Python for multi-year GOCI ocean color products analysis: sharing the advantages and issues

Breakout I. Ocean source scientific computing tools and resources

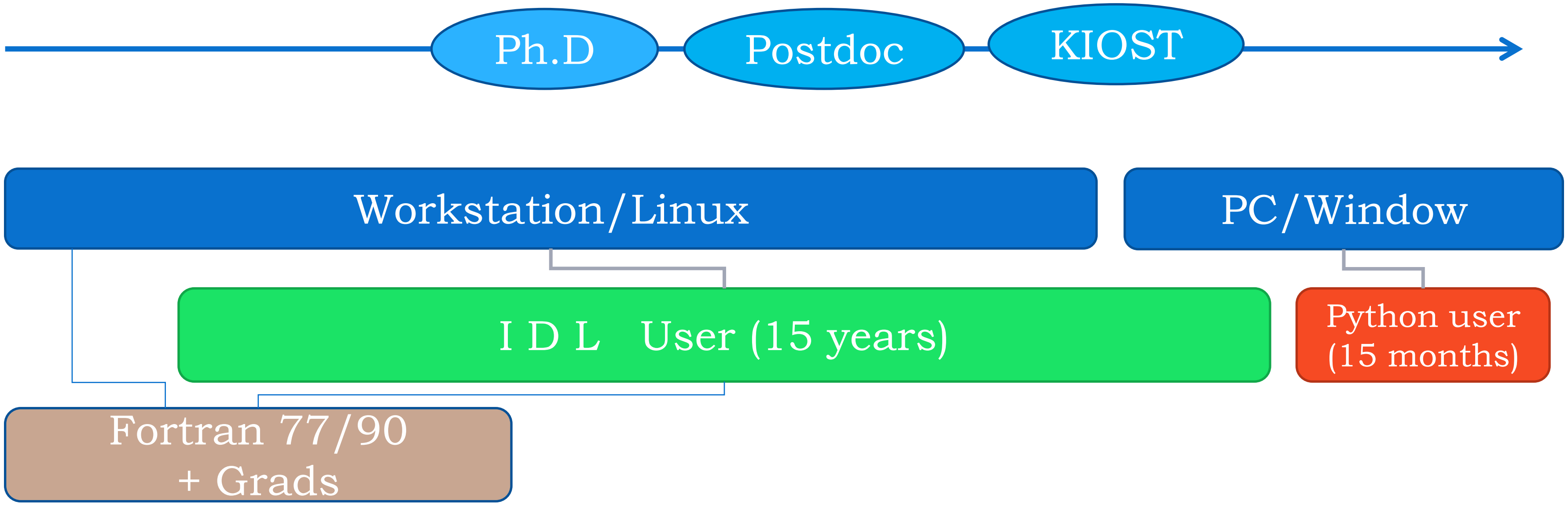
Myung-Sook Park and Seonju Lee

Korean Ocean Satellite Center(KOSC)
Korea Institute of Ocean Science and Technology(KIOST)

Collaboration with Jaehyun Ahn, Youngje Park, Wonkook Kim, and Sunju Lee

History of my scientific tools

I was major in atmospheric science (particularly, remote sensing and climate) and now currently working in ocean remote sensing in KOSC/KIOST.

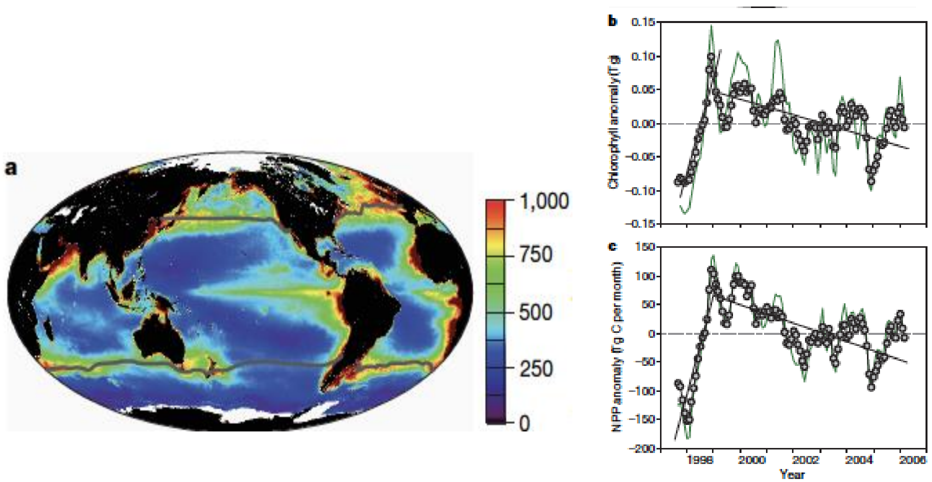


- Only a short-time user of Python but highly satisfied with this tool.
- no longer use IDL

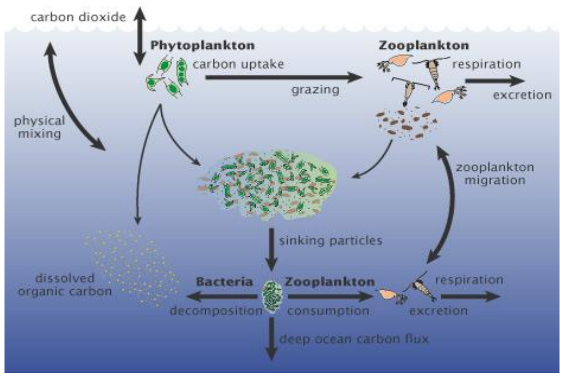
I will present how and why the community is using the tool.

Research background

Establishing the dataset of multi-years ocean color record over global and regional seas is a fundamental for climate-ocean interaction.



