Project P04: Paleoclimatology

Collaboration with the LSCE laboratory

Conducted by:
Pasha Alidadi
Linn Habberstad
Teddy Tonin
January 21, 2024

Abstract

The project in data science on "Paleoclimatology," in collaboration with the LSCE laboratory, focuses on modernizing the Analyseries software, developed in the 1990s for time series analysis in paleoclimatological studies, particularly ice core data interpretation. Key enhancements include transitioning to Python for compatibility and incorporating modern libraries like PyQT5 for user interface improvements. The project tackled challenges like software development, data representation, user interface enhancement, more concise and tidy code structures and implementing new functionalities such as Multiseries for simultaneous data recalibration. The report discusses results, challenges, and the future outlook, emphasizing user-friendliness, efficiency, and expanding analytical capabilities for open-source distribution.

Table of contents 1

1. Introduction

Analyseries, a time series analysis and comparison software created by Didier Paillard is a tool employed by the Laboratory of Climate and Environmental Sciences (LSCE). Notably, Elisabeth Michel and many others utilize it for paleoclimatological studies. One of the methodologies employed by LSCE is the science of ice cores. By analyzing ice cores, scientists can interpret climate signals embedded within them. These cores, extracted from polar regions and glaciers, act as time capsules, preserving valuable data about Earth's past atmospheric conditions. The analysis of these cores provides insights into the atmosphere, temperature trends, and the occurrence of volcanic eruptions over millennia. This valuable data was analyzed within the software developed in the 1990s, only being compatible with older MacOS versions, rendering it incompatible with recent macOS or Windows-based computers and thus a pain point for many researchers globally. In response, the LSCE has engaged CentraleSupélec students to develop a new version of AnalySeries. This updated version, maintaining the same functionalities, is being designed as open-source software in Python. It incorporates various modern libraries, such as the PyQT5 library for the UI, to facilitate user-driven enhancements. The continuation of updating and improving the application has given it several new functions and efficiencies since the last iteration in Spring 2023.

2. Background

2.1 Paleoclimatological Data

Researchers in paleoclimatology at the LSCE and various other research centers globally are frequently engaged in the analysis of sediment cores because their compositions bear witness to past climatic phenomena. These sediment cores undergo numerous physicochemical measurements like rates of dioxygenases, carbon isotopes, and other chemical compounds that plot out the history of the climate and its changes over time. During the project, the group members visited the laboratory to see cutting-edge instruments like mass spectrometers that allow the separation of isotopes in marine samples. During the project this fall, an additional visit was scheduled to see the work of Elisabeth and Didier, scientists at the LSCE. The goal was to understand the functionalities and agree upon the spaces for improvements, accounting for the past work. Figure 1 displays the utilized data within the Analyseries application.

[3]:

	depth ODP849	d18Oforams-b	d13Cforams-b	depth ODP846	d18Oforams-b.1	d13Cforams-b.1	Time (ka)	Benthic d18O (per mil)	Standard error (per mil)
0	7.0	3.66	0.21	12.0	3.380	0.14	0	2.588	0.031
1	17.0	3.49	0.08	23.0	3.460	0.01	1	2.588	0.037
2	28.0	3.31	0.19	33.0	3.765	-0.08	2	2.539	0.031
3	45.0	4.17	-0.15	43.0	4.140	-0.17	3	2.646	0.027
4	55.0	4.69	-0.27	53.0	4.470	-0.21	4	2.655	0.033
...	...	...	...	...	...	...	...	...	...
2111	NaN	NaN	NaN	NaN	NaN	NaN	5305	2.154	0.037
2112	NaN	NaN	NaN	NaN	NaN	NaN	5310	2.150	0.093
2113	NaN	NaN	NaN	NaN	NaN	NaN	5315	2.202	0.073
2114	NaN	NaN	NaN	NaN	NaN	NaN	5320	2.265	0.089
2115	NaN	NaN	NaN	NaN	NaN	NaN	5325	2.051	0.051

2116 rows × 9 columns

Figure 1: Dataframe containing several isotopes and corresponding depths and time for the ice core analysis.

For a better understanding of the data, the metadata file in Figure 2 can provide a more thorough explanation.

Depth and Depth (m) EDC	Depth at which ice core sample was obtained
δ18Oforams-b, δ18Oforams-b.1	Values for the oxygen isotope found in a sample
δ13Cforams-b, δ13Cforams-b.1	Values for the carbon isotope found in a sample
Time (ka)	The time in thousands of years (ka) before present
Benthic δ18O (per mil)	Values of oxygen isotope ratio at the bottom of the ocean
Standard error (per mil)	An indication of the precision of the measurements
Ice age (a 1950) AICC/AICC.1	Age of the ice at a particular depth
Gas age (a 1950) AICC	Age of the gases trapped within the ice core
EDC CH4 [ppbv]	Concentration of methane
CO2 composite Bereiter	Concentration of carbon dioxide
nssCa800ka	Non-sea-salt calcium concentrations
EDC deuterium excess (deut cor)	Deuterium excess data
Depth/ice age m EDC	Ratio between the depth and the age of the ice

Figure 2: Metadata extract in a table format for better representation.

Each measurement in the data links to a time scale or depth scale, which allows for the analysis of the chronology of marine sedimentation. Afterward, researchers can interpret these marine sediments through comparison from a core with reference data (Thompson, 2000). The study concludes by establishing the existence of different sedimentation rates and the definition of the geological specificities of the specific location for the extracted ice core.

2.2 Analyseries software timeline

2.2.1 The initial Analyseries software

The development of Analyseries in the 1990s was a response to the complex task of comparing two time series. As illustrated in Figure 1, the paleoclimatological data used in the analysis of ice cores are represented either in terms of time or depth and essentially can be boiled down to studying time series. To facilitate the analysis of time series data, Didier Paillard designed the Analyseries software. The innovation enabled the numerous data manipulations necessary for dating sediments to be done simultaneously instead of just processing them independently.

The initial version of Analyseries carried essential functionalities, such as “linear” (Linage) or “polynomial” (Splinage) dating interpolation. The dating of an ice core was conducted by temporally recalibrating the core’s scale about a known reference scale. Furthermore, the software provided access to all types of mathematical analysis (Math). The mathematical analysis includes functions such as the user being able to simulate their time series and perform advanced statistical analysis on their series. The user could also access a statistics menu (Stats) that calculates averages, medians, and variances on selected data series. Additionally, the application detected cycles present in multiple time series by performing the Pearson R2 test and studying intercorrelations. Lastly, it offered functionalities such as filtering, the study of periodograms, and fitting to try to conform the series to a given model whose parameters are obtained by maximum likelihood.

2.2.2 Project scope and progress

Since the initial version in the 90s, the Analyseries software has become more extensively utilized by the global scientific community for various applications, necessitating significant updates for its transition to open-source. The request from the LSCE emphasized the need to improve the software’s readability and documentation of the code for open-source use. Therefore, the focus of the project was on refining the code, upgrading the user interface for intuitive use, and adding detailed documentation to facilitate future development. Recognizing the need for compatibility with modern technology, the previous team undertook the task of recreating Analyseries. The original version was coded in C++ and limited to older versions of MacOS, so the software was recreated using Python and modern libraries. This redevelopment made Analyseries accessible on contemporary MacOS and Windows PCs.

During the project in spring, the previous team focused on the emphasized areas of the LSCE. The effort targeted translating the old code into a modern language using commonly used libraries within Python while ensuring the user has a functional user interface. By modularizing the code, the previous team ensured that future development would facilitate adding or removing features when needed. As it was the

start of the project, the team managed to implement a small number of functions into the new version of Analyseries, as illustrated in Figure 3.

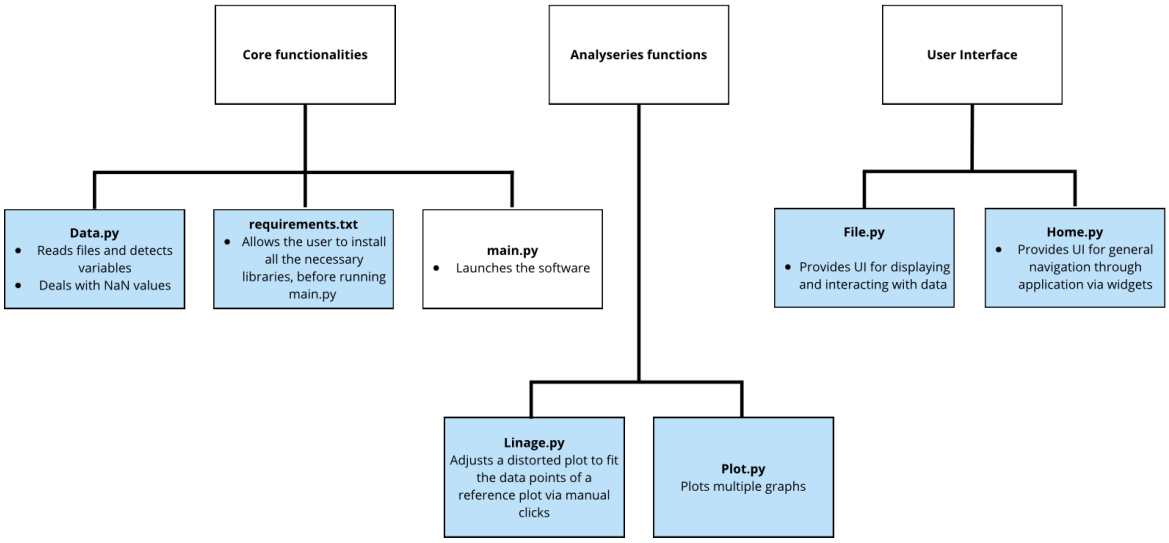

Figure 3: File structure of the application after the update in spring, where the light blue files represent the updated functions.

