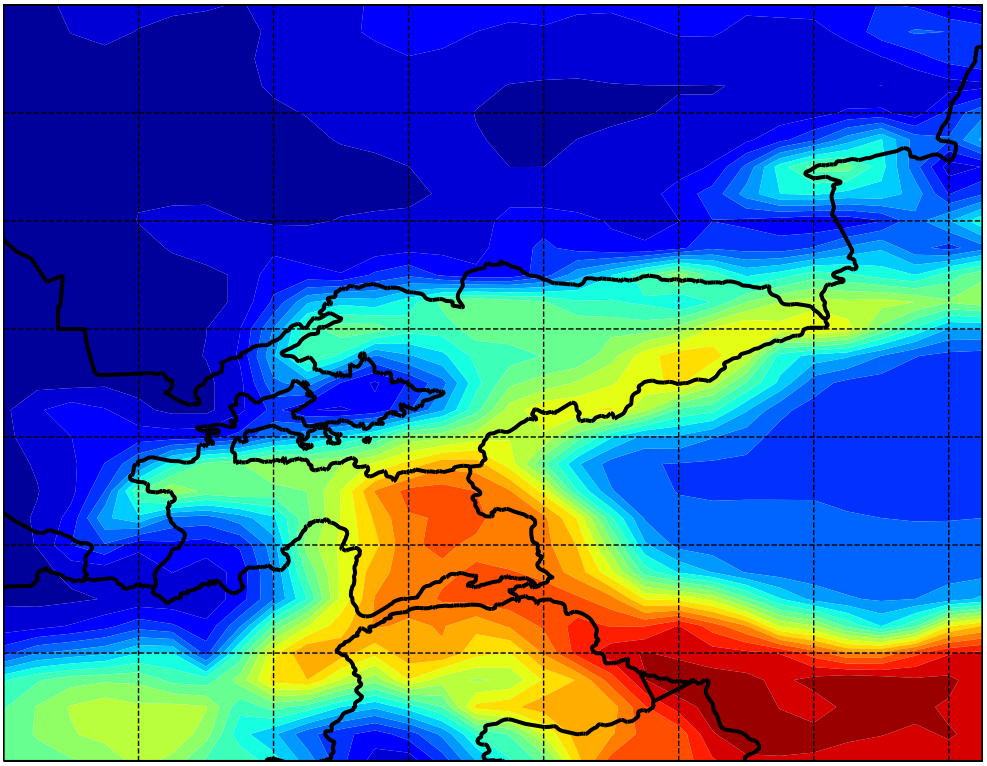
American University of Central Asia

Internship Report RS RAS



Place of Internship

Research Station

of the Russian Academy of Sciences, Bishkek

Program

Applied Mathematics and Informatics

Student

Moritz M. Konarski

Group

2017

November 13, 2020 – Bishkek

Contents

1 Introduction

This report covers the tasks and results of my internship at the Federal State Budgetary Institution of Science Research Station of the Russian Academy of Sciences in Bishkek (RS RAS). RS RAS is a subsidiary of the Ministry of Science and Higher Education of the Russia Federation and employs 137 people. Since 1978 it has been researching seismic processes and developing geodynamic models. The internship took place from the 7th of September 2020 to the 7th of November 2020 and was conducted remotely due to the continuing COVID-19 pandemic. My AUCA supervisor for this internship was Olga Zabinyakova, Scientific Secretary of RS RAS and my RS RAS supervisor was Sanzhar Imashev, Acting Head of the Laboratory for Integrated Research of Geodynamic Processes in Geophysical Fields.

The internship was split into an educational section and an industrial section. According to my internship dairy form, the aim of the educational internship was to acquire the knowledge necessary to understand the research carried out by RS RAS. During the industrial part of the internship I should participate in a certain part of their work or in work that is similar to theirs. To fulfill these requirements my RA RAS supervisor gave me the following tasks for my educational internship:

1. familiarize yourself with web resources providing access to NASA Earth Remote Sensing data;
2. familiarize yourself with the scientific data format netCDF (Network Common Data Form);
3. study libraries used to work with the netCDF format in various computing environments.

For the industrial internship I was tasked to:

1. register on the NASA Earthdata platform to access satellite data;
2. develop a library for working with netCDF files in the Python programming language (using satellite data as an example);
3. develop a computer application for data visualization and reanalysis of NASA MERRA2 satellite data.

These tasks are outlined in my internship diary form. In the next section of this report I will cover the educational part of the internship and talk about NASA Earth Remote Sensing data, the netCDF data format, and libraries used to work with netCDF files.

Then, I will detail the industrial part of the internship and talk about registering on the NASA Earthdata platform, downloading NASA MERRA2 data, and the development of the Python application. The Python application development will be split into multiple sections. As part of the industrial internship section the development process will be described. In a separate section the finished program with all of its components will be discussed.