Marine phytoplankton is tiny but ubiquitous over ocean surface of Earth



CO_2 are fixed into organic material by phytoplankton.
-- organic carbon is transformed to marine ecosystem by sinking and grazing

Motivation

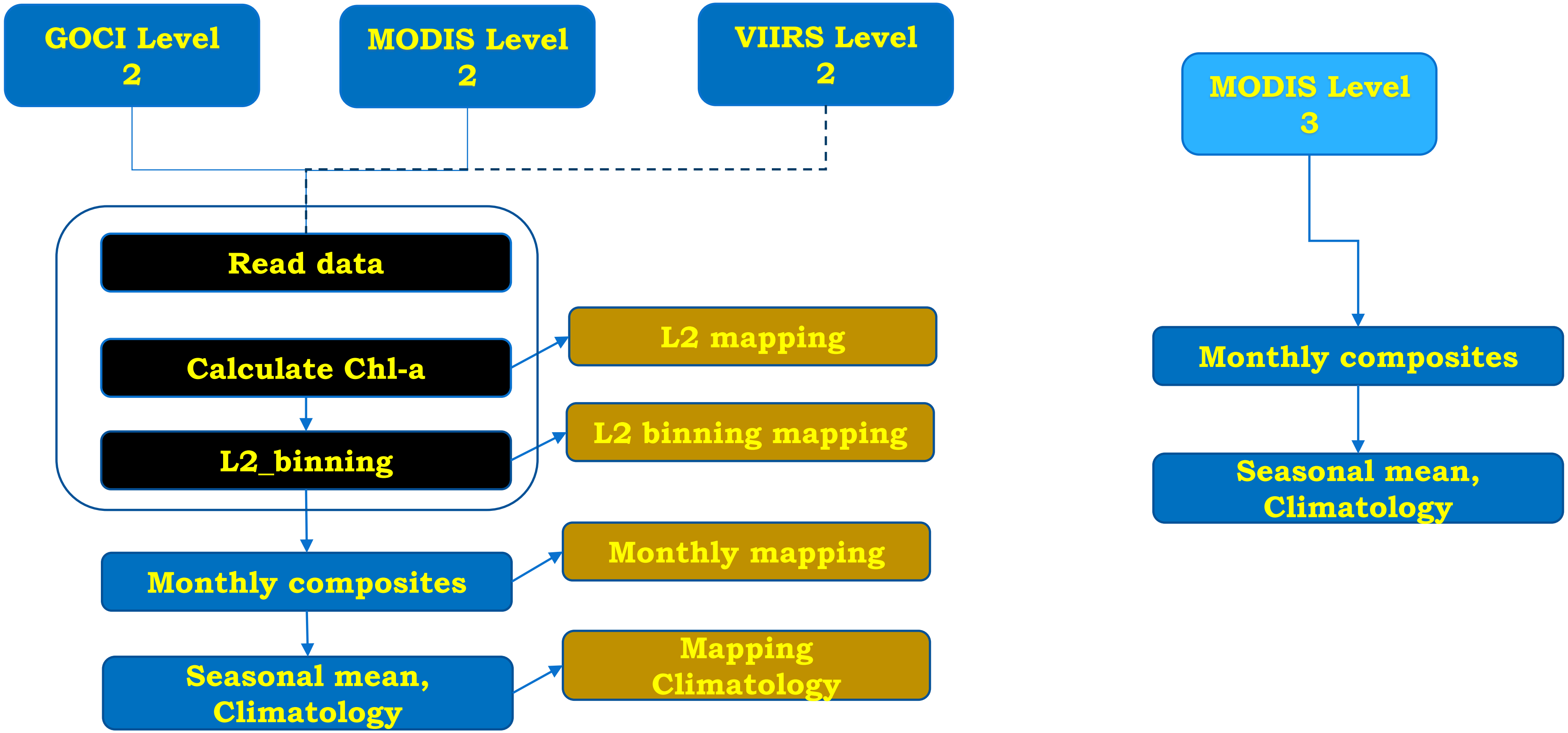
- The first ocean color sensor on a geostationary orbit, Geostationary Ocean Color Imager (GOCI) has been operated for more than eight years since 2010.
- The GOCI has a capability to examine hourly ocean color variability over seas between Korea, China, and Japan of which area is about 2500 km x 2500 km.
- Recently, the version 2 GOCI ocean color products with improved atmospheric correction method (Ahn et al. 2016) was newly released by Korean Ocean Satellite Center (KOSC).
- The validation of the primary ocean color product, GOCI Chrolopyll-a (Chl-a) concentration, has been persistently performed using *in situ* observation (Kim et al. 2016).
- It is time for next step to perform GOCI and multi-satellite ocean color products comparison to clarify GOCI's consistency with NASA heritage ocean color satellites.

Research objectives

- The objectives of the preliminary study are **to investigate the consistency of Chl-a products between GOCI and MODIS for the North-East Asian region, and to examine how to explain the differences between two satellite products.**

- For preparation of multi-decadal records of GOCI and GOCI-II for their climate application, **we need to conduct these kinds of multi-satellite consistency tests (GOCI vs. polar orbiting satellites, GOCI vs. GOCI-II).**