Special attention was given to the "Linage" function, enhancing it to be more user-friendly and intuitive. The team developed the function so the software could open multiple files simultaneously - a feature not available in the original version. They also tackled the issue of "cross-clicks" within the Linage function, which previously could lead to errors in temporal recalibration. These comprehensive efforts transformed Analyseries into a more versatile and accessible tool for scientific research.

3. Methodology

Building upon the foundation laid by the previous team, the project this fall started with a thorough evaluation of the initial Python version developed during spring. The overall objective was twofold: to align the software with the LSCE's specific needs and to further modernize the software to enable it to become an open-source, focusing on enhancing user-friendliness, efficiency, and the breadth of its features and flexibility. In practice, a common approach for modernizing legacy software to modern environments is entailed within the frameworks of modern cloud providers, such as Amazon Web Services. One such method is called "Re-architecting". It refers to redesigning the application's architecture to align with modern standards and could be applied to legacy software (R. Wise, L. Sheppard, J. Huggins, and D. Broadwell, 2015).

Although this version uses no native cloud features, the framework is a reasonable guide to achieving the goal and proves that it does follow the industry's best practices.

The initial comprehensive analysis of the current version was followed by detailed user feedback sessions to understand practical challenges, the usability, and the effectiveness of the software to identify further areas to develop. During the procedure, several flaws were identified in terms of data representation, file processing, and code efficiency. Additionally, some limitations with the functions within the software were identified, both from the initial 1990s version and in the version iterated this spring. Lastly, further features that had not yet been explored were identified and examined in terms of usability. Regarding the flaws in the current version of the application, the team was dedicated to eliminating these, through refining the code that the defaults stem from. Furthermore, for the functionalities that had not yet been implemented in Python and the general areas of improvement regarding the functions that were found in both the initial version and the refined version, the team allocated time to understand the mathematical logic behind the functions, and consequently added and refined the code in question. Regarding code efficiency, the team dedicated time to refactoring the existing code to make it more concise, efficient, and scalable. All improvements mentioned above typically involved code reviewing, refactoring, and refining sessions, as well as feedback sessions when improvements had been implemented to validate them.

As mentioned briefly above, there was a specific unexplored area that was given a lot of attention during the project duration. As contemplated by the spring team, the current team decided to implement a Multiseries function. The purpose of the function is to facilitate the simultaneous recalibration of multiple data visualizations. In a conventional agile software development setting, this prototype would be incorporated in a constant feedback loop with user trials to ensure the best possible product tailored to our target group (Al-Saqqa, Sawalha, and Abdel-Nabi, 2020). However, in this specific case, the user trials only involved a demonstration to the researchers at LSCE. The methodology for incorporating this feature involved a phase of concept validation and iterative development. To reach the overall goals, it was necessary to understand the underlying file structure of the project, as the previous team had already developed it. As illustrated in Figure 4, the modular nature allows for quick code implementation and debugging, as each file has its specific purpose and each function has a separate file. Moreover, the files generally fall under one of three categories: core functionalities, which include file management and software launching, user interface that deals with how the user interacts with the software, and all the remaining functionalities of the old Analyseries application.

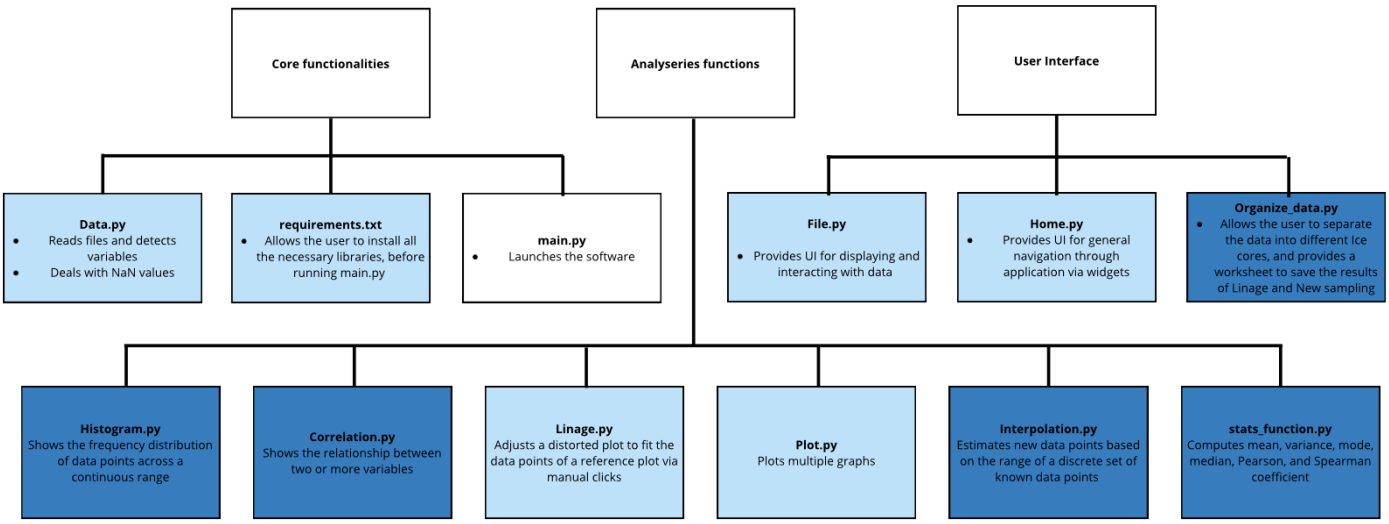

Figure 4: File structure of the application. Light blue files are the ones that have been modified from the previous iteration and dark blue files are completely newly added functions.

4. Implementation and results

4.1 User Interface

When focusing on enhancing the user-friendliness of the spring version, the team used the initial software as a source of inspiration to enhance the user interface. Therefore, a "worksheet" feature was implemented by the team. In the Python version recreated this spring, the application could display elements but not offer data management or saving capabilities. The main purpose of this feature was to separate the data between different ice cores in the same manner as it is done in the initial version of the application. The implementation was based on the idea that the data frame storing the file was divided into ice cores, and each time an axis is monotonic it is likely the scale of the corresponding ice cores. This was done through a function called is_monotonic_increasing as displayed in Figure 5 below.

```
def is_monotonic_increasing(array):
    return np.all(np.diff(array) >= 0)
```

Figure 5: Code snippet from the worksheet feature.

The worksheet was then implemented as a window that displays the organization of the data frame (into ice cores). The user then chooses to either agree to keep the organization as such or apply another way of structuring by filling in the boxes. The implementation resulted in an update where, when the user adds a new file, a dedicated window opens as illustrated in Figure 6, which allows the user to match each data point to the corresponding ice core. The identification process of the ice cores is based on finding the

columns that display a monotonic sequence, generally representing depth or time. After completion of the matching process, the user confirms with an “OK” in the application and the software will subsequently display data and allow for data management, similarly to the initial version as illustrated in Figure 7.

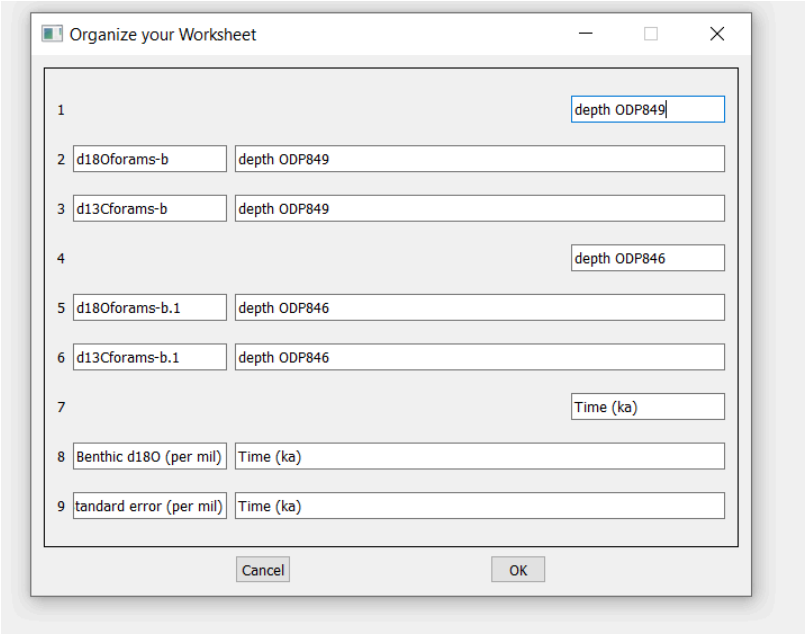

Figure 6: The worksheet window that opens when a user adds a new file.

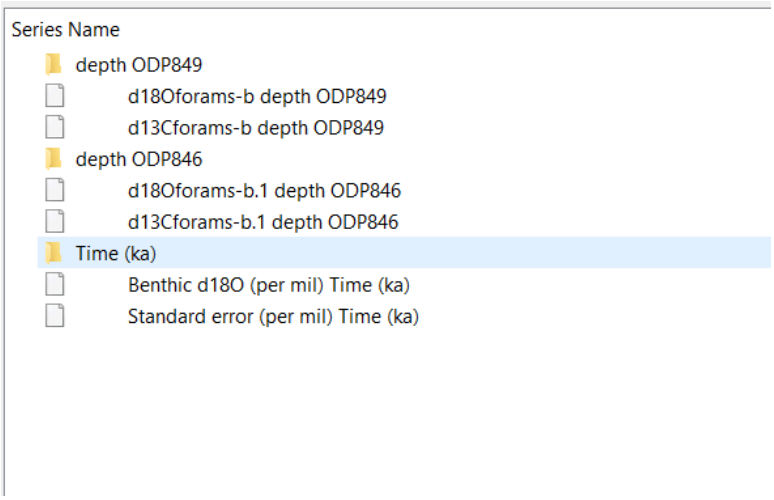

Figure 7: Display of the data that allows for data management. The folders represent ice cores, and the files within the folders are the physical measurements.

