plot_modis.pro in the IDL Editor, and enter the following lines of code that will (a) create a grid array, (b) get the list of grid files to be read, (c) open and read the contents of each grid file, and (d) extract that AOD average value at the specified latitude and longitude:

```
PRO PLOT_MODIS, LAT, LON

COMPILE_OPT IDL2

;- Create equal angle grid
;- (lower left corner of grid is at 180W, 90S, i.e. -180.0, -90.0)
resolution = 1.0
ncol = long(360.0 / resolution)
nrow = long(180.0 / resolution)
```

```
grid = fltarr(ncol, nrow)

;- Get list of grid files
list = file_search('a1*.grd')
help, list

;- Create output arrays
data = fltarr(n_elements(list))
time = fltarr(n_elements(list))

;- Compute index of the required grid cell
lonlat_to_index, lon, lat, resolution, index
help, index

;- Loop over grid files
for i = 0, n_elements(list) - 1 do begin

  ;- Read this file
  input_file = list[i]
  openr, lun, input_file, /get_lun
  readu, lun, grid
  free_lun, lun

  ;- Get data for the desired grid cell
  data[i] = grid[index]

endfor

help, data

END
```

Notes:

1. The PROGRAM statement now lists two arguments named LAT and LON for this procedure. This will allow you to enter the desired latitude and longitude values on the command line.
2. A global grid array named GRID is created, given the grid resolution defined by the variable names RESOLUTION.
3. A list of input files is returned by the FILE_SEARCH function in an array named LIST.
4. Output arrays for data values and the corresponding time series values are created by the FLTARR function, named DATA and TIME, respectively.
5. The grid index corresponding to the specified latitude and longitude is computed by the LONLAT_TO_INDEX procedure, and stored in a variable named INDEX.
6. A loop over all the input grid files is started via a FOR statement.
7. Each input file is opened by calling the READ procedure, and the global grid array is read by the READU procedure. The input file is closed by calling the FREE_LUN procedure.
8. The AOD average value at the desired location is extracted, and stored in the array named DATA.

Save the program, and then compile and run it as shown below (31.23N, 121.47E is the location of Shanghai):

```
IDL> plot_modis, 31.23, 121.47
```

The output from the program should appear as shown below:

```
LIST            STRING    = Array[323]
INDEX           LONG      =        43861
DATA            FLOAT     = Array[323]
```

Since you will be creating a time series plot, you will need to get the date for each input file and convert it to a numeric value, which can then be used by the IDL plotting procedure. Add the highlighted lines shown below to your program:

```
PRO PLOT_MODIS, LAT, LON

COMPILE_OPT IDL2

;- Create equal angle grid
;- (lower left corner of grid is at 180W, 90S, i.e. -180.0, -90.0)
resolution = 1.0
ncol = long(360.0 / resolution)
nrow = long(180.0 / resolution)
grid = fltarr(ncol, nrow)

;- Get list of grid files
list = file_search('a1*.grd')
help, list

;- Create output arrays
data = fltarr(n_elements(list))
time = fltarr(n_elements(list))

;- Compute index of the required grid cell
lonlat_to_index, lon, lat, resolution, index
help, index

;- Loop over grid files
for i = 0, n_elements(list) - 1 do begin

  ;- Read this file
  input_file = list[i]
  openr, lun, input_file, /get_lun
  readu, lun, grid
  free_lun, lun

  ;- Get data for the desired grid cell
  data[i] = grid[index]

  ;- Get year and day of year from filename
  year = long(strmid(input_file, 3, 2)) + 2000
  day_of_year = long(strmid(input_file, 5, 3))

  ;- Get month and day of month
  ydn2md, year, day_of_year, month, day

  ;- Get Julian date
  time[i] = julday(month, day, year)

endfor

help, data
help, time

END
```

Notes:

1. The year and day of year (e.g., 2011, 120) are derived from the name of each grid average file. The STRMID function extracts the last two digits of the year, and the day of year, from the variable INPUT_FILE. The LONG function converts the string data type to a 32-bit integer.
2. The YDN2MD function computes the month and day of month, given the year and day of year.
3. The JULDAY function computes the Julian day from the month, day, and year. This is necessary because IDL uses Julian day values to plot time data.

Save your program, then compile and run it. Now the program will print the name, type, and size of the TIME variable, as shown below:

```
IDL> plot_modis, 31.23, 121.47
LIST            STRING    = Array[323]
INDEX           LONG      =        43861
DATA            FLOAT     = Array[323]
TIME            FLOAT     = Array[323]
```

The final step in the AOD analysis is to create a 16-day average of the global gridded data, and then plot it as a time series. Add the highlighted lines shown below to your program:

```
PRO PLOT_MODIS, LAT, LON

COMPILE_OPT IDL2

;- Create equal angle grid
;- (lower left corner of grid is at 180W, 90S, i.e. -180.0, -90.0)
resolution = 1.0
ncol = long(360.0 / resolution)
nrow = long(180.0 / resolution)
grid = fltarr(ncol, nrow)

;- Get list of grid files
list = file_search('a1*.grd')
help, list

;- Create output arrays
data = fltarr(n_elements(list))
time = fltarr(n_elements(list))
```