"GOCI and OC satellite Inter-comparison system" --Python 3.6

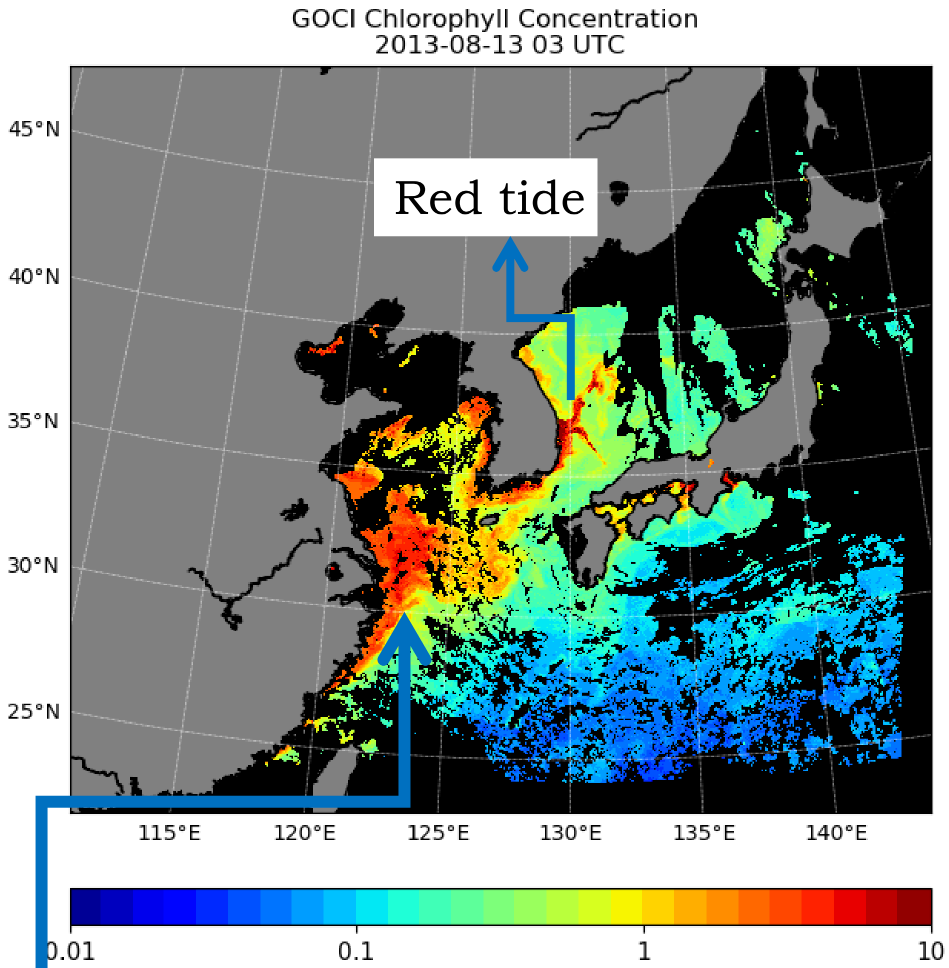


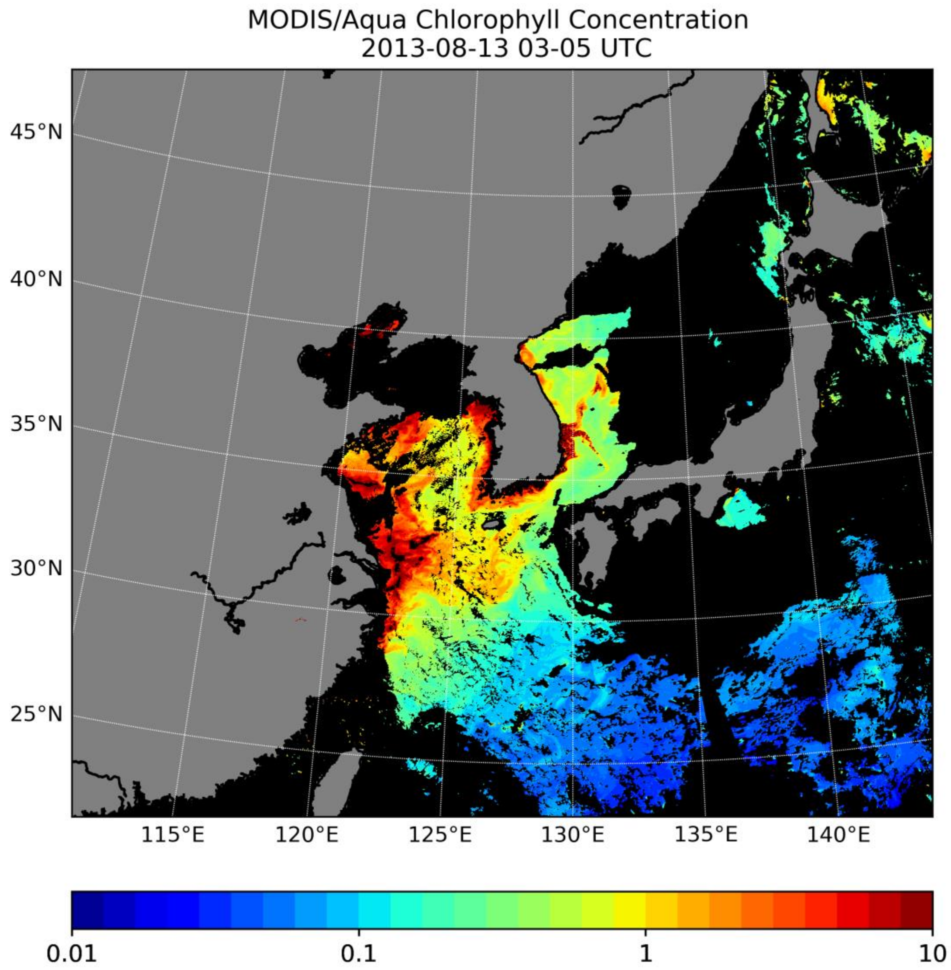
- Grid characteristics
  - Equal-angle binning (CZCS) --> Equal-area binning (2 km)
- Product temporal averages
  - Un-weighted, arithmetic, averages of pigment concentration for all pixels containing valid data

Guide to the Creation and Use of Ocean-Colour, Level-3, Binned Data Products, IOCCG Report Number 4, 2004, Edited by: David Antoine