The other enhancement that was done to the User Interface, also inspired by the original version of Analyseries, was an update of the application menu. This was implemented by defining the menu in the file home.py and implementing the function create menu. The results are presented in Figure 8 below.

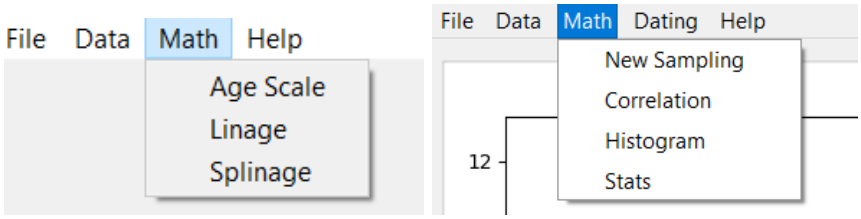

Figure 8: The before and after display of the application menu in the software. The before is illustrated to the left and corresponds to the version iterated in spring 2023. The after is what the team implemented during this fall.

Lastly, to enhance the overall user experience, an update implemented was to allow for multiple windows when plotting. In the previous versions, whenever the user would change a variable, a window that was already opened illustrating a plot would be overwritten. This was implemented by adding two more functions to the current class “Fenetre_graphe” and adding a new attribute in the function “self.graphWidget”, which is a list to keep track of the different graph windows as illustrated in Figure 9.

```
def update_graph(self, x, y, abs, ord):
    # Clear the current graph
    self.graphWidget.clear()

    # Update the graph with new data
    pen = pg.mkPen(color=(255, 0, 0), width=5)
    self.graphWidget.setLabel('left', ord)
    self.graphWidget.setLabel('bottom', abs)
    self.graphWidget.plot(x, y, pen=pen)

def closeEvent(self, event):
    self.isClosed = True  # Update the flag when the window is closed
    super(Fenetre_graphe, self).closeEvent(event)
```

Figure 9: Implementation of multiple graph window displaying.

The result of the implementation of the multiple graph displaying is shown in Figure 10.

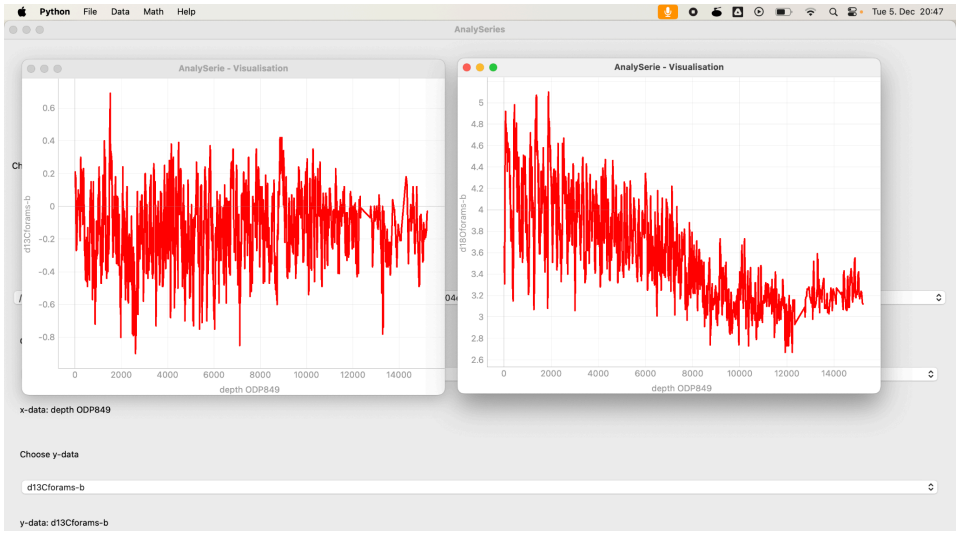

Figure 10: Ability to display multiple graph windows.

4.2 Data representation and file processing

During the dedicated time to eliminate the flaws of the current version, two main issues were tackled. The first one regarded inadequate data representation, more specifically when uploading raw data in a ".txt" format. The application failed to display variable names correctly via the "read_data function" within the Data.py file (see Figure 4), resorting to generic titles like "Unnamed: X", as seen in Figure 11. The issue was addressed by modifying the software's code in two key steps: refining the code to accurately recognize and display header rows with column names and implementing a data cleaning process to filter out columns containing NaN values.

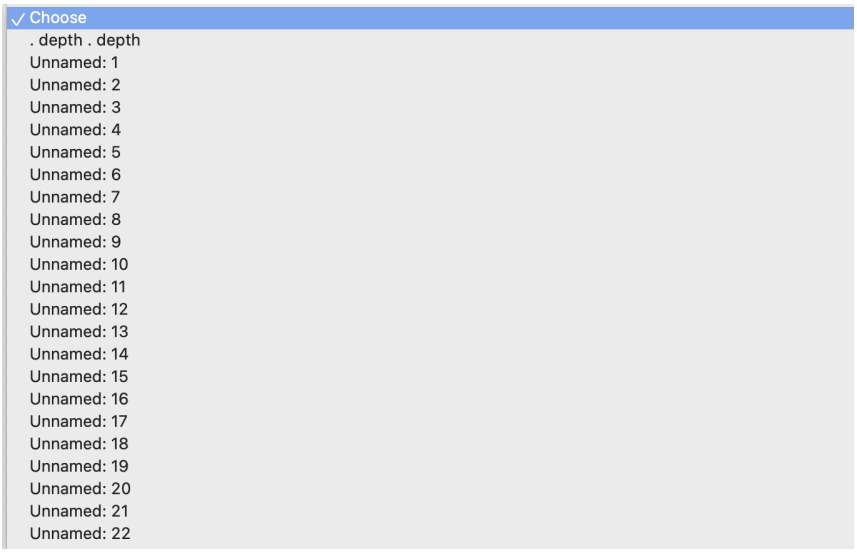

Figure 11: Automatic detection of variables not functioning.

The implementation resulted in the following update, illustrated in Figure 12.

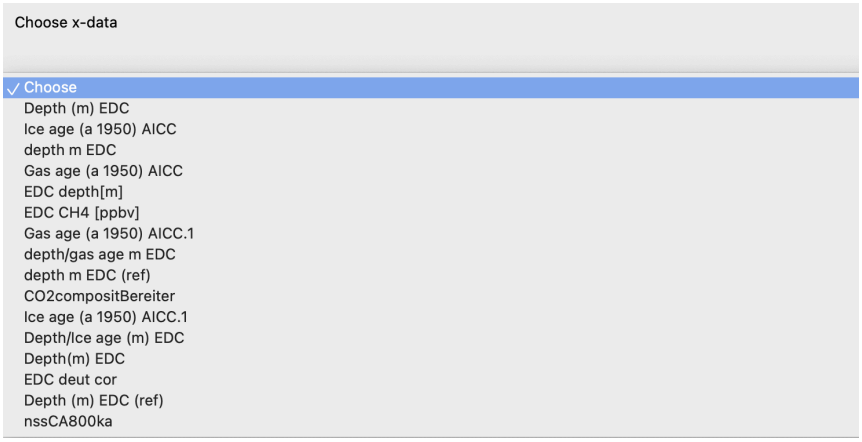

Figure 12: The updated automatic detection of variables functioning.

The second flaw identified during the evaluation of the current version was the software's inability to process ".xlsx" files. The solution implemented was to develop a method to automatically convert ".xlsx"

files to ".txt" format so it could detect the variables as normal. Illustrated in Figures 13 and 14 below is the code before and after the implementation of the solution.

```
def read_data(filepath):
    if filepath.endswith('.xlsx'):
        # Excel file
        dtype_dict = {col_name: str for col_name in pd.read_excel(filepath, nrows= detect_header(filepath))}

        df = pd.read_excel(filepath, header=detect_header(filepath), dtype=dtype_dict)
```

Figure 13: Display of the previous implementation that was unable to read ".xlsx" files.

```
def read_data(filepath):
    if filepath.endswith('.xlsx'):
        # Excel file
        dtype_dict = {col_name: str for col_name in pd.read_excel(filepath, nrows=detect_header(filepath))}
        df = pd.read_excel(filepath, header=detect_header(filepath), dtype=dtype_dict)

        # Convert the DataFrame to a .txt file
        txt_filepath = filepath.rsplit('.', 1)[0] + '.txt'  # Change the file extension to .txt
        df.to_csv(txt_filepath, sep='\t', index=False, header=True)  # Save as a tab-delimited .txt file

        print(f"Converted Excel file to {txt_filepath}")
        df = df.dropna(axis=1, how='all')
        return df  # Return the DataFrame for further processing
```

Figure 14: Display of the implementation of the new method to convert and read ".xlsx" files.

