Keith Arora-Williams
May 7, 2016
Landscape Hydrology Final Project

General Lake Model of Upper Mystic Lake

Introduction

I covered most of the introductory information in my slides and proposal. So I'll keep to describing the actual code I have written and have not written and why. At present it is ~1700 lines of code, but needs a lot of work still. The code, data files, and code for the figures are here: https://github.com/karoraw1/GLM_Wrapper.git

Structure

The file tree of the repository is laid out as follows with the data, and plot files left out:

```
├── Bathymetry.py: used to plot, scale, and interpolate area-depth relationships
├── GEODESC/
│   ├── build_db.sh: SLURM batch file for submission to MARCC
│   ├── GEO_SpatialFilter.py: A function used to filter each archive to just the closest points
│   ├── GEO_MetaData.py: Used to build the metadata & archive databases
│   ├── GEO_Plot_Swaths.py: used to make map-based plots of swaths with Basemap
│   ├── GEO_MetadataOnly.txt: a list of FTP URLS for metadata XML files
│   ├── GEODESC_CloudFiles.txt: a list of FTP URLs for HDF5 archives & metadata XML files
│   └── test_files: contains successful data and metadatabase builds.
├── glm_case_folders/
│   └── sunnyDay/: auto-gen folder for config files & formatted data inputs for one run
│       ├── glm2.nml: the main configuration file for the control case
│       └── output/: auto-gen folder assigned as output destination for data created by GLM
├── donePlots/ contains plots used in this report
├── JD_converter.py: This converts Julian Days to Gregorian Calendar Days
├── LakeModel.py: class definitions for Lake and data objects
├── LICENSE: I didn't know which license to use BSD2/3, MIT, or GNU
├── MakeGLMConfigs.py: main script containing Lake and each type of data object
├── README.md
```

├── `waterdata/` contains USGS water data that will be used to create inflow.csv and outflows.csv input files to GLM. Also contains Bathymetry for Hobbs Brook and groundwater levels.

└── `weatherData/` contains three of the four weather data products

```
    ├── BostonLoganAirportWeather.csv
    ├── CERES_SSF_XTRK-MODIS_Edition3A_Subset_2010010102-2014010116.nc
    └── CERES_SYN1deg-Day_200508-201406.nc
```

GCHN Data

The GCHN data was the easiest to read in and contained only 2 missing values across all of 2000-2015. It contains daily values on wind speed, temperature min/max, precipitation, & snowfall. It is called as an object with an associated Pandas dataframe.

```
BOS_weather = LakeModel.GHCN_weather_data(met_path)
BOS_weather.clean_columns()
```

CERES Data

Two data products were obtained from CERES. The first was the Level 2 SSF product including humidity, cloud cover, and temperature various others. The bounding box utilized to subset the data was too restrictive. A larger one was used obtaining the Level 3 SYN-deg product containing two sets of measurements obtained at two different points in space. One (42.5 N, 71.5 W) is 18.5 miles northwest of the site. The other is 1 degree to the west and 33.5 miles north east of the site in the Atlantic Ocean. The two SYN products were much more alike (R=0.94) than like the Giovanni data (R = 0.71, R=0.73). However they clustered away from the less processed Level SSF (R = 0.15, 0.19, 0.21). A few months of the SSF & Giovanni signals are shown in **Figure 5**. The data objects are invoked as follows:

```
CERES_SSF = LakeModel.CERES_nc(ceres_ssf, "3")
CERES_data = LakeModel.CERES_nc(ceres_SYN1, "4")
test.read_GEODISC_cloud_data('cloud_data_5pt.csv',  GEODISC_path)
```

The GEODISC data needs to be broken off into its own object and not be associated with a Lake object method. Based on the error metrics, I think the Level 3 SYN-deg Cloud Area Fraction is the best choice, but the Column-Averaged Humidity can still be sourced from the Level 2 SSF archive, as it is not included in the more processed product.

