Introduction to the IAU AG Engel `toolpac`

Sophie Bauchinger

February 2023

This tutorial aims to introduce you to the AG Engel Python module `toolpac`, which provides useful tools for analysing air samples. Working through these examples will introduce you to getting information from various file types, bringing information onto common time scales, plotting in 1D and 2D, as well as identifying outliers and quantifying the time lag of your data with respect to a reference time series.

Setup

If you have yet to install python, the easiest way to do so it by downloading anaconda (including Python 3.9) from anaconda's website. Once your installation is up and running, have a look here for guidance on getting started with toolpac: `https://github.com/AtmosphericAngels/toolpac/tree/dev`. The `conda` commands can be executed in the Anaconda Prompt on Windows, or the terminal on macOS / Linux. You'll need to clone the `toolpac` git repository - you can for example use GitHub Desktop, then find the command to clone a repository and insert the URL from the toolpac github page. Make sure to clone the `dev` branch, as the `master` branch is not up to date. Set up your virtual environment as described in the toolpac README.md, then install Spyder inside the environment.

Any packages you didn't already add when setting up the environment, you can add later by going into the command prompt, activating the environment and using conda or pip to install. These are some of the packages you will be using:

- matplotlib - essential for data visualisation, especially matplotlib.pyplot (plt)
- numpy (np)- numerical computations, n-dimensional arrays, ...
- pandas (pd) - data analysis and manipulation tool
- geopandas - adding geospatial operations in pandas
- xarray (xr) - n-dimensional labeled arrays and datasets, useful for netCDF files
- toolpac

Python is an object-oriented programming language. Creating a new class, allows you to create a new type of object which can have attributes (eg. a name, a year) and methods, which are class-specific functions. An introduction to classes can be found here: `https://www.w3schools.com/python/python_classes.asp`. Familiarising yourself with this functionality a little will help you understand the toolpac code you will be using.

In Spyder, you can inspect an item with `ctrl + I`. In the 'Help' pane, this will bring up information on a function such as parameters that can be set, the output and sometimes even usage examples. The 'Variable Explorer' in the same window will show you all user-defined variables and their properties. Depending on the type of variable, double-clicking will let you explore its attributes, methods and values. You can generally check the output of a function or command in the console, and liberally use `print` statements to check variables at any point in your code. You can check the type of variable with `type()`.

Make sure to open your file in a way that doesn't leave it open indefinitely - you can either specifically call `file.close()` once you have all your information, or better yet, by using a context manager such as `with open(file) as f:`. Some functions such as `np.genfromtxt()` don't require you to do this. Also be aware that for standard libraries and packages, the American spelling of words such as 'color' and 'Normalize' is used.

1 Reading data of different file formats and data types

You'll be looking at data from two ground-based measurement stations - Mauna Lao Observatory (NOAA) and Mace Head (AGAGE) - as well as air samples taken during CARIBIC-2 flights and data from the MOZART model. Note that the files you'll be working with will have different formatting, file types and in the case of CARIBIC, you need to combine data from multiple flights, which are located in their individual directories. Missing data is also not handled consistently across different data sources.

1.1 General

Before deciding how to import data from a file, it's always a good idea to open your data in a text editor first if possible. Importantly, you should **not** be using Excel to open .csv (comma-separated values) files, even though this may be your operating system's default. Additionally, you should generally not modify data files that you did not create yourself. Although this may simplify extracting the data for one-time use, you may need to reuse your code many times for the same, updated, data; in which case having written code in a resilient way from the start will improve your workflow in the long-term. The data you will be working with in this tutorial is formatted in an international data format, so values are written with a decimal point rather than a comma. Keep in mind that when working with data from German sources, you may find that decimal points are used instead.

1.2 Ground-based measurements

Search for and download SF_6 measurement data from the observatories Mauna Lao (MLO) and Mace Head (MHD). Choose the year 2012 for MHD data and get both the daily and monthly averaged data from MLO (choose the .dat file here). The data is available on the following websites:

- Mace Head (Ireland), AGAGE:
  `https://agage2.eas.gatech.edu/data_archive/agage/gc-ms-medusa/complete/macehead/`
- Mauna Lao (Hawaii), NOAA:
  `https://gml.noaa.gov/dv/data/index.php`

By opening the files in a text editor, you can get an overview of what information there is to be found in the files, as well as the general data structure. You should be able to see that the mixing ratio measurements are located below a header outlining supplementary information for both stations.