4.3 Code optimization

Furthermore, focusing on the efficiency of the software, the team found that the coding was quite complex and untidy throughout the application. Whilst the code was much more efficient and easy to interpret in comparison to the initial version, the latest version still contained many areas for improvement. Therefore, a systematic refactoring of the application's code was undertaken, including removing redundant loops and improving the data access approach, notably within the Plot.py file. Figures 15 and 16 below provide one of many examples of the implementation of code optimization.

```
def push_choice_y(self, index):
    self.y = index
    y1 = math.inf
    i = 0
    while (y1 == math.inf) and i < len(list(self.df.columns)):
        if list(self.df.columns)[i] == index:
            y1 = list(self.df.columns)[i]
        i += 1
    self.y_values = list(self.df[y1])
    self.labely.setText("y-data: {}".format(index))
    if self.x is not None:
        self.graphe()
```

Figure 15: Previous untidy and inefficient code snippet in Plot.py.

```
def push_choice_y(self, index):
    if index in self.df.columns:
        self.y = index
        self.y_values = list(self.df[index])
        self.labely.setText(f"y-data: {index}")
        if self.x is not None:
            self.graphe()
```

Figure 16: The same code as in Figure 15 but optimized.

4.4 Stats Functions

The Stats Functions were part of the initial version of Analyseries but had not yet been deployed by the team during spring into Python. Therefore, they were incorporated during the fall into the updated version. This was done by implementing code with the mathematical formulas to calculate the mean, median, variance, Pearson's correlation coefficient, Spearman's correlation coefficient, and the probability of zero rank/linear correlation of a chosen variable of a dataset.

4.5 Correlation

The correlation function was also part of the initial version of Analyseries, but had not yet been deployed into Python. Therefore, the correlation function was implemented through a thorough creation of the mathematical function from scratch, which contains a lot of coding, to add to the updated version. In the broad sense, correlation is a measure of an association between variables. When data is correlated, it implies that the change in the magnitude of one variable will be associated with the same change in another variable. It can be a positive correlation, meaning the same, or a negative correlation, meaning the opposite direction. Several mathematical formulas and coefficients can be applied for different purposes, for example, Pearson's correlation coefficient or Spearman's correlation coefficient (Schober, Boer, and Schwarte, 2018). Equations 5 and 6 in segment 4.4 above display the mathematical formulas applied for those calculations. While implementing the correlation function into Python was one part of the update, the other part contained modernizing the function. Initially, the correlation function was redesigned so it could handle multiple files simultaneously. If multiple files are uploaded, the user can select which one of the variables to plot for the correlation. In contrast to the initial version, the user can also select one variable from one file and the other variable for the correlation plot from another file. Another implemented feature was that the user can display up to four plots at the same time, enabling comparison between them, as illustrated in Figure 17.

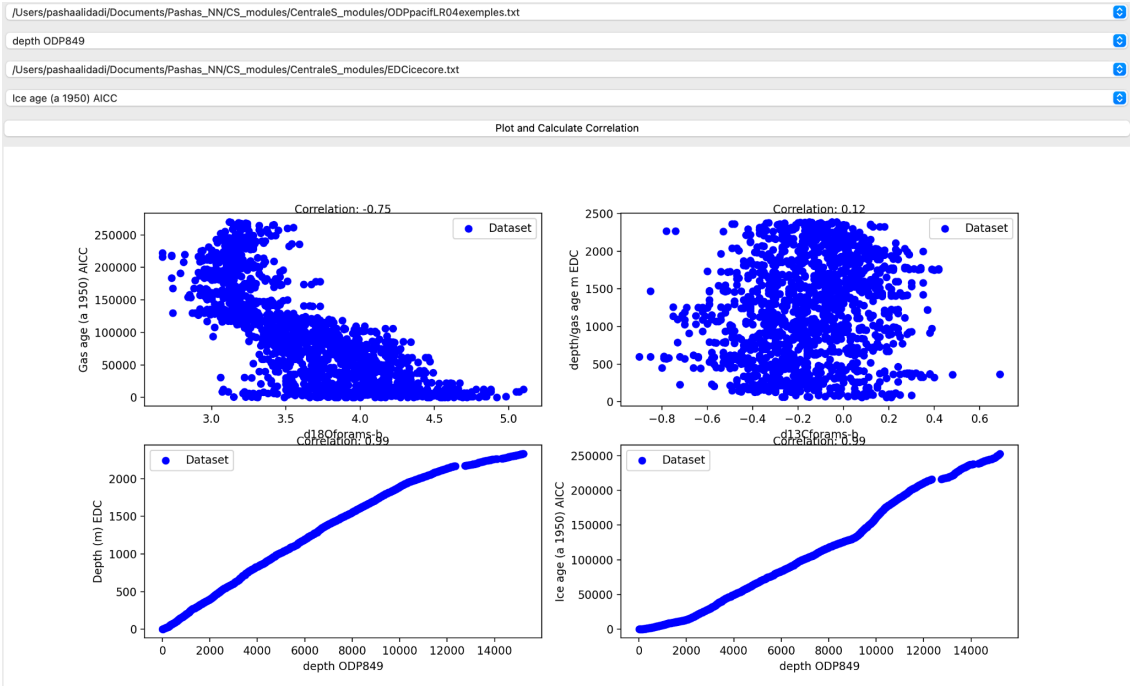

Figure 17: Four plots of the correlation displayed simultaneously.

Moreover, an important implementation was the introduction of an automatic adjustment feature that enabled the comparison of variables of different lengths. The outdated application could not perform this, as presented in Figure 18.

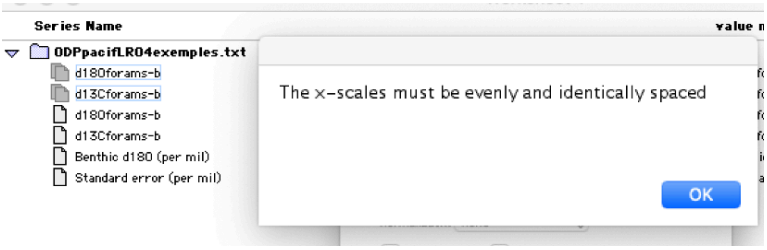

Figure 18: The old Analyseries version could not modify the variables to be correlated together.

The automatic adjustment feature was introduced through a function that standardizes the length of variables from different datasets by automatically shortening the longer variable to match the length of the shorter one. This is further supported by a warning window when this shortening operation happens so that the user is aware of this modification.

4.6 Histogram

The histogram function was another function that the initial version of Analyseries contained, but that had not been implemented by the team during spring. The Histogram function works in the same way as in the initial version where the user chooses between cumulative probability, probability density, and the specific values for the histogram that is then plotted as displayed in Figure 19.

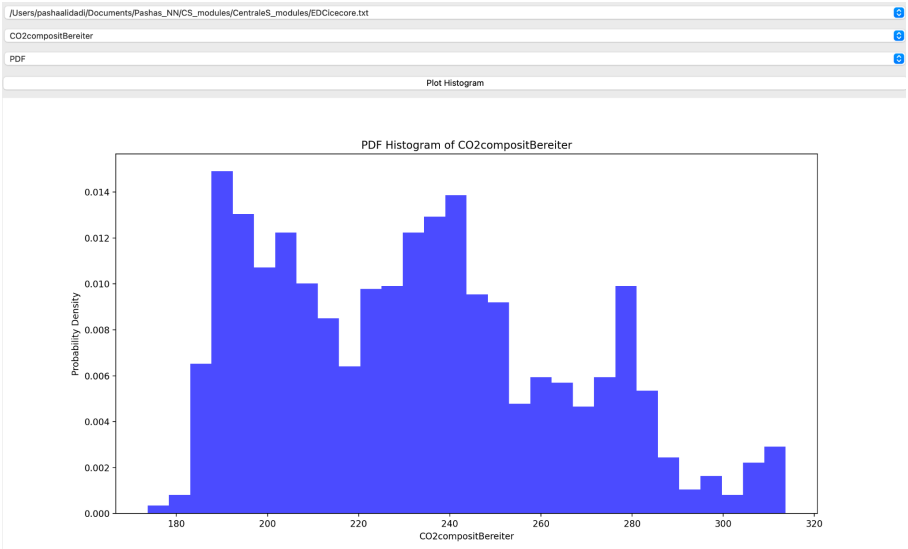

Figure 19: The implemented histogram function in the updated version of Analyseries.

During the implementation of the histogram function, an automatic adjustment feature was introduced in the same manner as for the correlation function.

4.7 Interpolation

Another function that was implemented during the fall, part of the initial version, was the interpolation function. The purpose of this feature is to provide a new sampling to a set of original data. The initial version of the software provides two types of interpolation: linear interpolation and interpolation through integration. During the fall, the team implemented both types in the updated version. Linear interpolation is used to estimate a value y between two known data points (x_0,y_0) and (x_1,y_1). The formula can be defined as follows:

$$y = y_0 + \frac{(x - x_0)(y_1 - y_0)}{(x_1 - x_0)}$$

where y is the interpolated value to find, x is the point at which you want to estimate the value and lastly (x_0,y_0) and (x_1,y_1) are the coordinates of the known data points (Sauer, T., 2012).

The method of linear interpolation assumes a straight line connecting the two known data points. The formula therefore finds the value of y at a given x by finding the weighted average of (y_0,y_1), as one can also see in Figure 20.