GEODISC Data

The forms of cloud fraction data hosted by GEODISC are associated with four NASA mission/instruments: OMI/Aura, MODIS/Aqua, AIRS-AMSU, and GPM/GMI. These data sets are provided in various flavors of temporal resolution, spatial resolution and units, along with derived products. The OMMYDCLD product is what I chose to use as it includes the OMCLDO2 product, as well as a second "effective" measurement of cloud fraction that integrates the MODIS06_L2 product.

Mirador presents data, subsetted by time and location, in the form of a list of FTP URLs. Each URL is either an HDF5 archive of data, or an XML file containing corresponding metadata. As a defense against the spottiness of the CERES cloud fraction data, an additional 6 years of data was requested from Oct. 2004 to Jan. 2014. A total of 5,197 archive & metadata URL pairs were presented. The script `GEO_MetaData.py` and associated helper functions, `GEO_SpatialFilter.py` and `GEO_Plot_Swaths.py`, were written to process and explore this data.

A single archive was downloaded for some preliminary analysis (`test_f.he5`). The h5py module was used to read in each archive in the form of a hierarchal dictionary, with multiple layers of keys, eventually leading down to data stored as `numpy` arrays. Cloud fraction measurements were presented as a "swath" i.e. matrix of 60 geo-location coordinate pairs x ~1600 time points, many of which (~14%) null values. A visual representation is shown in **Figure 2**, which recreates a figure shown in the data product's documentation.

Due to the large amount of unrelated information in each archive, a location-based filtering scheme was implemented. To determine the appropriate distance threshold, the number of observations captured at varying distances was plotted (**Figure 3 (Top)**). Based on this a flat threshold of 0.3 degrees was selected and the entire dataset was extracted. However, an improved approach that dynamically selects the minimum possible distance threshold (i.e. the red dot in **Figure 3 (Top)**) was implemented after extraction and will be used in the future. The effect on the quantity and standard deviation of retained data points in a sample of 160 swaths is shown in **Figure 3 (Mid & Bottom)**. An important note is that the fraction of non-null values observed in the full, flatly-thresholded data set (85.26%) did not significantly change compared to the dynamically thresholded sample (88.75%)

The non-null, location-filtered measurements of normal & effective Cloud Fraction were compared. The amount of variation between the two measurements was much larger across the whole data set, compared to that of clusters of adjacent points. The normal Cloud Fraction data was used as this was described in the model text, however this may merit more investigation.

Upon inspection, it was determined that the time/date of each measurement was only included in the metadata. The time stamps within the archives were given in seconds since the start of measurement and thus required context. To address this, a function was defined to extract the corresponding archive name & time stamp data from each metadata ASCII-format XML file.

The space required to store all 5,1987 relevant archives was estimated at about 84.61 Gb and the amount of time required to download & parse each in series was between 43 and 58 hours. To cut this down, two functions were developed for downloading, parsing, and removing remote files. The modules `Joblib` and subprocess were utilized in both. `Joblib` splits loop-based processing into multiple parallel threads or Python processes. Subprocess allows commands to be issued to the BASH shell, such as `wget`. The function accepts the list of metadata files provided by GEODISC for the user-specified time frame. The current design splits processing metadata and actual data between two functions. The processed metadata is stored as a CSV, which is read back into memory when data from processed swaths can be added. The final result is stored in a <1 Mb CSV file which can be fed to the `Lake` class method `read_GEODISC_cloud_data()` for further processing.

The first successful trial run download time for all files was ~ 8 .3 hours. Histograms of the mean & standard deviation of cloud cover, as well as the standard deviation in geo-position, and the time of day are shown in **Figure 4**.