Research results

Comparison between GOCI and MODIS

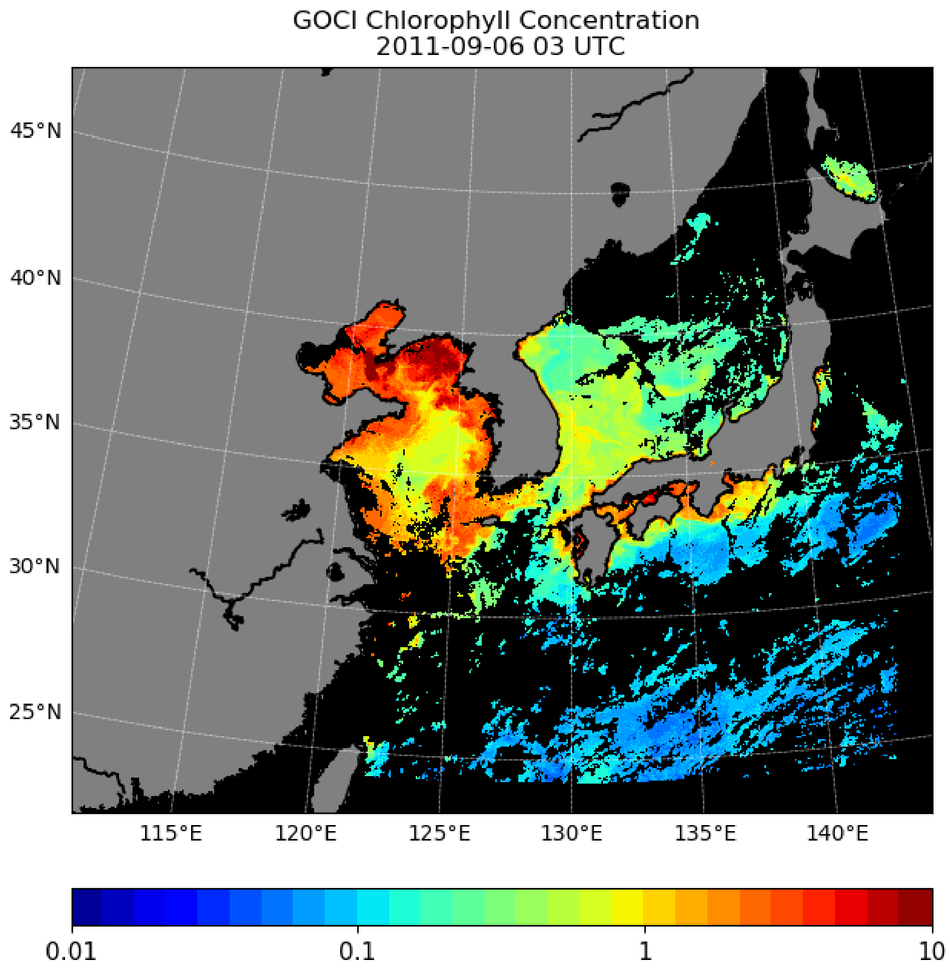


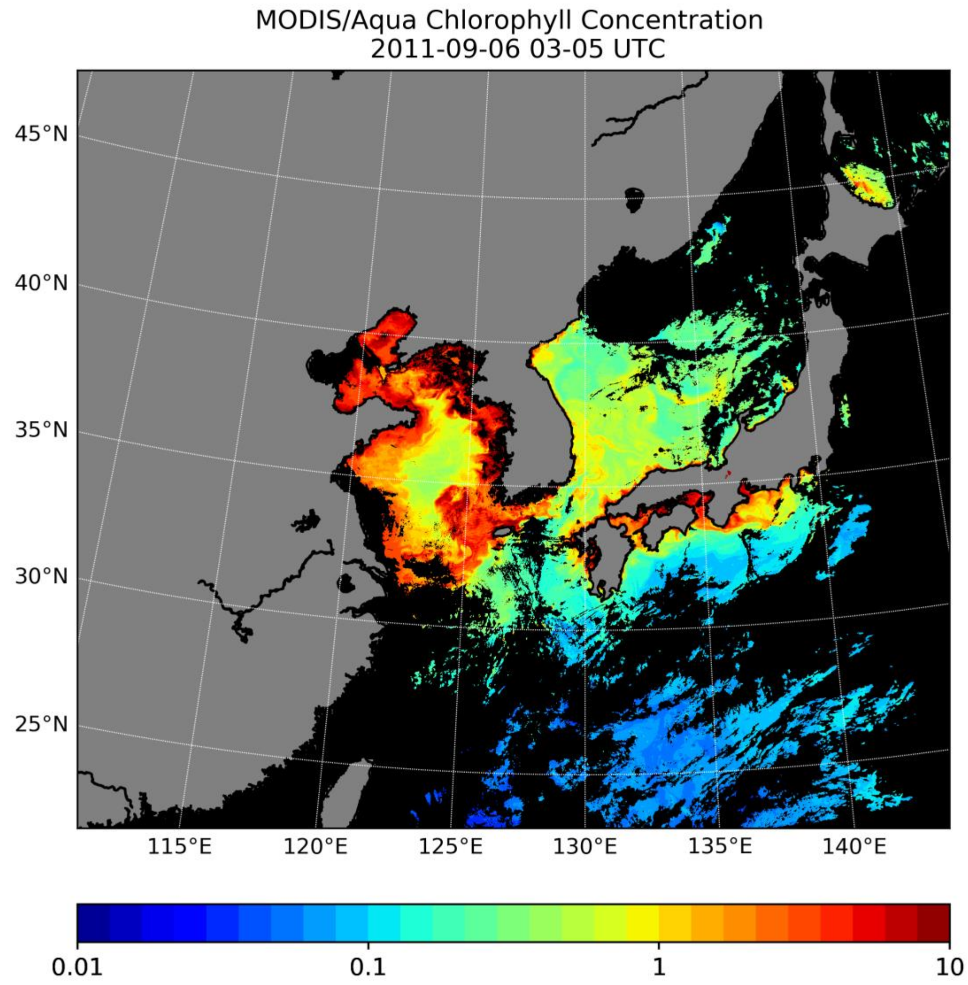


turbid sediment dominated waters associated with Yangtze River freshwater discharge