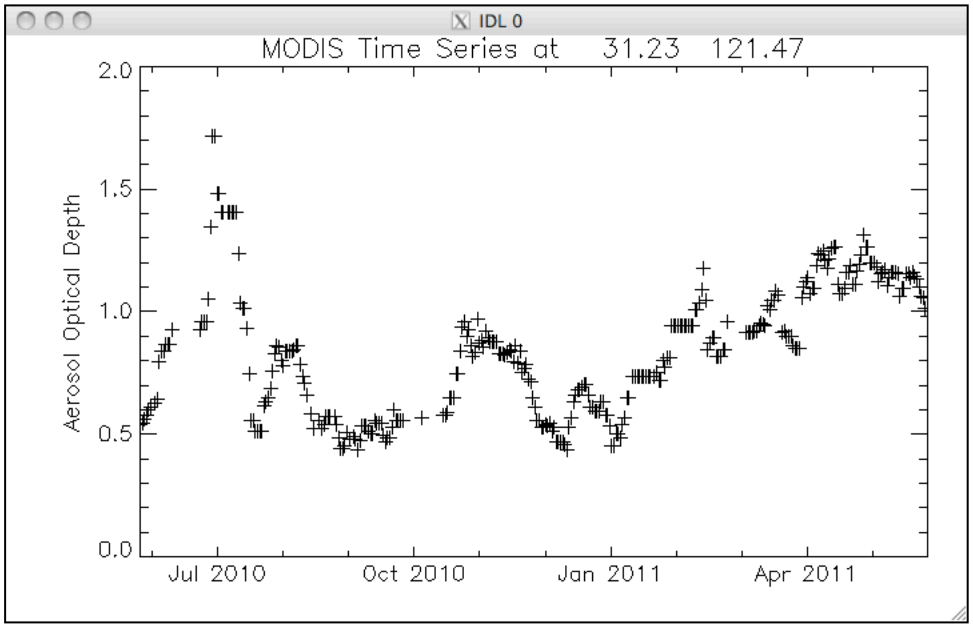
```
;- Compute index of the required grid cell
lonlat_to_index, lon, lat, resolution, index
help, index

;- Loop over grid files
for i = 0, n_elements(list) - 1 do begin

  ;- Read this file
  input_file = list[i]
  openr, lun, input_file, /get_lun
  readu, lun, grid
  free_lun, lun

  ;- Get data for the desired grid cell
  data[i] = grid[index]

  ;- Get year and day of year from filename
  year = long(strmid(input_file, 3, 2)) + 2000
  day_of_year = long(strmid(input_file, 5, 3))

  ;- Get month and day of month
  ydn2md, year, day_of_year, month, day

  ;- Get Julian date
  time[i] = julday(month, day, year)

endfor

help, data
help, time

;- Set missing values to NaN (not a number)
loc = where(data lt 0.0, count)
if (count gt 0) then data[loc] = !VALUES.F_NAN

;- Compute time series, smoothed over 16 days
smoothed_data = smooth(data, 16, /nan, /edge_truncate)

;- Load greyscale color table
loadct, 0

;- Plot the data
result = label_date(date_format=['%M %Y'])
plot, time, smoothed_data, $
  xrange=[min(time), max(time)], $
  xstyle=1, $
  psym=1, $
  color=0, background=255, $
  xtickunits='months', $
  xtickformat='label_date', $
  ytitle='Aerosol Optical Depth', $
  title='MODIS Time Series at' + string(lat, lon, format='(2f8.2)')

END
```

Notes:

1. Missing data values in the gridded AOD data for are replaced with a special floating point value known as 'Not A Number', or NaN. These values will be filtered out in the remainder of the analysis when the IDL smoothing and plotting tools are called.
2. A smoothed time series over 16 days (moving window) is computed by the SMOOTH function.
3. A grey scale color table is loaded by calling LOADCT.
4. The formatting desired for the date labels on the time series x-axis are established by calling the LABEL_DATE function.
5. The smoothed time series is displayed by calling the PLOT procedure, with keyword meanings as follows:
    a. XRANGE and XSTYLE set the x-axis range to the precise range of the time series data,
    b. PSYM sets the plotting symbol to a cross,
    c. COLOR and BACKGROUND set the plotting and background colors to black and white, respectively,
    d. XTICKUNITS and XTICKFORMAT are used to tell the PLOT procedure to display x-axis labels as months,
    e. YTITLE and TITLE set the titles for the y-axis and the plot, respectively.

Save the program, then compile and run it as shown. You should see a time series plot of MODIS AOD for the city of Shanghai, like the figure below:

```
IDL> plot_modis, 31.23, 121.47
```



Try creating similar plots for Beijing (39.90N, 116.40E) and Seoul (37.56N, 126.97E).

END OF ECNU LAB FIVE