USGS Water Data

The USGS data is imported into a `Pandas` DataFrame. A text parser was added as a class method to substitute the numerical column labels with human readable units. The data contained in the `waterData/` folder pertains to the inflow & outflow rivers the drain Upper Mystic Lake, as well as data from a number of rivers and reservoirs nearby. The date ranges vary but the contextual data was gathered as a source of comparison because the time coverage for the outflow river is too short to be used alone. There was conductance data for two of the nearby rivers, although none for Upper Mystic Lake. Located 11 km away, Hobbs Brook is a river that drains into a basin 4x as deep as Upper Mystic lake (~22 m deep). Another monitoring station was placed at the mouth of the stream that feeds a much shallower reservoir. Fresh Pond at a distance of 9 km. The difference in depth may be related to the difference in temperature during the winter shown in **Figure 6**, but I am not sure about the change in salinity. The object instantiation & processing looks like this:

```
Hobbs = LakeModel.USGS_water_data(flow_ps[3])
Hobbs.preprocess()
Hobbs.read()
Hobbs.print_metadata()
```

Development Goals

Organizational Problems

The code needs to be restructured so that the Lake object accepts only the values associated with parameters and not the names as well. The code itself should be better organized such that there are sequential sections where each category of input parameter is fed in one at a time. There are plenty of functions that are not associated with methods and such orphans are symptomatic of poor organized object hierarchy. The same can be said for the plot processing steps, which can at least be arranged into loops for brevity.

Improving parallelization

There are two major faults in the current implementation. The major problem involves the fact that downloading archives will occasionally get hung up. By attempting to process and download in a single parallelized step, frequently IOErrors were raised when the script looked to open a file that either hadn't downloaded or had required multiple attempts, leading to the addition of a numerical suffix (*.1). To address this, the best course would be to split downloading and parsing into sequential steps. Additionally, implementing a Python-based method of downloading FTP files might improve coordination with Python-based parallelization tools. Initially, the Python package `urllib2` was tested for this purpose, but it didn't work with Joblib, however `ftputil` appears to be better alternative.

The second weakness involves the call to the `Parallel` class in Joblib, which uses all the default arguments. The default choice for the `backend` argument is `multiprocessing`, which limits concurrent processes to the number of available CPUS (e.g. 8 for my computer) . Future work should compare performance of the `multiprocessing` and `threading` back-end options available in the `Parallel` class.

Wrap GLM and Automate Parameter Input

Implement function to run GLM from Python in parallel. Although use of `Joblib` was moderately successful in initial trials database creation, further reading revealed that the subprocess module could be used without help to run spawn independent processes. I have sufficient input data for testing purposes. This is my primary development goal at this point.

Data validation

More cross-referencing of data sources prior to integration of all input time series's into a single duration & frequency aligned data frame. Multiple sources & formats for temperature, precipitation, and wind-speed have not been compared. This could be done using the error metrics implemented in the error_metrics() function (KGE, NSE, R) and is meaningful, as shown in F**igure 6**. Warnings and visual representations of deviations in input data should be presented to the user.

I am also interested in the resemblance of PDFs/PMFs of distinct data types. This can be observed in the histograms in Figures 7 and 4. There data presented in the histograms appear to fall into 3 groupings. One is typified many low value samples and few large events. These include snow, rain, and cloud cover, but also net short wave radiation flux. The second grouping includes temperature and short wave radiation, which appear to be mixtures of two or more Gaussians. The third grouping contains humidity and wind speed and resemble a left-skewed univariate Gaussian. A key step in manipulating and modeling these data types would be to determine which distribution best models them. A function could be developed that takes a time series, coverts it into a PMF or CDF and then performs a goodness-of-fit test to determine what combination of distribution & shape parameters best model each distribution.