Example 1.1. Mace Head measurements
Define a function with `def foo():` that imports the Mace Head measurement data from your file and creates a Pandas DataFrame from it. By using the `numpy` function `genfromtxt()` you will obtain the data in the form of a multidimensional numpy array. Set the argument `skip_header` so that only the actual measurements are read in. You can get the data headers by calling `line.split()` on the corresponding line of the file. For MHD, the units and column names are in separate lines, so you can take the names and units and add them together to obtain your final column titles. Now use `pd.DataFrame()` to create a *Pandas DataFrame* (df) from the array and the column titles. Convert the first column of your array (obtain this using `array[:,0]`) from fractional year to datetime objects using the function `fractionalyear_to_datetime` from `toolpac.conv.times`. Now insert the datetime objects as a new column with `df['datetime'] = array` and set it as the dataframe index with `df.set_index('datetime', inplace=True)`.

Example 1.2. Mauna Loa measurements
Data from the Mauna Loa observatory can be imported very similarly as in the previous example. Make sure to adjust the argument `skip_header` in `np.genfromtxt()` to match the number of lines in the header, and find the names of the columns from the line above the data. Note that the Mauna Loa data is available from 1998 to 2022, so you will want to write your function in a way that returns only data from specific years. For example, you can choose only the year using `df.loc[]`, then reset the index to start from your first measurement of that year.

1.3 CARIBIC-2 flight data

The files containing measured data from flights of the second phase of CARIBIC can be found on the network server `\\141.2.225.99\exchange\CARIBIC\Caribic2data\`. By mapping the server as a network

drive, you will be able to access the data easily. Note that there is a folder for each flight, which contains measurements of various parameters. The naming convention of the folders gives the flight number as well as the departure date in the form of YYYYMMDD. For these exercises, you'll be focusing on measurements of SF_6, which can be found in the text files starting with 'GHG' within the flight folders. The file names themselves give information on the data set and instrument used. In order from left to right: measured data type, departure date, flight number, departure airport, arrival airport, time resolution (optional) and version number. The data in the measurement files is formatted according to NASA Ames formatting. From the first line of the file you can see that the CARIBIC data has file format index (FFI) 1001. This format is also used for HALO campaigns and important for a lot of work in our working group.

The module `toolpac.readwrite.find` makes it easy to find all relevant files (greenhouse gas measurements) distributed across many different folders and subfolders. You can find all files with a specific string in the directory name (eg. flight number or date) using the function `find_dir`, or similarly call pyffind_file to eg. find all files that start with 'GHG' in all subdirectories of the chosen path. Each flight may have multiple versions of the same data file, so make sure to select only the latest version, which is indicated by a higher number at the end.

Example 1.3. Selecting files using `toolpac.readwrite.find`
Use `find_dir` and pyffind_file to create a list of paths to all GHG measurement sets taken on CARIBIC flights in 2012. As there were 36 flights in 2012 and GHG data was taken on each of those flight, you should obtain 36 items. Open one of the GHG files in a text editor to see the file structure and the information contained in it. You should be able to identify columns for the time of the measurement (expressed as time in seconds after midnight of the departure day in UT), the longitude, latitude and altitude/pressure, as well as the mixing ratios of the greenhouse gases themselves.

Example 1.4. Getting data using `FFI1001DataReader`
Extract the data from each file by calling the `toolpac.readwrite` function `FFI1001_reader` on each of the files identified in the last step. The function returns a `FFI1001DataReader` object, which contains all the information from the file you read in. By setting the `xtype` parameter to `'secofday'`, the time given will automatically be converted to datetime objects. Set the function parameter 'df' to 'True' to create a pandas dataframe, which contains all the information you will need for these exercises. Concatenate the data from each flight's dataframes into one using the function `pd.concat`. In order to be able to select data from individual flights later, you can extract the flight number from the file name and insert is as a new column.

1.4 Simulated climate model data - MOZART

Next you'll import SF_6 mixing ratios created by the model MOZART from the netCDF file `RIGBY_2010_SF6_MOLE_FRACTION_1970_2008.nc`. The file is available on the network server under `\\141.2.225.99\exchange\MODELL\MOZART\acp-10-10305-2010-supplement`. The package `xarray` will allow you to easily get the data from this file type. To get started, have a look at the official documentation at `https://docs.xarray.dev/en/stable/`.