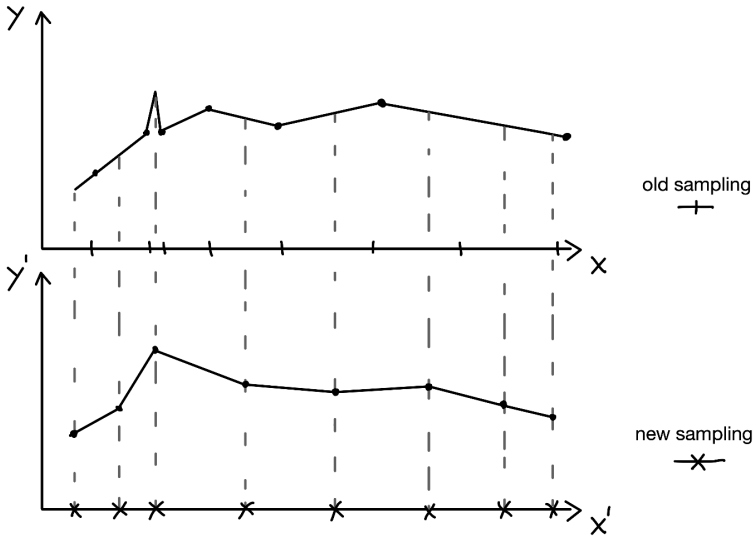

Figure 20: Own depiction of a linear interpolation.

The integration method operates a bit differently. The method is specifically useful in this case with environmental data, since the data can be very noisy. Dividing the abscissa into many sections helps with smoothing out the noise of the data while keeping the important trends of the graph.
It interpolates with the computed mean value of the function f over an interval near the data point t_a. The computed mean value of that integrated section provides a much better approximation of the function locally (Noor *et al.*, 2014).

After generating the integrals of these sections, one can choose between a linear, staircase, or cubic spline interpolation technique, each having different strengths and use cases. The staircase interpolation shines through its simplicity, requiring minimal computational resources, and its ability to maintain a constant value between data points, is useful for representing discrete steps or categories. However, its limitations include its discontinuity and potential data distortion, especially for continuous data.
Therefore, although the most computationally intensive, the cubic spline interpolation provides a smooth, continuous curve for accurate representation of data with natural variations and allows modeling complex non-linear relationships between data points (Sauer, T., 2012).
The mathematics behind the interpolation functionalities are coded inside the file named Interpolation.py. Figure 21 visualizes the basic understanding of how the interpolation through integrals works.

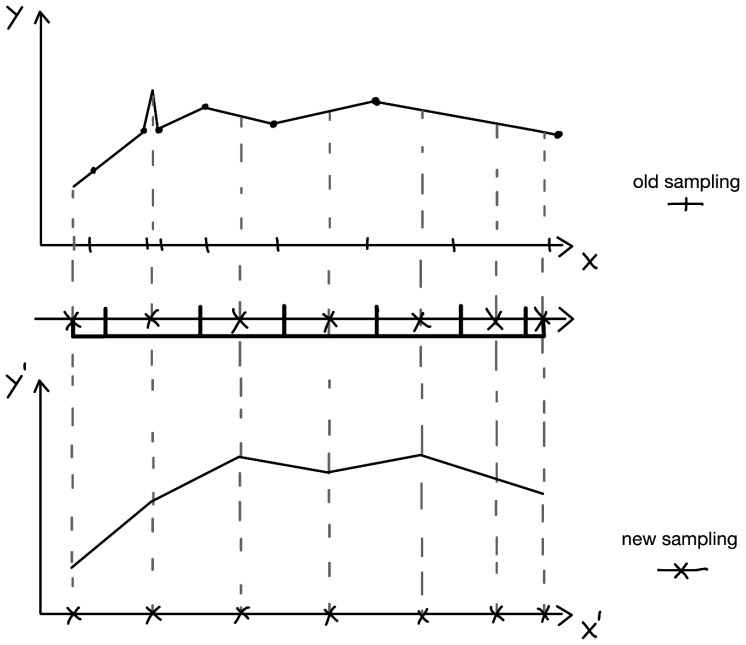

Figure 21: Own depiction of an interpolation via integrals.

When running the interpolation function in the updated version, the user receives the "Resampling" functionality as a small window, as in the initial version, for them to choose the data to sample (x and y). The user then decides between sampling evenly or applying the same sampling as other data. While Mathematicians and analysts prefer working with evenly sampled data, scientists in climatology often need data that are not evenly sampled. The purpose of the "using scale of" feature is to apply the sampling of another data (like the sampling of another ice core) instead of creating new evenly sampled data points. Furthermore, the user can choose the parameters for the sampling, the desired interpolation type, and the type of function as displayed in Figure 22.

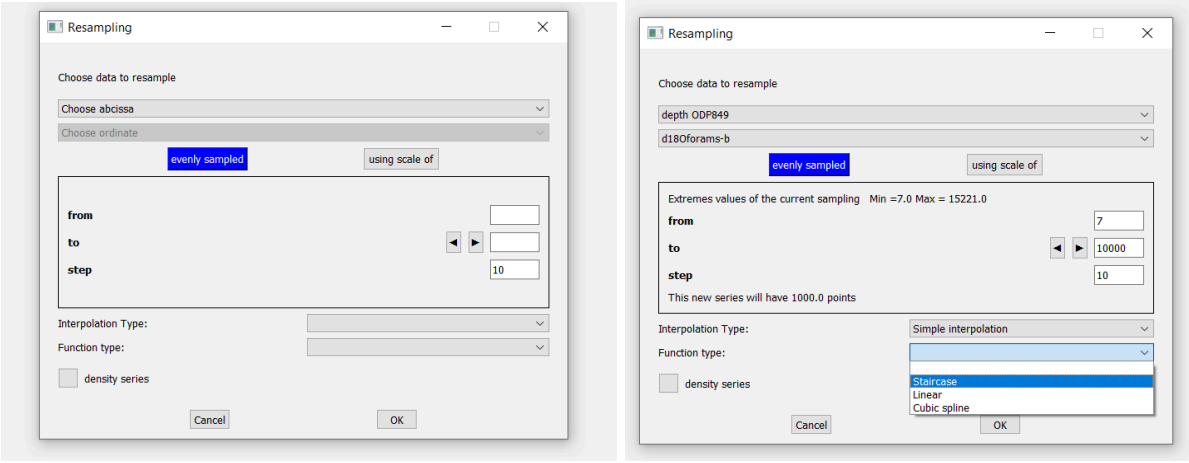

Figure 22: The window that appears when the user opens a new Resampling.

Once the user is done, they proceed with an “ok” and the plot appears in the same way as in the 1990s version as illustrated in Figure 23.

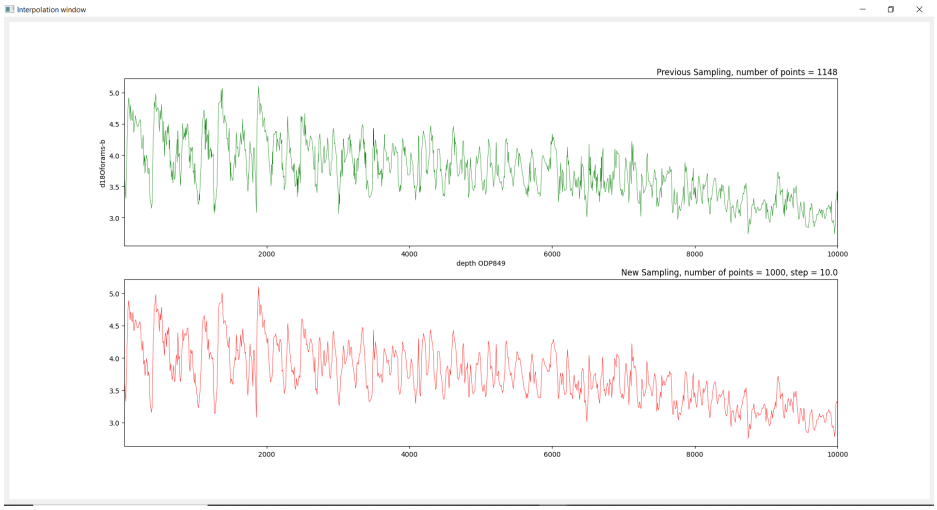

Figure 23: The plot that appears after the user has filled everything in the Resampling window and proceeded with “ok”.

4.8 Linage

4.8.1 Problem Formulation

The Linage function has been a central part of the software and was implemented in the Python version by the team this spring. Researchers at the LSCE use this function to recalibrate ice cores with reference data, this is the most basic use of the function. To mathematically frame this the two extracted ice cores from the same geographical area can be called **Ice Core** a and **Ice Core** b. From each of these ice cores measurements are extracted at a specific depth/time. Let’s call x_a the sampling abscissas for **Ice Core** a and x_b that of **Ice Core** b.

Now suppose that a variable of interest (composition in Oxygen for example) y_a is measured in **Ice Core** a, using the sampling abscissa x_a. The same variable of interest y_b is measured in **Ice Core** b using the sampling x_b. Since the ice cores are extracted from the same geographical location, the plots of (x_a,y_a) and (x_b,y_b) will show some great similarities, but due to errors, one plot could be distorted compared to the other.