Comparison between GOCI and MODIS

L2 mapping

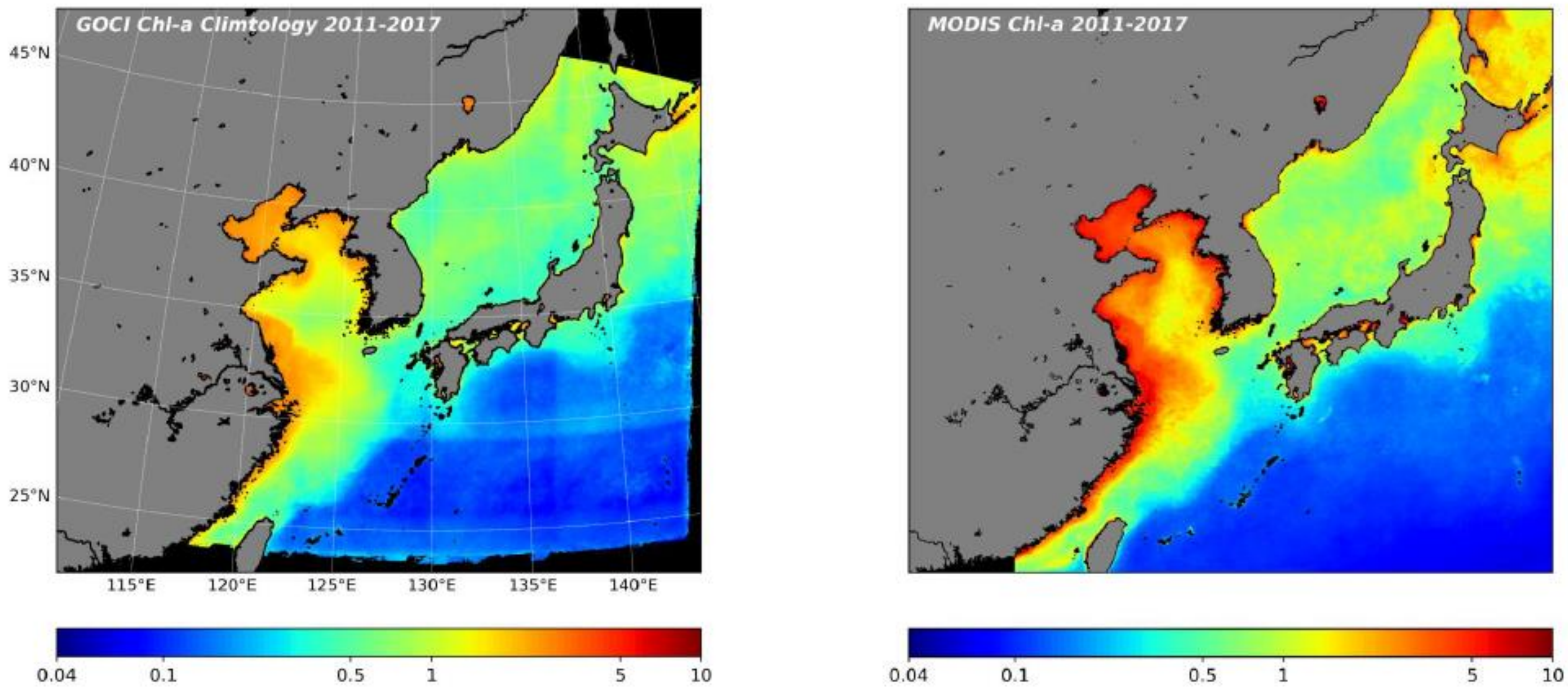




a reasonable agreement between GOCI and MODIS Chl-a distributions is notable, but some differences in the magnitudes of Chl-a are also found.

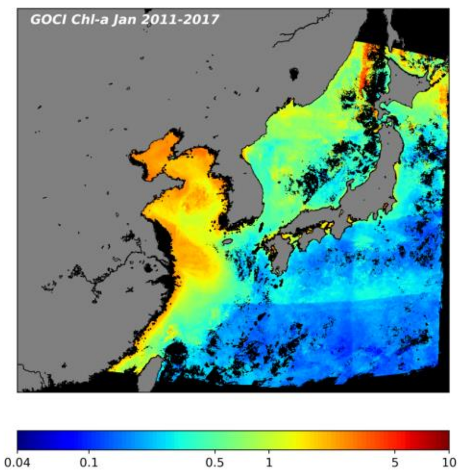
GOCI Climatology (7 years)

L2_binning → **Monthly composites** → **Seasonal mean, Climatology**

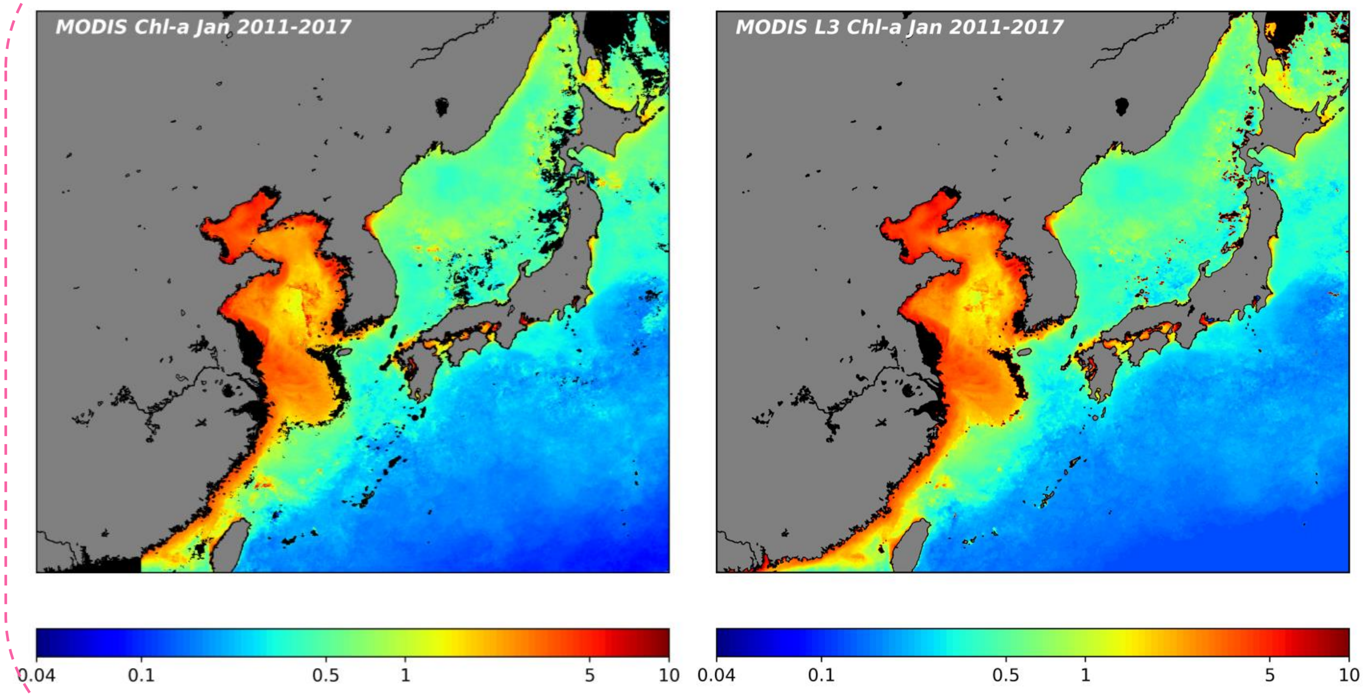


- Significant overestimation of MODIS Chl-a values in comparison to GOCI is obvious along the coastal sea over the Yellow Sea.
- In this turbid ocean environment, uncertainties in ocean color remote sensing remain both satellites dues to issues by atmospheric correction and Chl-a algorithm.