Example 1.5. Using `xarray` to get MOZART data
Make sure to either open the file using a `with xr.open_dataset() as ds:` statement or explicitly close the file after importing the data. Define your function to accept a specific year as a parameter. The function `xr.open_dataset` returns an *xarray dataset* (ds), which is a mutli-dimensional framework of storing data and general attributes as well as attributes for each variable. You can easily see the dimensions of the dataset by calling `ds.dims` and all coordinates and variables by calling `ds.info`, where ds is a placeholder for the name of the dataset you created. Note that the coordinate 'Hybrid Sigma Level' is a topographic measure of altitude. To compare MOZART and CARIBIC data, you'll want to look only at level 995.0 (index 27). Similarly select only the year given in the function call from the data. From your dataset, you can obtain a multi-indexed dataframe by calling `.to_dataframe`. The coordinates of the netCDF file will be stored as indices, more information on this is available in the pandas documentation online. You will want to keep both the dataset and the dataframe on hand, as you may find yourself wanting to use one or the other depending on the context.

2 Aggregating data onto common time or grid

When comparing different sets of data, the spatial or temporal resolution of the available data will not necessarily coincide between them. In these examples you will be calculating the monthly averages at Mauna Loa and Mace Head observatories, as well as averaging CARIBIC and MOZART data onto grid cells of size 5×5 degrees.

Be aware that there may be missing data for some time periods, for example CARIBIC flights may not have taken place in every month of the year and some substances may not have been taken continuously on the measurement stations. To correctly calculate the average mixing ratios without including this missing data, you need to think about the format of your missing data points. You can then replace these values in your dataframes and setting them to `np.nan`, which is a `float`. To calculate the minima and maxima of arrays that have nan-values, make sure to use `np.nanmax() / np.nanmin()`.

Example 2.1. Monthly mean of Mauna Lao, Mace Head and CARIBIC data
From previous exercises, you should have obtained data from these three sources in the form of pandas DataFrames, with the index being datetime objects corresponding to the measurement time. Define a function that will take a DataFrame and return a new one with monthly averaged data. For this you will first box your data to months using `pd.PeriodIndex`, then calling `df.groupby` on the result. Then return the monthly averages with `df.mean()`.

Remember that you downloaded data from Mauna Loa both for daily and monthly averages. You can now verify that your code is working properly by calculating the monthly mean from the daily values and comparing it to the monthly values you downloaded. The timestamps of these means may not coincide, depending on how you are handling you data. Keep this in mind for plotting both on the same graph later and adjust your code if necessary.

Example 2.2. CARIBIC 1D binning: `bin_1d`
Find the average SF_6 mixing ratio as a function of latitude or longitude on grid cells of 5 or 10 degrees. Open `toolpac.bin_1d_2d` in a text editor and look at the parameters and attributes of the output of `bin_1d`: A `Bin1d` instance is created from data for the dependent (here this would be SF_6) and independent (here either latitude or longitude) variables, the total extent over which you want to bin your data over and the size of the bins. Important attributes of this resulting object are `vmean`, the average value of your dependent variable, and `xmean`, the mean of your dependent variable, which corresponds to the midpoint of your grid cell. `Bin1d` also returns the standard deviation and median of both x and v, however this will not be relevant here.

Example 2.3. CARIBIC 2D binning: `bin_2d`
Calculate the average values of SF_6 in CARIBIC data for a grid size of 5x5 degrees using `bin_2d` from `toolpac.calc.bin_1d_2d`. The output of this function will be a Bin2d object as defined in the source file. Make sure that the CARIBIC SF_6 mixing ratios (v), latitude (x) and longitude (y) are all arrays of equal length. Set `xmin / xmax` to the minimum and maximum latitude of your data, and equally for longitude. Choosing a grid size of 5 degrees for both directions, now return the binned CARIBIC SF_6 data with `bin_2d`.

Example 2.4. MOZART 1D and 2D binning
Repeat the steps from Examples 2.2 and 2.3 for MOZART data.

3 Plotting ground-based measurements and monthly averages over time

You'll now be using the library `matplotlib` to display the measurement data. Have a look at the documentation to get started: `https://matplotlib.org/stable/api/pyplot_summary.html`. Depending on the data, different functions will be of use here - you may create a scatter plot to display measurements from a fixed location, but a 2D plot to show all available data from a global climate model. Before going further, make sure that you have written functions that return the information from the data files in a usable format such as dataframes or datasets.