Figures

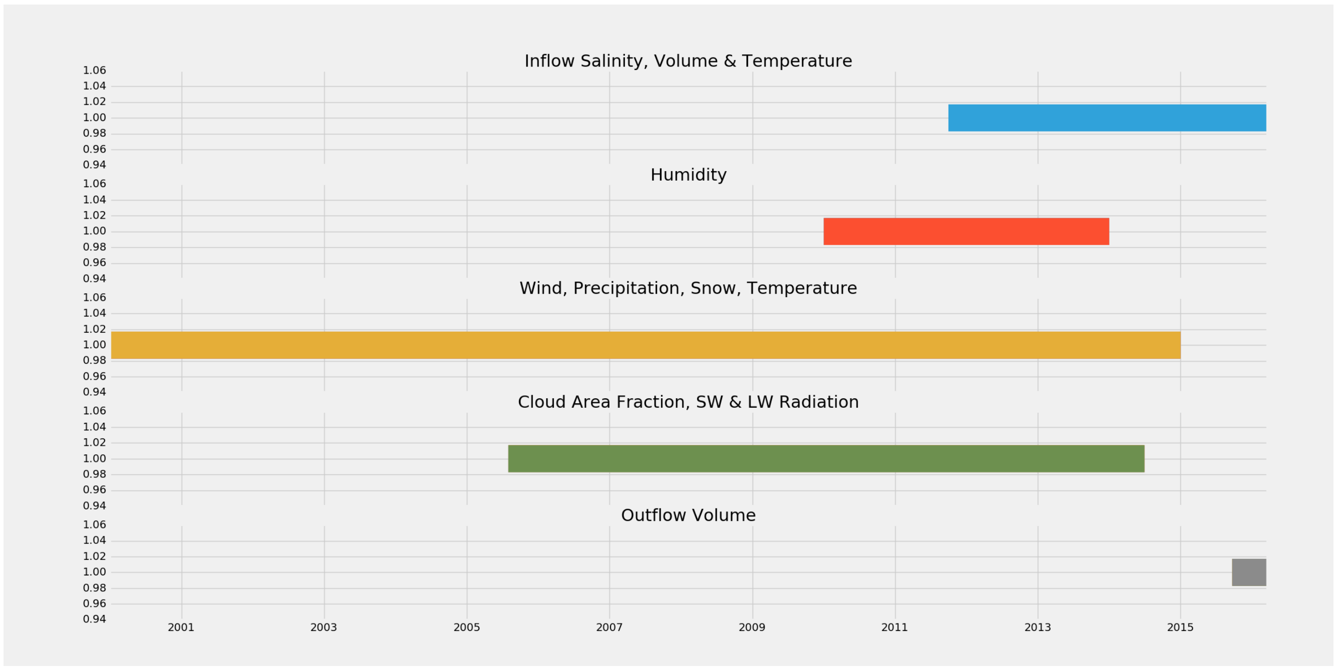
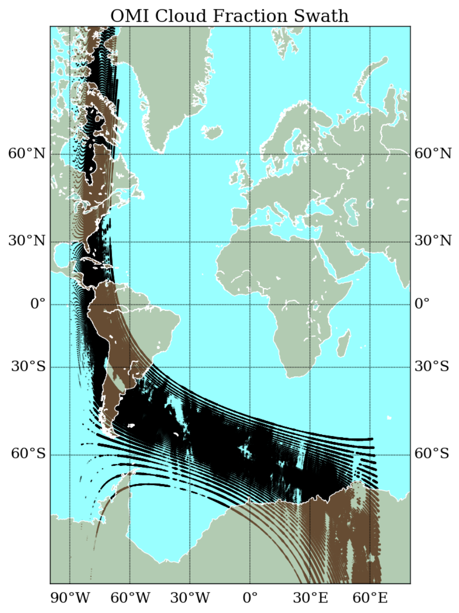


Figure 1: Data Set Temporal Coverage
This shows the time covered by the data set. Hopefully there is enough over that 8-10 months of overlap between inflow and outflow datasets that a longer one can be estimated within a reasonable confidence bound.