As a practical example of using the program, I will follow the suggestion of my supervisor and analyze the vertical structure of the air temperature (variable `T`) to show its vertical gradient, the stratopause, and the behavior around the ozone layer.

2 Educational Internship

2.1 NASA Remote Sensing Data

NASA Remote Sensing data is available via the Earth Science Data Systems (ESDS) Program (see here). The program covers the data acquisition, processing, and distribution with the goal to enable the widespread use of NASA mission data. The data is available for free and their software is publicly available as Open Source Software.

Part of ESDS is the NASA Goddard Earth Sciences (GES) Data and Information Services Center (DISC) which provides data on atmospheric composition, water and energy cycles, and climate variability. The GES DISC provides over 3.3 Petabytes of data, including the MERRA2 dataset. MERRA2 (the Modern-Era Retrospective analysis for Research and Applications version 2) focuses on historical climate reanalysis using satellite data. The dataset I worked with (M2I3NPASM) contains data from January 1st, 1980 until October 1st, 2020 (at the time of writing). It covers the whole globe with measurements taken every 3 hours. M2I3NPASM includes 14 measured variables in addition to latitude, longitude, time, and a pressure level. The measured variables include the surface pressure, specific humidity, eastward and northward wind, and temperature. The measurements are done on a cube sphere grid (add explanation) and later processed to fit the standard latitude and longitude grid. This processing generally involved a bilinear interpolation of data values.

The data is provided on the GES DISC website (see here) and can be accessed from there. The website gives the option to download only a subset of the data by selecting a certain time range, latitude and longitude range, and group of variables. This is advantageous because a full file for 1 day is about 1.1 GB in size. For my internship I worked with a subset of the data that includes all variables but is restricted to a latitude of 34°N to 48°N and a longitude of 65°E to 83°E. This restricts the area to Kyrgyzstan and sections of all surrounding countries. Additionally, the file size is reduced to a manageable 6 MB per file so that a whole year of data only takes up 2.2 GB (in 365 files), the same space two files of the complete data would take up.

2.2 The netCDF Data Format

The M2I3NPASM data is provided in a file format called the Network Common Data Form (netCDF). NetCDF is made up of a data format and libraries that can read and

write its data. NetCDF is developed and maintained by Unidata, a community of research institutions, with the goal of sharing geoscience data and the tools to use and visualize it. Unidata is funded by the National Science Foundation, a US government agency that promotes and supports science and research. Unidata also maintains libraries (programming interfaces) for C, Java, and Fortran. Based on these, multiple other interfaces are available, including one for Python.

The netCDF data format is specifically designed to hold scientific data. According to the Unidata website, the netCDF data format has the following features:

- **Self-Describing**. A netCDF file includes information about the data it contains.
- **Portable**. A netCDF file can be accessed by computers with different ways of storing integers, characters, and floating-point numbers.
- **Scalable**. Small subsets of large datasets in various formats may be accessed efficiently through netCDF interfaces, even from remote servers.
- **Appendable**. Data may be appended to a properly structured netCDF file without copying the dataset or redefining its structure.
- **Sharable**. One writer and multiple readers may simultaneously access the same netCDF file.
- **Archivable**. Access to all earlier forms of netCDF data will be supported by current and future versions of the software.

The NASA M2I3NPASM data is available in "classic" netCDF–4 format, meaning that it is backwards compatible. These files have 4 dimensions:

1. longitude in degrees east (meaning west is represented as negative),
2. latitude in degrees north (making south negative),
3. pressure in hPa,
4. time in minutes since the first time point in a file.

To be self-describing, the NASA M2I3NPASM netCDF files contain information about, among others, the intitution that created the file, date and time of the beginning and end of the dataset, and the minimum and maximum latitude and longitude values. The files furthermore contain metadata for each of the measured variables. The most important of these are the fill values that identify missing data, the long name, a full version of the short variable name abbreviation, and the units of the variable.

2.3 NetCDF Libraries

Because I am working with the Python programming language for my internship I require a netCDF library that works in that programming language. Unidata provides an interface between the netCDF library for the C programming language and Python. This interface is called netCDF4. It has most of the features of the C library and enables the creation and reading of netCDF files using Python. This interface is used

in my industrial internship work to work with the netCDF files downloaded from GES DISC.

3 Industrial Internship

3.1 NASA Earthdata

3.1.1 Registration

To access data on the NASA Earthdata platform (which includes GES DISC), a registered account is required. The purpose of the registration is for NASA to improve their service and to offer notifications and saved preferences. The Earthdata account then needs to be linked to a GES DISC account to access the M2I3NPASM dataset I am working with. A full guide can be found under this url.