On a general note, you can create subplots with `fig, ax = plt.subplots()` and specifying either the number of columns or rows or both. You can let the subplots share a y- or x-axis by setting the parameter `sharey` or `sharex` to `True`.

Example 3.1. Plotting the Mauna Loa and Mace Head datapoints
Create scatter plots from the data at Mauna Loa and Mace Head. Display the measurement time on the x-axis and the measured mixing ratios on the y-axis. You can display multiple datasets in one figure, or you can create a new one for each - just make sure to call `plt.show()` before defining a new figure. Try customising your figure with the parameters `color, label, marker, etc`. To give your axes labels, use `plt.ylabel` and `plt.xlabel`. By calling `plt.legend()`, a legend will automatically be created using the labels you've given your data series.

Example 3.2. Plotting monthly means on top of the Mauna Loa and Mace Head data
Now plot the monthly means of the Mauna Loa and Mace Head data that you've calculated in Example 2.1 on the same plot as the data from the previous example. Note that depending on the way you handled the timestamps for those sets of data, you may find that one monthly average is set to the start of the month and the other to the end. This can create an offset of a full month for the same value and distort your plot. You can solve this by setting the time of all averages to eg. the 15^{th} or the 1^{st} of the month, or displaying the average as a horizontal line spanning the entire month with `.hlines`.

4 Plotting of timeline and 1-dimensional averages of global data

Example 4.1. 1D Plotting of CARIBIC data
Plot your CARIBIC data over time on a scatter plot, making sure to label your axes. Should your time axis have overlapping labels, you can to use `fig.autofmt_xdata()` to automatically format your dates (this only works if you have created your figure as subplots using `fig, ax = plt.subplots()`). If you want to plot several years, you can loop through both your data and list of years using `for data, year in zip(data_list, year_list)`. In the i^{th} iteration of the loop, 'data' and 'year' will then correspond to the i^{th} item in data_list and year_list, respectively. Depending on where you initiate and show your figure, the data from different years will either be plotted on the same graph or separately.

Example 4.2. 1D Plotting of MOZART data
The xarray dataset you created in Example 1.5 has predefined plotting methods, which you will now use to display mixing ratios over longitude / latitude. Select the specific year and latitude you want to display by calling `.sel()` on the SF_6 DataArray and choosing a specific time and latitude. Set `method = 'nearest'` so that you don't have to input exact values here. Now plot the corresponding mixing ratio over longitude by calling `.plot.line(x=longitude`. This function is based on matplotlib.pyplot.plt() and you can therefore customise your plot with the same keyword arguments.

Now repeat the same for a plot of values at a specific longitude over latitude. You can select multiple values of year / lon / lat to be displayed on the same line plot by giving `.sel()` an array instead of a specific value. Now show plots for several longitudes and latitudes next to each other on one figure by using `fig, (ax1, ax2) = plt.subplots(ncols=2, sharey=True)`. You can choose where your data is displayed through the 'ax' parameter of the `.plot.line()` function. Give your plots coordinate axis labels using `ax1.set_ylabel / ax1.set_xlabel` (use ax2 for your second subplot).

5 Grid-cell averages and 2-dimensional plotting of global data

In this part you will be plotting SF_6 mixing ratios as coloured cells on a latitude / longitude grid of various grid cell sizes. For plotting 2D data, you will be using the matplotlib function `imshow`: `https://matplotlib.org/stable/api/_as_gen/matplotlib.pyplot.imshow.html`. To customise your colour gradients and to add colourbars to your graphs, you will be using colormaps (`matplotlib.cm`) and `matplotlib.colors.Normalize`. Colorbars can then be created with the module `matplotlib.cm.ScalarMappable`. By default, its limits will be the limits of the data you are showing. To allow better comparison between data sets, however, setting a common scale is necessary. You can either set the values that correspond to the colormap limits directly in `imshow()` through `vmin` and `vmax` or by normalising the colours with `norm = Normalize(vmin, vmax)`. This norm can be used as a parameter in `imshow()` as well as in the plotting methods of an xarray dataset. For an overview of colour customisation options in matplotlib see here: `https://matplotlib.org/stable/tutorials/colors/index.html`.