Validating our methodology

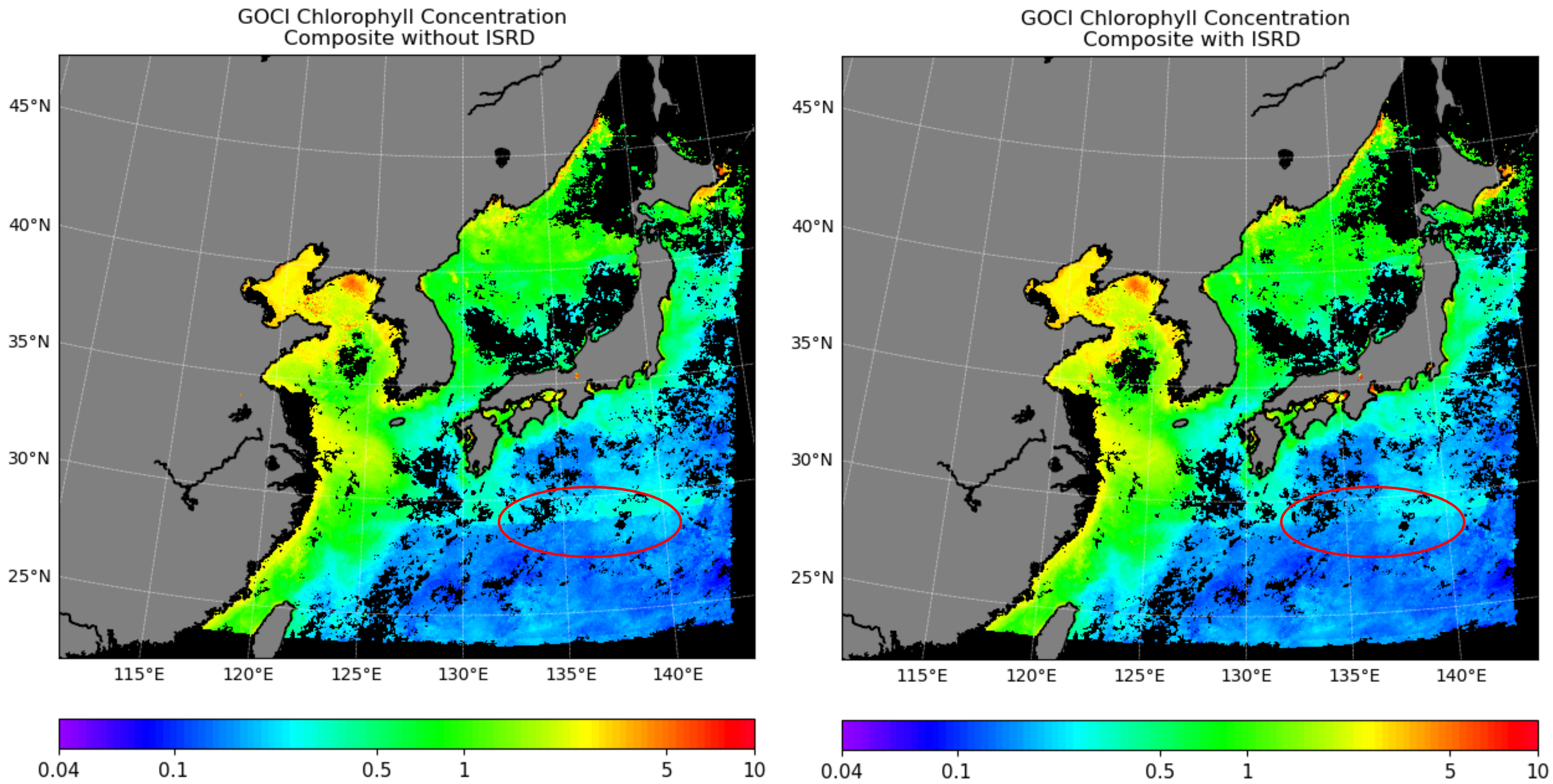


Monthly data calculated from MODIS L2 versus L3



✓ "GOCI-OC satellite inter-comparison system" using MODIS L2 properly reproduces MODIS L3, which confirming the reliability of our calculation.

ISRD correction effect (Monthly mean, Dec 2013)



Kim, W., J.-H. Ahn, Y.-J. Park (2015). "Correction of Stray-Light-Driven Interslot Radiometric Discrepancy (ISRD) Present in Radiometric Products of Geostationary Ocean Color Imager (GOCI)." IEEE Transactions on Geoscience and Remote Sensing 53(10): 5458-5472.

About open-source (python) programming

L2_binning, Chl-a Calculation, and Visualization

Spyder (Python 3.6)

File Edit Search Source Run Debug Consoles Projects Tools View Help

Editor - [redacted]

0_create_goci_bin_nc_daily_v2019_0403.py*

```python
# This program reads GOCI Rrs and Flag information
# to calculate chl_a concentration
# map projection

import sys
sys.path.insert(0, '[redacted]')
from matplotlib.colors import LogNorm
from math import floor
import numpy as np
from pyproj import Proj
from mpl_toolkits.basemap import Basemap, cm
from pylab import figure, savefig
import matplotlib as mp
import matplotlib.pyplot as plt
import mpl_toolkits
import datetime
import matplotlib.colors as colors
from affine import Affine
from netCDF4 import Dataset

# import GOCI_py_lib
import os
from f_read_goci import *
from c_read_lonlat import n_lat, n_lon
from f_flag_all import flag
import shutil
import time
import tracemalloc
```