3.1.2 Downloading

To actually download GES DISC Data there are many options, but `wget` might be preferable. `wget` is a utility for downloading files from the internet. It is generally used as a command line tool and it is available for most platforms. The steps necessary to set up wget to download data from GES DISC are outlined here. The process simply involves setting up your login information in a local file. Then, one needs to acquire the urls that point to the data that is meant to be downloaded. As described above in the section on NASA remote sensing data, a subset of the dataset (e.g. M2I3NPASM) should be specified to significantly reduce the download time. Then the GES DISC website will provide one or more download links, generally in a `txt` file. This file can then be given to `wget` (following the platform-specific instructions) to download these files. Once the files have been downloaded, they are ready to be analyzed.

3.2 Python Application Development

This subsection is based on the notes in my internship diary and will explain the process of developing the Python application. It will not go into great technical detail because I want to reserve that for the description of the finished program. Additionally, every program written during my internship contributed to the final program and thus when I describe the final result I will be indirectly describing the important results obtained along the way. The decision to work with Python was made in our first meeting and we chose it because everyone had had at least some experience with it and Python has a rich ecosystem of libraries that would enable us to complete all the parts of our assignment.

3.2.1 Simple Beginnings – 10.09. to 20.09.

The first step was to download the netCDF data from the GES DISC website to start working with it. I created the required account (as outlined above) and downloaded 3 days worth of data of the M2I3NPASM dataset using `wget`. Then, I develped a simple command line python program that enabled me to more comfortably download large quantities of files. This program used the Python requests library to download the files, and not `wget`. This program has since been retired because `wget` is completely sufficient and there is no need for this program.

The next simple command line program I wrote listed all the available variables in a netCDF file. This was the first time I worked with these files and thus I had to find out how it works. I chose the `netCDF4` Python library for this task because it is developed by the same group that created the file format itself. I also had to find out how to access the variable names that are stored in a netCDF file.

Once I had familiarized myself with the file format, I wrote a simple program capable of creating a heat map graph of a data type that only has latitude and longitude dimensions. This program was not flexible and most values were hard-coded, but I forced me to explore how to plot two–dimensional heat maps using Python. I used the `matplotlib` library to create the plot itself and to save it as a picture. To create the map features I used the `cartopy` library which specializes in creating all kinds of maps.

3.2.2 Figuring out Data Storage – 22.09. to 01.10.

Because M2I3NPASM data comes in 1 file per day it can be cumbersome to work with data that covers more than a single day (many files would have to managed at once). Furthermore, each file includes multiple variables (because one generally does not know which specific variable will be needed) which means that there is unnecessary data once one decides to analyze a single variable. To solve both of these issues, my supervisor suggested that I extract one particular variable from multiple netCDF files and save all the data into a single file. Now a file format for these files needed to be found.

The data in netCDF files is stored in multi-dimensional arrays because this structures resembles the structure of the data most closely. Thus our data format should also support multi-dimensional arrays. After considering multiple alternative options (Parquet, HDF5, netCDF), with my supervisor's advice I decided on NPZ files, a format for the `numpy` Python library. This format is convenient because `numpy` is one of the most popular scientific python computing libraries and used as a backend for many other libraries meaning that it is widely usable. Also, NPZ files natively support multi-dimensional arrays which makes them a good fit for our data. The compression of NPZ files is another advantage because it saves space when the file becomes large. NPZ files were chosen over NPY files, which are also files used by the `numpy` library,

because NPY files are not compressed and thus take up more space. NPY files also only hold one single multi-dimensional array while NPZ files can hold multiple arrays. This enables me to store all the necessary data in a single file (the actual data plus data for the dimensions time, latitude, longitude, level).

Unfortunately NPZ files are not self-describing and cannot hold information about the data that they contain. To not lose the information about our data that the netCDF files hold, another file type to store this information was required. For this purpose I chose JSON files, which can easily be read from and written to using Python and most other programming languages. Conveniently, the Python dictionary data type – a list if key-value pairs, e.g. `"age": 21` where `"age"` is the key and `21` is the value – can be easily converted to JSON and stored for later use. This is the approach I decided to implement.

After choosing these files types, I started to develop a program that would take every netCDF file in a directory and take the data of one variable from each file and put it into one large multi-dimensional array. It also extracts the values for the latitude and longitude as well as the start time of the first file and the end time of the last file. The data, latitude, and longitude are then saved to a single NPZ file. Then the important metadata – the minimum and maximum values of data and dimensions, start and end time, measurement intervals, units, variable names – are extracted, put into a Python dictionary, and then saved as a JSON file.

3.2.3 Improving the Data Extraction and Plotting – 07.10. to 21.10.2020

After both basic versions of a heat map plotting program and a data extraction program were developed, I worked on making them work together and iteratively improving them.