Example 5.1. 2D Plotting of binned CARIBIC data
Create a 2D plot of the CARIBIC data you mapped onto a 5×5 grid in Example 2.3. Display the grid cell averages of the mixing ratios, which is the attribute `vmean` of the Bin2D instance you obtained then. Create a colormap and Normalize instance with your chosen mixing ratio limits. You can now use `plt.imshow()` with the data values, the colormap, the norm, and setting the 'extent' to the minima and maxima of latitude and longitude. To avoid your data being upside-down, also set the argument `origin='lower'`. Now add a colorbar to your graph with `cbar = plt.colorbar(ax=plt.gca()`, where the `ax` parameter has been set so that your colorbar is plotted on the same figure. You can adjust the orientation and thickness of your colorbar through arguments here too. Now add a label to the colorbar using `cbar.ax.set_xlabel()`, set labels for your x- and y.axes and display the plot with `plt.show()`.

Example 5.2. 2D Plotting of MOZART data
You can plot MOZART data on a 2D grid with an xarray dataset method. Create colormap and Normalize instances the same way as for the CARIBIC data in the previous example, then pass these arguments to `df.SF6.plot()`. By choosing the same limits for the colormap in this plot and in Example 5.1, you can easily compare the mixing ratios between measurements and simulation for the same years.

Example 5.3. Adding coastal outlines to global plots
For analysis and geographical orientation, now add a world map outline to your 2D plots using `geopandas`. Get the geospatial data with `world = geopandas.read_file( geopandas.datasets.get_path('naturalearth_lowres'))`, then plot the outlines using `world.boundary.plot()`. To display this on the same plot as your data, you need to set the axes parameter within `.plot()` to `ax = plt.gca()` or equivalently setting it to an `Axes` instance created using `fig, ax = plt.subplots()`. Within the plotting function, adjust parameters such as linewidth and color until appropriate.

You will notice that your plots for MOZART and for the world outline are do not have the same longitudinal coordinates: values of longitudes in MOZART data range from 0 to 355 degrees east, however they are symmetric around 0 for CARIBIC and the geopandas world outline. To compare these data sets, you will therefore need to find a way to remap the MOZART data so that eg. longitude 300°E is mapped to -60°E. One way of achieving this would be by using list comprehension (for an introduction see here: `https://www.geeksforgeeks.org/python-list-comprehension/`). Format your longitudinal data as an array, then using list comprehension return the first half of the array as is (select using the condition: `if value < 180`). Subtract 360 from the second half of the array before adding the two parts together again. Setting the longitude of your original dataset to these new values should now allow you to display the world outline and model data with the same coordinates.

6 Identifying outliers

With the toolpac module `.outliers.outliers` you can separate time series mixing ratio data into a *baseline* and *outliers*. An initial baseline is created from the complete data set according to the fit function given, then outliers are identified with reference to it. These fit functions are defined in `outliers.ol_fit_functions`, and you will be using a quadratic fit named `simple` for SF_6 mixing ratios. The function `find_ol` identifies outliers above, below or on both sides of the baseline, depending on the value of the `direction` argument. The function output consists of 4 items: (1) a list indicating whether or not a value of your input array has been flagged as an outlier, (2) an array of the distance between corresponding data point and the baseline, (3) a Boolean value indicating if a warning was encountered and (4) the parameters of the baseline fit function.

Example 6.1. Finding and plotting outliers
Find outliers in CARIBIC data using a quadratic baseline fit (`simple`), the measurement time series and corresponding SF_6 mixing ratios. Calculate this for all the directions (`'np', 'p', 'n'`) and visualise your results by setting `plot=True`. Now repeat this exercise with Mauna Loa and Mace Head data.

7 Calculating time lag

To calculate the relative time lag of CARIBIC air samples, compare the SF_6 mixing ratios to those from ground-based observations. As the reference, you will take a 20 year data range of measurements taken

from 2000 to 2020 at Mauna Loa. Create a DataFrame using the data from 2000 to 2020 and calculate the monthly averages to smooth out the data. The reference time series needs to be complete for all months, therefore you will need to resample your data using `df.resample('1M'` to fill in rows with the missing months, then interpolate the data in between measured values.

Example 7.1. Time lag using `calculate_lag` from `toolpac.age`
The function `calculate_lag` requires data in the form of arrays, where time series values are given as fractional years. To convert to this from datetime objects, use the appropriately names toolpac function `..conv.datetime_to_fractionalyear`. Create arrays of the time series in the form of fractional years and of the mixing ratio measurements using `np.array()` and then calculate the lag of a single year of CARIBIC data with respect to measurement at Mauna Loa. To visualise your results, plot the output of `calculate_lag` on a simple scatter plot, where the CARIBIC measurement time will be on the x-axis.