Modules	Usage
sys	accessing to some variables used or maintained by the interpreter and to functions that interact strongly with the interpreter.
Numpy	fundamental package for scientific computing with Python
Proj.pyproj	Performs cartographic transformations between geographic (lat/lon) and map projection (x/y) coordinates
Datetime	The datetime module supplies classes for manipulating dates and times in both simple and complex ways.
Dataset.netCDF4	Open, read, and write a Netcdf file
Shutil	File copy and removing

L2_binning

```python
proj_id = 'laea'
datum = 'WGS84'
lat_0 = '35.0'
lon_0 = '130.0'

resolution = 2000.

## convert GOCI lon/lat array to cartesian coordinates
area_dict = dict(datum=datum,lat_0=lat_0,lon_0=lon_0,x_1=0, y_1 =0,proj=proj_id,units='m')
prj=Proj(area_dict)
```

- Conversion of GOCI longitude/latitude arrays to [Cartesian coordinate system] coordinate (in km)

(35N, 130E) → (0, 0)

```
>>> n_lon
array([[ 111.32442474,  111.33106232,  111.33769989, ...,  148.66227722,
         148.66891479,  148.67556763],
       [ 111.3260498 ,  111.33268738,  111.33931732, ...,  148.66067505,
         148.66729736,  148.67393494],
       [ 111.32766724,  111.33430481,  111.34093475, ...,  148.65904236,
         148.66567993,  148.6723175 ],
       ...,
       [ 116.41764832,  116.42254639,  116.42746735, ...,  143.57252502,
         143.5774231 ,  143.58233643],
       [ 116.41808319,  116.42298889,  116.42790222, ...,  143.57208252,
         143.57699585,  143.58190918],
       [ 116.41851807,  116.42343903,  116.4283371 , ...,  143.57164001,
         143.57655334,  143.58146667]], dtype=float32)
>>> n_lat
array([[ 46.99014664,  46.99103546,  46.99192429, ...,  46.99192429,
         46.99103546,  46.99014664],
       [ 46.98567581,  46.98656464,  46.98745346, ...,  46.98745346,
         46.98656464,  46.98567581],
       [ 46.98120499,  46.98209381,  46.98297882, ...,  46.98297882,
         46.98209381,  46.98120499],
       ...,
       [ 21.55286217,  21.55351639,  21.55417252, ...,  21.55417252,
         21.55351639,  21.55286217],
       [ 21.54824638,  21.54890251,  21.54955864, ...,  21.54955864,
         21.54890251,  21.54824638],
       [ 21.54363251,  21.54428864,  21.54494286, ...,  21.54494286,
         21.54428864,  21.54363251]], dtype=float32)
>>>
```

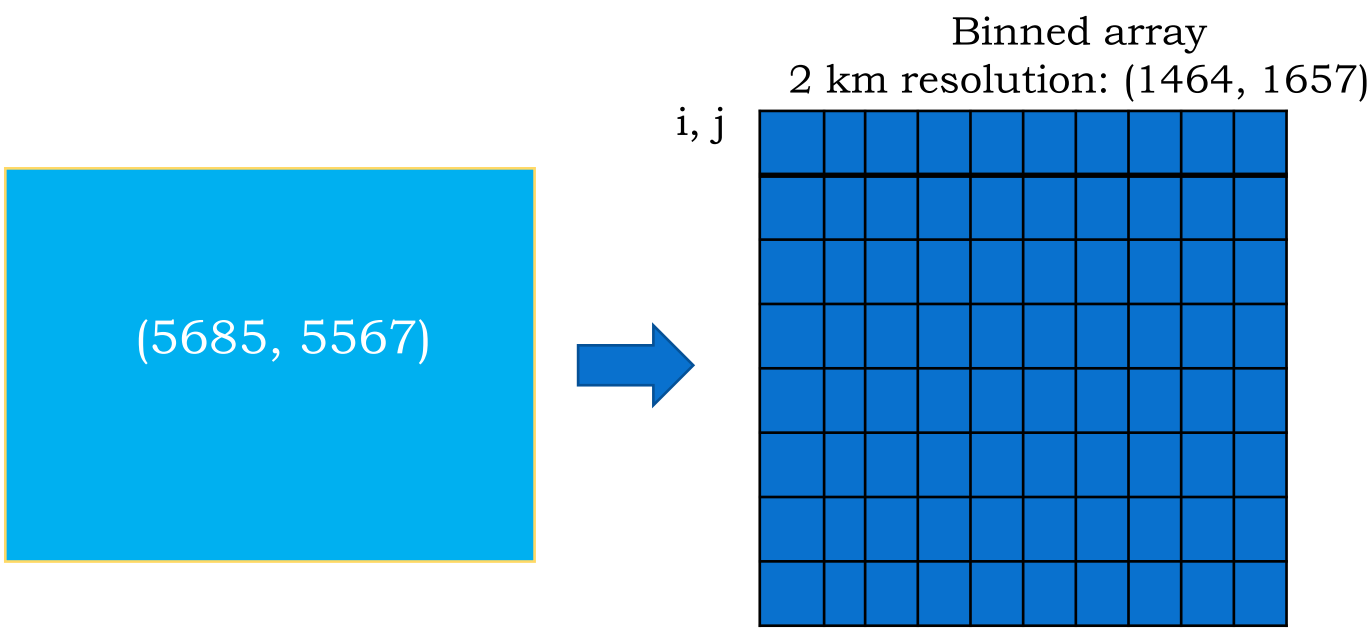
→

```
>>> lon_proj
array([[-1414212.62986592, -1413697.94909652, -1413183.27278737, ...,
          1413181.57533631,  1413696.25165638,  1414212.06405619],
       [-1414204.91518489, -1413690.19526128, -1413176.04566359, ...,
          1413175.47979768,  1413689.0635367 ,  1414203.7834676 ],
       [-1414197.73886242, -1413682.97979064, -1413168.88725866, ...,
          1413167.18951407,  1413681.84796825,  1414196.60704732],
       ...,
       [-1410712.69116666, -1410200.68309163, -1409686.29608583, ...,
          1409685.50993334,  1410197.53849239,  1410711.11887242],
       [-1410719.39460809, -1410206.5611639 , -1409692.94234717, ...,
          1409691.36998287,  1410204.98880497,  1410718.60843131],
       [-1410726.06706007, -1410211.64326925, -1409699.60017968, ...,
          1409697.24154422,  1410210.85706011,  1410724.49464717]])
>>> lat_proj
array([[ 1471796.44611518,  1471791.79293722,  1471787.17769274, ...,
          1471786.82827408,  1471791.4433911 ,  1471796.32955727],
       [ 1471285.04506912,  1471280.39607452,  1471275.90148408, ...,
          1471275.78501492,  1471280.16305137,  1471284.81196097],
       [ 1470773.75705593,  1470769.11220386,  1470764.2065738 , ...,
          1470763.85717777,  1470768.87918825,  1470773.52395533],
       ...,
       [-1399258.93682655, -1399250.38926165, -1399241.90493522, ...,
         -1399242.00356373, -1399250.78391895, -1399259.13422677],
       [-1399771.85546958, -1399763.19548552, -1399754.61103394, ...,
         -1399754.80828549, -1399763.39280875, -1399771.95416701],
       [-1400284.5623396 , -1400276.09821498, -1400267.52460887, ...,
         -1400267.82047798, -1400276.1968738 , -1400284.75972899]])
```