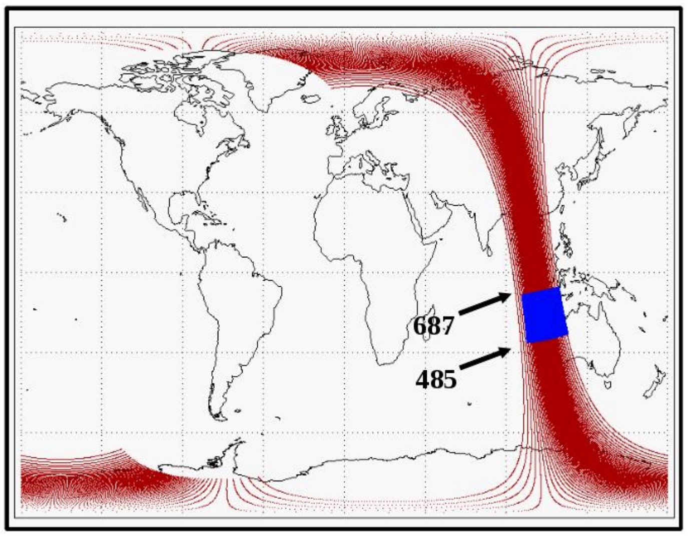


Figure 2: A Swath of Image Data
A plot of the test archive swath is shown below on the left. Null values and zeros show up as blank regions in the plot. The size of each point is proportional to observed cloud fraction, however this does not show up well. The purpose of the plot was simply to successfully achieve large scale geographical visualization, with the option of zooming. The plot on the right is a different swath taken from the data product's documentation. The blue box is a region of MODIS / OMI data overlap, from which the effective Cloud Fraction is derived.

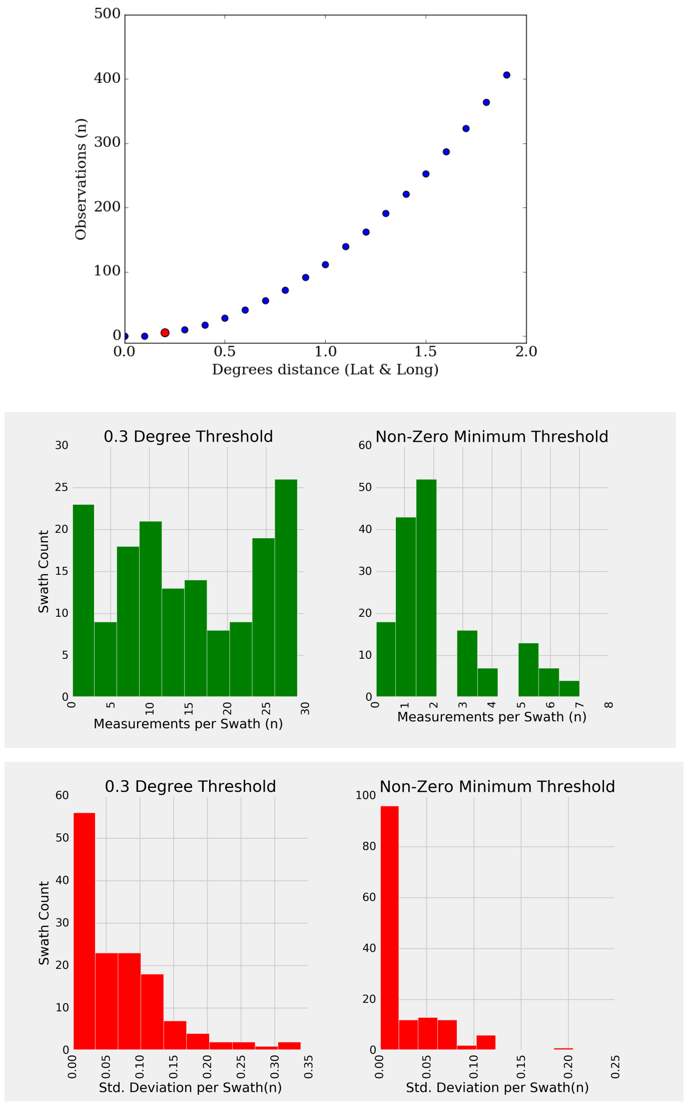


Figure 3: Adaptive Filtering
The top plot shows the relationship between distance thresholds and observations. The red marker shows minimum non-zero distance. The bottom pairs of plots show the effect of retaining all measurements within a 0.3 degree threshold (left) or using an adaptive minimum threshold (right) on Std (bottom row) and sample size (middle row)