Data Processing. When the processed data was first being used in plotting some bugs became apparent. The most severe of these was a mistake I made in handling the masked arrays contained in the netCDF files. These arrays contain data which might not be valid for certain values of the dimensions. In the netCDF files all of these values are filled with a special value called the `_FillValue`. This value is specified in the netCDF files and can be used to process the files correctly. If it is not handled correctly, the fill value will be interpreted as a proper value and make any plot unusable. What I did to fix this issue was to replace every occurrence of the fill value with the `numpy` data type `numpy.NaN` (a specfic value meaning Not a Number). This makes it simple to ignore these values in calculations because there can be no confusion about whether or not the numbers are valid. The fill value is also being saved to the metadata file so that it can be used at a late date.

Plotting. The main changes to the plotting program in this part of the development was the creation of a program that can plot time series, unifying the heat map and time series plotting program, and enabling the plotting programs to work with the above mentioned `numpy.NaN` values.

To plot a time series one has to select data for a specific point in space and then plot all the values of that point from a start date and time to an end date and time. The most challenging element here is the conversion of a user-entered date and time in the format `YYYY-MM-DD H` to a computer-readable date and time object and then to an array index that can be used to access the data. This was achieved by reading in the user-specified date and time as text, using a Python standard library function to convert it into a `datetime` data type. Then one can find the difference between the user-specified date and time and the start date and time of the data which can then be used to find the index of the array that corresponds to the given time.

Unifying the heat map and time series plotting involved copying the code of both programs to a single file and then setting up the command line program to accept the types of input required to plot each of the types of plots.

To get the plotting programs to work with `numpy.NaN` values, the only change I had to make to the programs was to change the functions that find the minimum and maximum values to functions that ignore `numpy.NaN` values. If this is not done, any operation involving a `NaN` value will itself result in `NaN`. Now the `NaN` values are left out of computations and will not invalidate the results.

3.2.4 Starting a GUI – 22.10. to 30.10.

4 Python Program Development

What does it do? How does it do it? What are the implementation details?

Make a step-by-step guide with screen shots

5 Data Investigation

I think that you should have Results section in your report where you will analyze those satellite data. May be seasonal variations. Day/Night min max values or spatial variations. Or even vertical structure of air temperature I suggest you to explore vertical structure of air temperature. Show tropopause zone. Show vertical gradient of temperature and why it is so. Show stratopause zone and explain inverse of gradient due to ozone layer. There is a lot of info about these zones and you can easily find it, and add good photos, graphs from Internet to have more interesting presentation. It will be not just software work but also with some scientific taste.

6 Conclusion

One of the desired outcomes of educational and industrial practitioner is the formation of research directions and topics student's graduation bachelor's work.

- what did I learn during the internship
- which skills did I receive

References

[1] "NumPy reference," 06.2020. Release 1.19.0. Retrieved from: https://numpy.org/doc/1.19/numpy-ref.pdf.

[2] W. KcKinney and Pandas Development Team, "Pandas: powerful python data analysis toolkit," 08.2020. Release 1.1.3. Retrieved from: https://pandas.pydata.org/docs/pandas.pdf.

[3] "The netcdf user's guide," Accessed 23.10.2020. URL: https://www.unidata.ucar.edu/software/netcdf/docs/user_guide.html.

[4] "Netcdf4 module documentation," Release 1.5.4. Accessed 23.10.2020. URL: https://unidata.github.io/netcdf4-python/netCDF4/index.html.

[5] J. Hunter, D. Dale, E. Firing, M. Droettboom, and the matplotlib development team, "Matplotlib," 09.2020. Release 3.3.2. Retrieved from: https://matplotlib.org/3.3.2/Matplotlib.pdf.

[6] "Cartopy documentation," Release 0.18.0. Accessed 23.10.2020. URL: https://scitools.org.uk/cartopy/docs/latest/.

[7] "Python documentation: json – json encoder and decoder," Release 3.9.0. Accessed 23.10.2020. URL: https://docs.python.org/3/library/json.html.

[8] "Python documentation: Built-in types – dict," Release 3.9.0. Accessed 23.10.2020. URL: https://docs.python.org/3/library/stdtypes.html#typesmapping.

[9] "Python documentation: datetime – basic date and time types," Release 3.9.0. Accessed 23.10.2020. URL: https://docs.python.org/3/library/datetime.html.

[10] "Ipython documentation," Release 7.18.1. Accessed 23.10.2020. URL: https://ipython.readthedocs.io/en/stable/.

Appendices

A Code Listings

What does it do? How does it do it? What are the implementation details?

B Supplemental Graphs

put graphs here that don't fit into the main body

C Tables

tables that contain data values for illustration maybe