L2_binning

```python
# TODO: This does not account for the extent deformation induced by the projection
top_left = prj(west, north)
bottom_right = prj(east, south)

# Convert the lat and lon arrays to projected coordinates
lon_proj, lat_proj = prj(n_lon, n_lat)

# Define output array shape (nrow, ncol)
destination_shape = ( int( abs(top_left[1] - bottom_right[1]) / float(resolution)), int( abs(top_left[0] - bottom_right[0]) / float(resolution)) )

# Define affine transform and the inverse transform
#aff = Affine(resolution, 0.0, top_left[0], 0.0, -resolution, top_left[1])
top_left1 = (-1384000.0, 1600000.0)
aff = Affine.from_gdal(top_left1[0], resolution, 0.0, top_left1[1], 0.0, -resolution)
ffa = ~aff

# Convert projected coordinates to destination array indices
destination_ids = ffa * (lon_proj, lat_proj)
xedges = range(0, destination_shape[0] + 1, 1)
yedges = range(0, destination_shape[1] + 1, 1)
y = np.asarray(destination_ids)[1]
x = np.asarray(destination_ids)[0]
x1 = x.flatten()
y1 = y.flatten()
```

Used affine module to convert world coordinate to image coordinate.



Visualization: mapping, saving figure, and writing as nc

```
lat_0 = '35.0'
lon_0 = '130.0'

fig = plt.figure(figsize=(7,7))
ax = fig.add_axes([0.03 ,0.03,0.95,0.95])
lons = n_lon
lats = n_lat

m = Basemap(projection='laea', resolution='h', lat_0=lat_0,lon_0=lon_0,
 llcrnrlon=nanmin(lons), llcrnrlat=nanmin(lats), urcrnrlon=nanmax(lons), urcrnrlat=nanmax(lats) )
xm_bin, ym_bin = m(lon, lat)
m.drawcoastlines(linewidth = 0.5)
m.drawlsmask(land_color='gray',ocean_color='black',lakes=True)
#m.drawparallels(arange(-90.,90,5.),labels=[1,0,0,0],fontsize=10,linewidth=0.5,color='white')
#m.drawmeridians(arange(0.,360.,5.),labels=[0,0,0,1],fontsize=10,linewidth=0.5,color='white')

color = [(plt.cm.gist_rainbow(i)) for i in range(225,0,-1)]
new_map = colors.LinearSegmentedColormap.from_list('new_map', color, N=len(color))
clevs = logspace(-1.3979400086, 1, 100)
log_norm = colors.LogNorm()
cs = m.contourf(xm_bin,ym_bin,chlor_avg,levels=clevs,cmap='jet', norm=log_norm)
cb = m.colorbar(cs,"bottom", size="5%", pad='10%',ticks=[0.04, 0.1, 0.5, 1, 5, 10])
cb.ax.set_xticklabels(['0.04', '0.1', '0.5', '1', '5', '10'],fontsize = 11)


title = 'MODIS Chl-a '+modis_time.strftime("%b")+' '+str(yr1)+'-'+str(yr2)
ax.text(0.03, 0.96,title, fontsize=12, fontweight='heavy', style = 'italic', transform=ax.transAxes, color = 'white')
fdir = 'F:\\MODIS\\'
plt.savefig(fdir+'MODIS_'+str(yr1)+'-'+str(yr2)+'_climt_'+modis_time.strftime("%b")+'_03-05UTC.png',dpi=400)
plt.close('all')
```

```
# generating nc file
f = Dataset(fdir+'MODIS_'+str(yr1)+'-'+str(yr2)+'_climt_'+str(mm).zfill(2)+'_03-05UTC.nc','w',format='NETCDF4')
x = f.createDimension('lon',1657)
y = f.createDimension('lat',1464)

latitude = f.createVariable('latitude', float32, ('lat','lon'), fill_value=9.96921e+36)
longitude = f.createVariable('longitude', float32, ('lat','lon'), fill_value=9.96921e+36)
count_sum = f.createVariable('count_sum', float32, ('lat','lon'), fill_value=9.96921e+36)
array_sum = f.createVariable('array_sum', float32, ('lat','lon'), fill_value=9.96921e+36)

longitude[:,:] = lon
latitude[:,:] = lat
count_sum[:,:] = cnt_sum
array_sum[:,:] = dst_sum

f.close()
```

세계가 주목하는 천리안 해양관측위성

해양수산부의 세계최초 정지궤도 해색위성

Advantages

- **Relatively fast** in reading, calculation, writing using my personal computer
  - ✓ Binning 365 GOCI images takes only several minutes (Python >> IDL)
- **Programming and visualization with the same tool**
  - ✓ My personal preference (Python = IDL >> Fortan + visualization tool)
- Can make a reasonably good figure only using default python module/library
  - ✓ Not too many extra works are necessary to make a figure fancy (Python >> IDL)
- Free, versatile: easy to transferring the developed algorithm or system to anywhere (but depending on the version of modules)

Issues

- Difficulties
    - My codes do not work on my Macbook after updating a library.