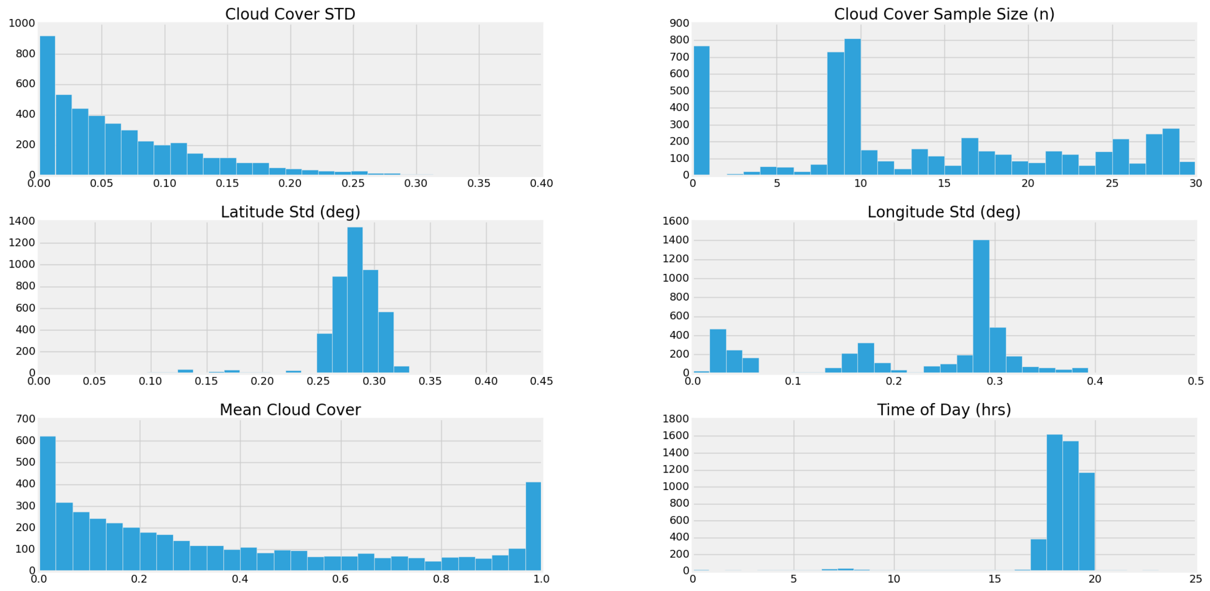


Figure 4: Cloud Cover from Extracted from Swath
Histograms of the mean & standard deviation of cloud cover, as well as the standard deviation in geo-position, and the time of day. The data were filtered based on a maximum distance of 0.3 in latitude and longitude. The bottom-right plot shows that most measurements were made at around the same time of day. Distinct data products for time & day versions of cloud cover were offered, however only 24 hr averages are accepted by the model. In regions where large variations in humidity occur throughout the day during certain seasons, this might make a difference.