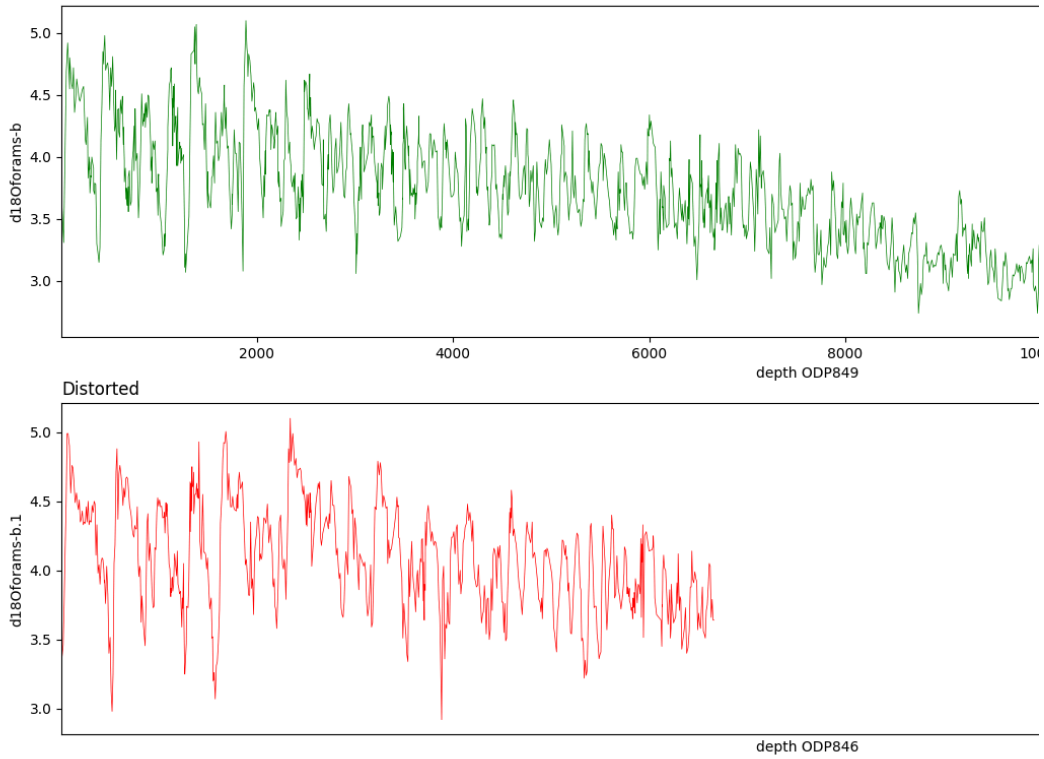

Figure 24: In Ice Core a = OPD49 plotting d18 using the sampling of Ice Core a. In Ice Core b = ODP846 plotting d18 using the sampling of Ice Core b. The plots show similarities but the cycle does not align.

Solving the issue with the alignment is the purpose of the Linage function. The solution is to create a linear interpolation between the points that should be aligned to make the cycles correspond between both ice cores examined. For example, p points are selected on each plot:

- In **Ice core** a also known as the *reference data*, the following points are selected: $((x_{i1}, y_{i1}), (x_{i2}, y_{i2}),.., (x_{ip}, y_{ip}))$, and in **Ice core** b also known as the *distorted data*, the following points are selected: $((x_{j1}, y_{j1}), (x_{j2}, y_{j2}),.., (x_{jp}, y_{jp}))$. Then the data of ice core b is recalibrated onto the data of ice core a.

- Now to compute the interpolation f, the pointers must satisfy the following requirement:

$$\forall\, k \in \{1, 2,..,\ p\},\ f(x_{i_k}) = x_{j_k}$$

- Linage performs a linear interpolation between the pointers, therefore the function f is solved for:

$$\forall\, x \text{ and } k \in \{1, 2,..,\ p - 1\} \text{ such that } x \in [x_{j_k},\ x_{j_{k+1}}],$$

$$f(x) = \frac{x - x_{i_k}}{x_{j_{k+1}} - x_{j_k}}$$

The interpolation function f is important because of the interpretability of its derivative. First, the abscissa of **Ice core** b can be recalibrated to that of **Ice core** a by using the interpolation function. The data of **Ice core** b is "plotted using the abscissa of the reference ice core". Second, the derivative of that function has an important meaning that is at the core of the Linage method. The distortion is interpreted as follows: sediments in **Ice core** b have accumulated at a different rate than sediments in **Ice core** a. The rate of that accumulation is different between sections of the ice core. If we suppose that the accumulation is uniform in the *reference* core then, in the *distorted* ice core, in the section between abcissas x_{j_k} *and* $x_{j_{k+1}}$ the sediments accumulated at the rate $\frac{x_{i_{k+1}} - x_{i_k}}{x_{j_{k+1}} - x_{j_k}}$. This interpretation justifies the linear interpolation. Other interpolations such as spline interpolation for Splinage are used in the Mac version, but they have different interpretations.

To observe the calibration what is left is to compute the distortion of abscissa x_b, by applying the age scale function. The data of **Ice core** b is then plotted as $(f(x_b), y_b)$.

In what follows Linage is applied to the data in Figure 24. Both plots should be aligned, so the Linage does it (illustrated in Figure 25).

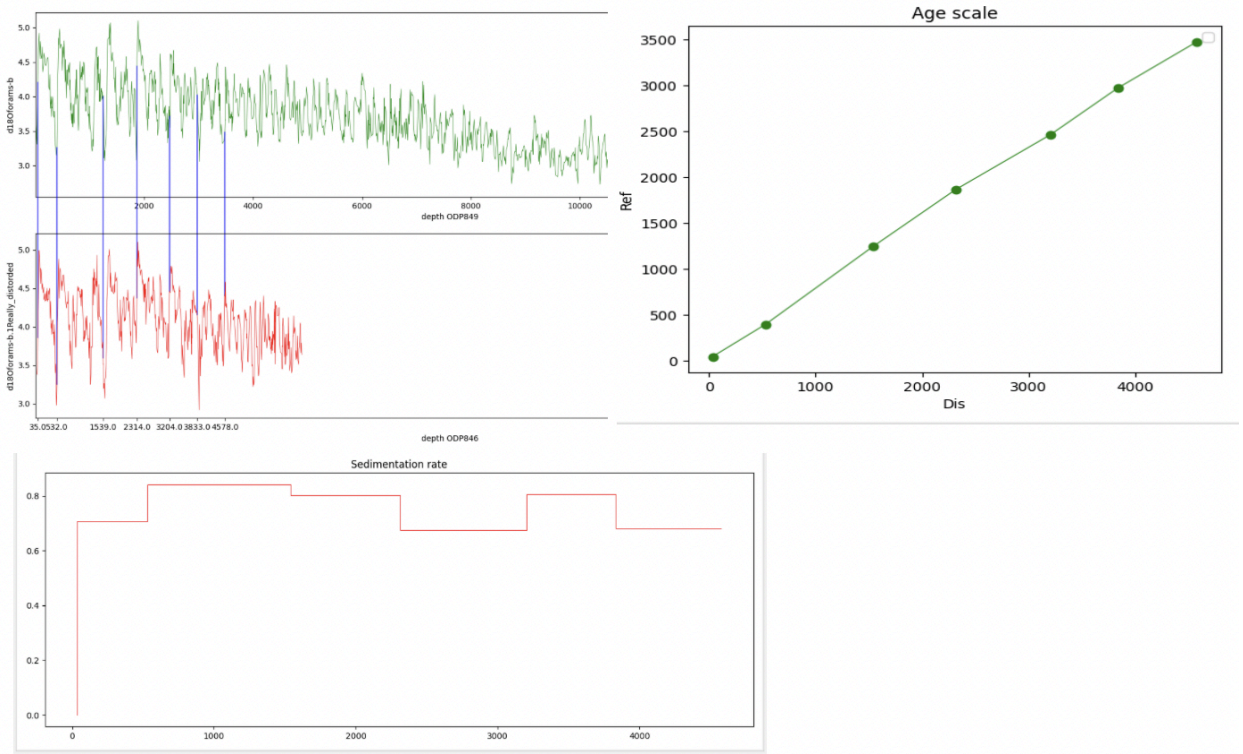

Figure 25: With the same ice cores, pointers are illustrated as well as the recalibration, the age scale function and the sedimentation rate function.

In the next section, the implementation of Linage is explained.

4.8.2 Implementation of Linage

During the fall, the team wanted to explore the area of potentially adding a new feature to this, called Multiseries. Therefore, it was important to understand the implementation of the Linage function and modify it, so the new feature could be implemented. The file Linage.py contains two class objects named Click and PlotGraph. The Click object stores the abscissa and ordinate of a selected point as well as the plot on which it was selected. The PlotGraph is a QMainWindow object that serves as an interface for the user to choose the data that wants to be recalibrated and observe the recalibration. The PlotGraph object has a method called linage, and this is the function that computes the f and draws the recalibration, as illustrated in Figure 26.

```
def linage(self):

    # Clicks are added into linage_points_prep. If they occured in the upper
    # plot, they are saved in linage_points_prep[0] and if the occured in
    # the lower plot, they are saved in linage_points_prep[1]
    # the list is initialized as a two-size zero list
    self.linage_points_prep[self._plotno] = Click(self._x, self._y, self._plotno)

    # If linage_points_prep has a click saved in both spots
    # (linage_points_prep[0] and linage_points_prep[1]), they are carried
    # over into linage_point_pairs where all the clicks are saved. Not
    # just the new ones.
    # if Linage does work till the end, then both values of linage_points_prep must be initialized again to zero. see end of the function
    if self.linage_points_prep[0] != 'No click yet' and self.linage_points_prep[1] != 'No click yet':
        if self.linage_point_pairs[0] == 'No pair added yet':
            self.linage_point_pairs = self.linage_point_pairs[1:]
        self.linage_point_pairs.append(self.linage_points_prep[:])  # adds the new pair of clicks in the format [click1, click2]

        if len(self.linage_point_pairs) > 1:

            # the x coordinates are extracted
            self.clicks_ref = [i[0]._x for i in self.linage_point_pairs]
            self.clicks_dis = [i[1]._x for i in self.linage_point_pairs]

            sorted_clicks_dis , indices = sort_with_indices(self.clicks_dis)
            sorted_clicks_ref = [self.clicks_ref[i] for i in indices]

            # f is a function that maps from the reference graph to the
            # the distorted graph by using the clicks.
            self.f = interpolate.interp1d(
                sorted_clicks_dis,sorted_clicks_ref, fill_value= 'extrapolate')  #  f is just the age-scale
            # g is the reciprocal function provided the clicks order is kept the same
            self.g = interpolate.interp1d(
                sorted_clicks_ref,sorted_clicks_dis, fill_value= 'extrapolate')

            self.new_x_dis = self.f(self.x_dis)
```

Figure 26: Code snippet within the Linage.py file.