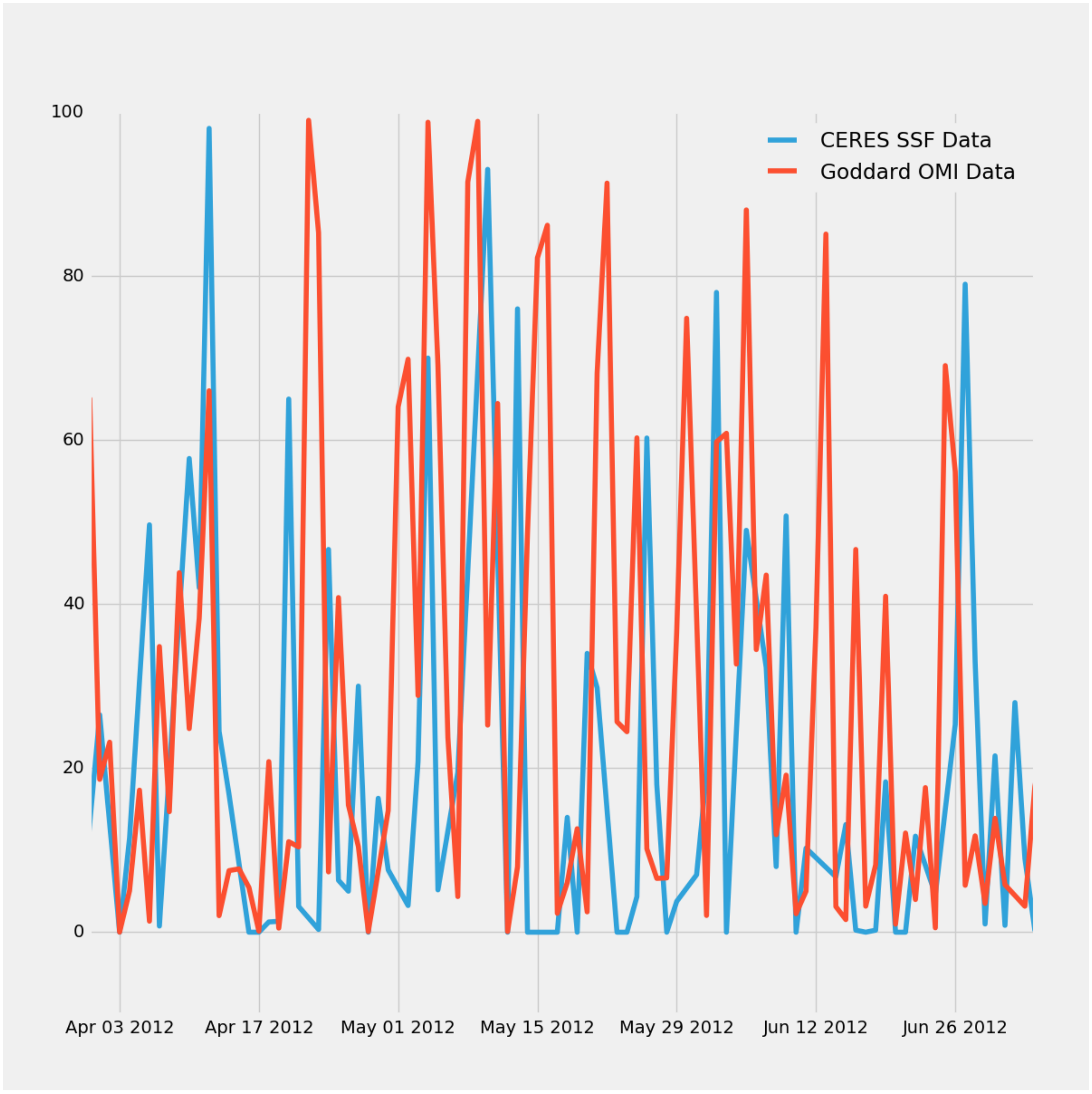


Figure 5: Cloud Cover from Different Sources
CERES Cloud Cover Level 2 Data & Goddard (GEODISC) Cloud Cover Level 3 Data are shown for the given duration. The indexing seems to be in sync although the Goddard data looks to be higher resolution.

Figure 6: Seasonal Variation between two nearby drinking Reservoirs (Fresh Pond &

Hobbs Brook) & Fit Metrics

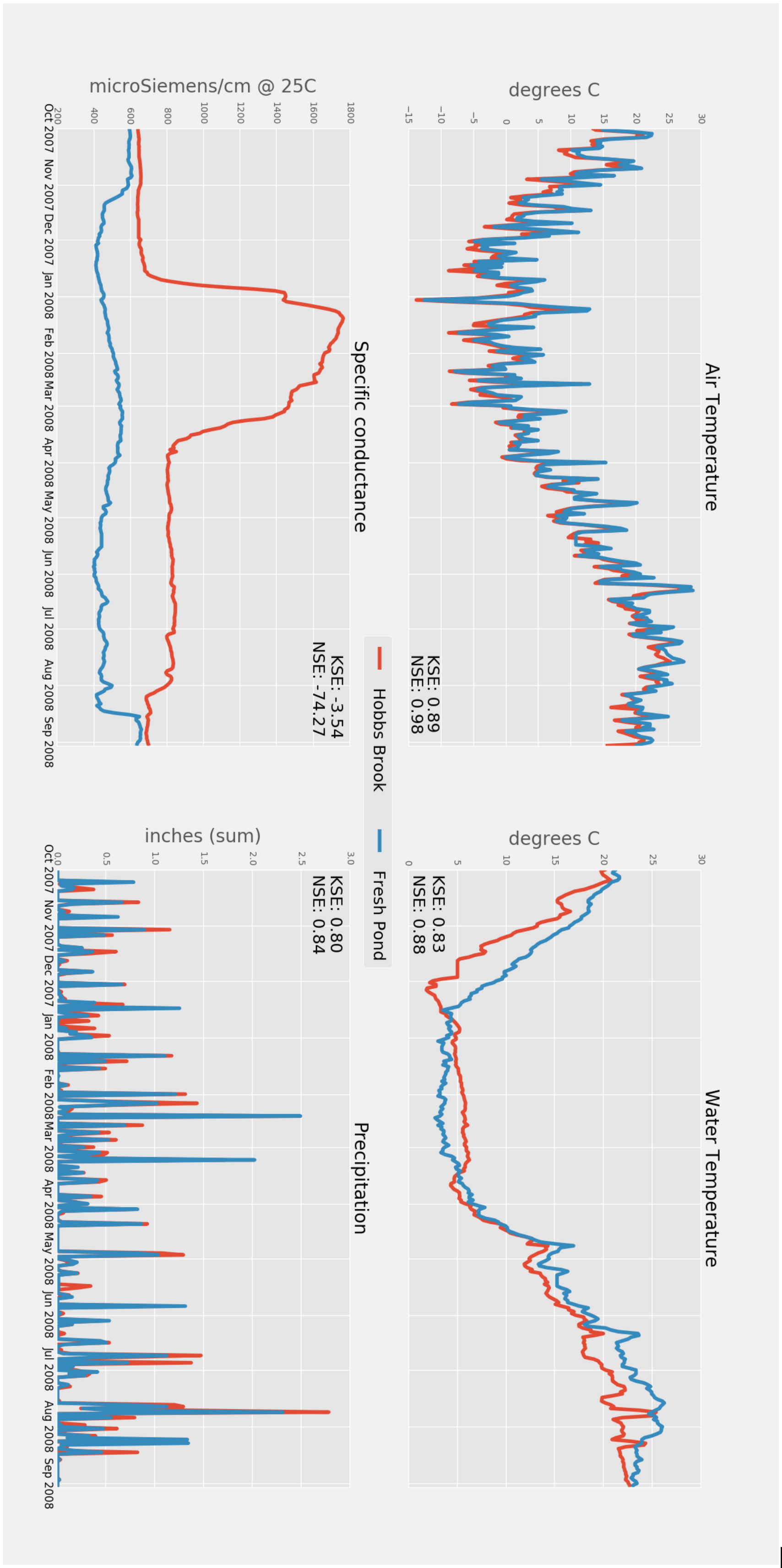


Figure 7 GCHN **Data (top) &**

CERES SSF data (bottom)