So far, the code computes the distorted data $f(x_b),y_b$, but to visualize the recalibration of both data remains. To achieve that, since it is a computation of the distortion of the Ice Core b using ice core a as a reference and to make sure both curves align, the same limit is set to the curves as seen in Figure 25. When the limit was implemented into the code, the software iterates through the rest of the list of clicks and draws blue lines to indicate where the selected points align, as shown in Figure 27.

```
        self.ax1 = self.figure.add_subplot(211)
        self.ax1.plot(self.x_ref, self.y_ref, color='g', linewidth=0.5)
        self.ax1.set(xlabel=self.abs_ref, ylabel=self.ord_ref)

        self.ax1.set_xlim([self.x_ref[0], self.x_ref[-1]])

        self.ax2 = self.figure.add_subplot(212)
        self.ax2.plot(self.new_x_dis, self.y_dis,
                      color='r', linewidth=0.5)
        self.ax2.set(xlabel=self.abs_dis,
                     ylabel=self.ord_dis + 'Really_distorded')

        self.ax2.legend()
        self.ax2.set_xlim([self.x_ref[0], self.x_ref[-1]])

        self.ax2.set_xticks(self.clicks_ref)
        self.ax2.set_xticklabels(np.round(self.clicks_dis))



        # The following loop iterates through all the saved clicks
        # and draws the lines accordingly.
        new_ref_function = interpolate.interp1d(self.x_ref, self.y_ref
            , fill_value='extrapolate')
        new_dis_function = interpolate.interp1d(self.x_dis, self.y_dis
            , fill_value='extrapolate')


        qrows = len(self.linage_point_pairs)
        if not self.linage_point_pairs == ['No pair added yet']:
            for i in range(0,qrows):
                con = ConnectionPatch(xyA=(self.clicks_ref[i],float(new_ref_function(self.clicks_ref[i])) ), xyB=(
                    self.clicks_ref[i], float(new_dis_function(self.g(self.clicks_ref[i])))), coordsA="data", coordsB="data", axesA=self.ax1, axesB=self.ax2, color='b
                print('xyA = ',(self.clicks_ref[i],float(new_ref_function (self.clicks_ref[i]))),'xyB =',(self.clicks_ref[i], self.linage_point_pairs[i][1]._y))
                self.ax2.add_artist(con)

        self.graph.draw()
```

Figure 27: Code snippet within the Linage.py file. Iteration on the saved clicks to draw the blue lines.

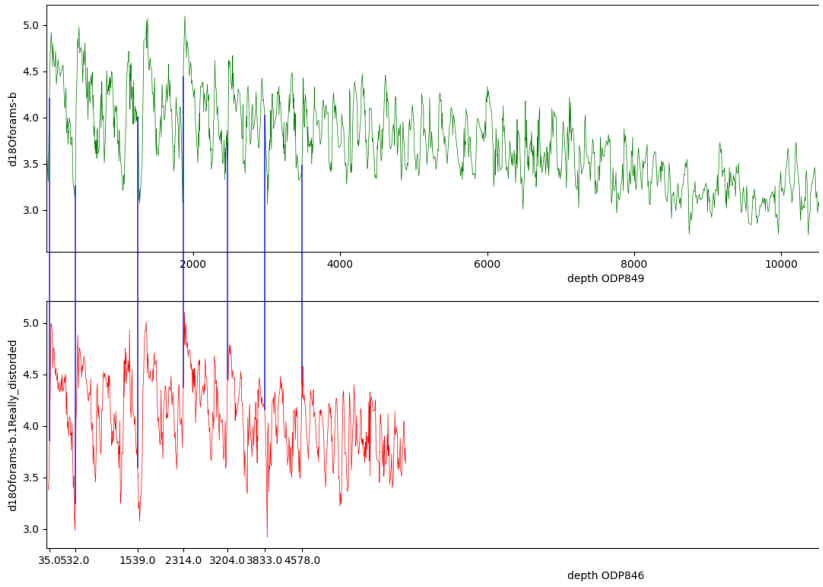

Figure 28: Plot of the recalibration and the clicks.

The goal after the implementation of the single Linage function was to provide an update so that the f function was using more than one data point in each ice core. This means that the function doesn't just use (x_a,y_a) and (x_b,y_b), but also introduces a new set of data points z_a and z_b or even more. The new feature is called Multiseries.

To enable the addition of a Multiseries function, the Linage window had to be updated. Therefore, an implementation of a separate window called Correlation Window to make the pointers have been added. Furthermore, a “start” button was implemented, so that in contrast to the previous version, the correlation plot runs once the user proceeds with “Ok” and has chosen all four data points in the desired order. The limitation before was that since the plotting ran right away, it made it impossible for the user to change the data points used to choose the pointers. Figure 29 and 30 displays how the new separate correlation window compares to the old version.

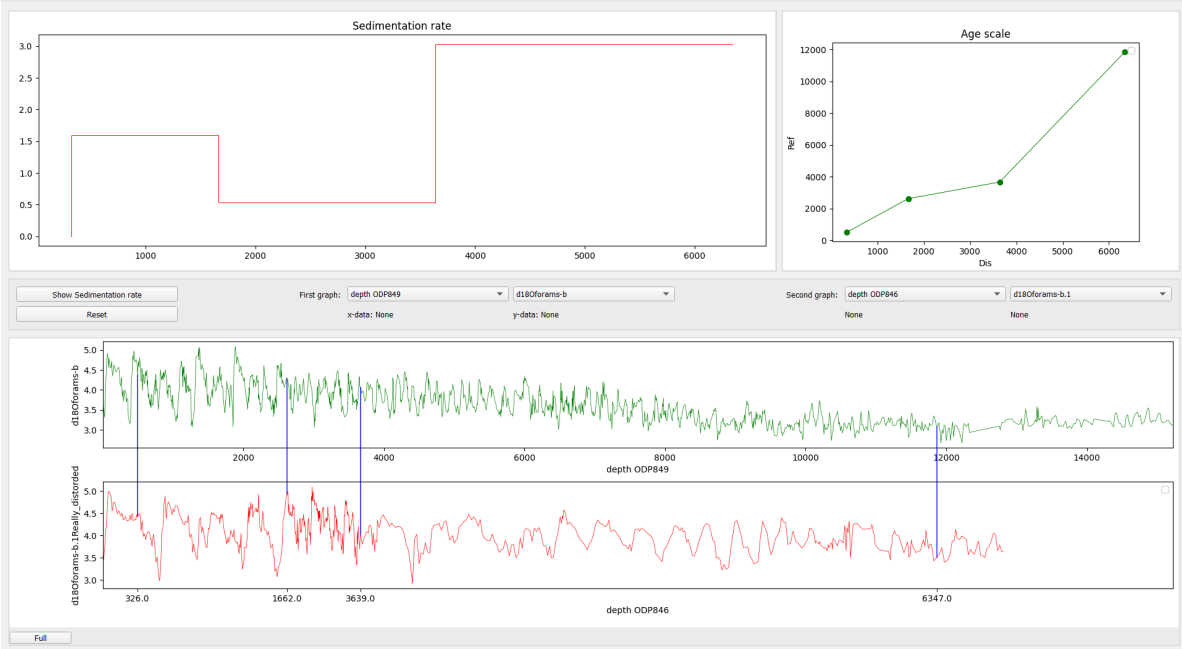

Figure 29: Display of when the correlation window was incorporated in the old version.

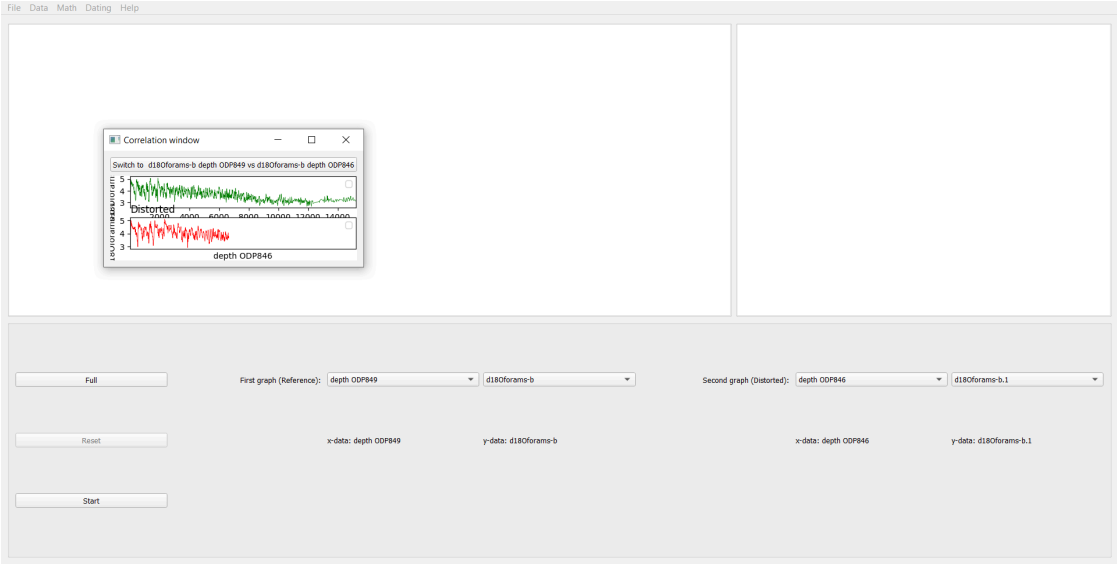

Figure 30: Display of the correlation window in the updated version.

A larger display of the correlation window can be seen in Figure 31.

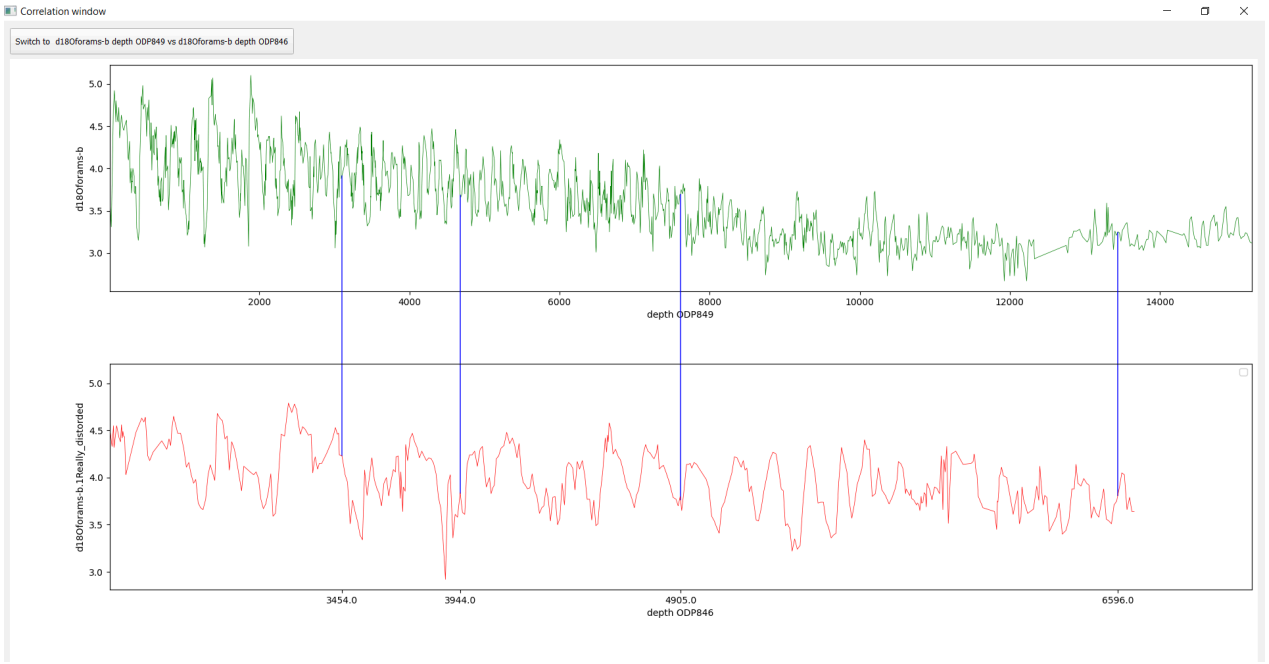

Figure 31: A Zoom on the new independent correlation window.

As displayed in the left upper corner of Figure 31 a new feature has been implemented in the form of a "switch" button. The button allows the user to switch between different series when wanting to use the Linage function as a Multiseries feature. The modifications that have been implemented result in what can be seen in Figure 32, where the age scale function and sedimentation rate are updated once the pointers are created, and the name of the windows displays which data are used for the pointers, just like in the previous version.

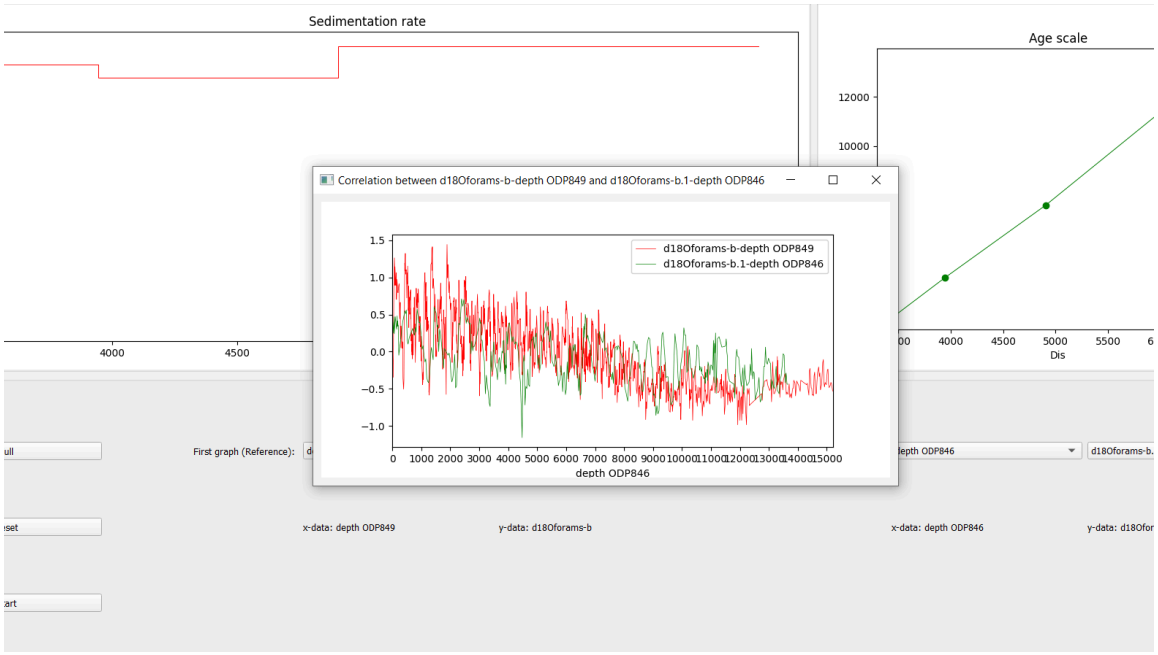

Figure 32: How it looks with the new implementations in the Linage function.

To make sure that the Linage function can be applied for multiple series at the same time, the precedent pointers are applied to the new data, whenever the user switches, therefore the abscissas are kept aligned. The implementation in Figure 32 was done through an update of the "plot_distorted" function, which plots the data that will be recalibrated, by applying the *age scale* function f before plotting and by drawing the pointers (see Figure 33).

```
def plot_reference(self):
    '''creates the reference graph and add it to the figurecanvas'''

    self.ax1 = self.figure.add_subplot(211)

    # we only plot when the userboth x and y are chosen by
    if (self.x_ref is not None) and (self.y_ref is not None):
        self.ax1.plot(self.x_ref, self.y_ref, color= 'g', linewidth=0.5)
        self.ax1.set_xlim([self.x_ref[0], self.x_ref[-1]])
        # labels only added if they exist
        if (self.abs_ref is not None) and (self.ord_ref is not None):
            self.ax1.set(xlabel=self.abs_ref, ylabel=self.ord_ref)
            self.ax1.legend()
        self.ax1.set_title('Reference', loc = 'right')

    self.plot_label_no = 10  # determines the number of labels of the X-Axes of the

def plot_distorded(self):
    '''Creates the graph to distord (lower graph)'''

    self.ax2 = self.figure.add_subplot(212)
    if (self.x_dis is not None) and (self.y_dis is not None):
        self.ax2.plot(self.f(self.x_dis), self.y_dis, color="r", linewidth=0.5)
        self.ax2.set_xlim([self.x_ref[0], self.x_ref[-1]])

        self.ax2.set_xticks(self.clicks_ref)
        self.ax2.set_xticklabels(np.round(self.clicks_dis))

        if (self.abs_dis is not None) and (self.ord_dis is not None):
            self.ax2.set(xlabel=self.abs_dis, ylabel=self.ord_dis)
            self.ax2.legend()
        self.ax2.set_title('Distorted', loc= 'left')



    new_ref_function = interpolate.interp1d(self.x_ref, self.y_ref
        , fill_value='extrapolate')
    new_dis_function = interpolate.interp1d(self.x_dis, self.y_dis
        , fill_value='extrapolate')


    qrows = len(self.linage_point_pairs)
    if not self.linage_point_pairs == ['No pair added yet']:
        for i in range(0,qrows): # à vectoriser
            con = ConnectionPatch(xyA=(self.clicks_ref[i],float(new_ref_function(self.clicks_ref[i])) ), xyB=(
                self.clicks_ref[i], float(new_dis_function(self.g(self.clicks_ref[i])))), coordsA="data", coordsB="data", axesA=self.ax1, axesB=self.ax2, color='b')
            print('xyA = ',(self.clicks_ref[i],float(new_ref_function (self.clicks_ref[i]))),'xyB =',(self.clicks_ref[i], self.linage_point_pairs[i][1]._y))
            self.ax2.add_artist(con)
```

Figure 33: Code snippet from the implementation of saving the precedent pointers when switching to new data.

4.9 Multiseries

Once the modifications necessary for the Multiseries had been implemented, the Multiseries feature was incorporated into the code. It was implemented while maintaining the same layout as for the single Linage function. This resulted in the ability to introduce several more sets of data points when using the Linage function. Let's use the same example as illustrated in Figure 24.

The pointers were created using d18O on OPDP46 and ODP49, now the Multiseries functionality can be used to update the age scale function based on additional pointers generated with another measurement like d13 as well, as illustrated in Figure 34 below.

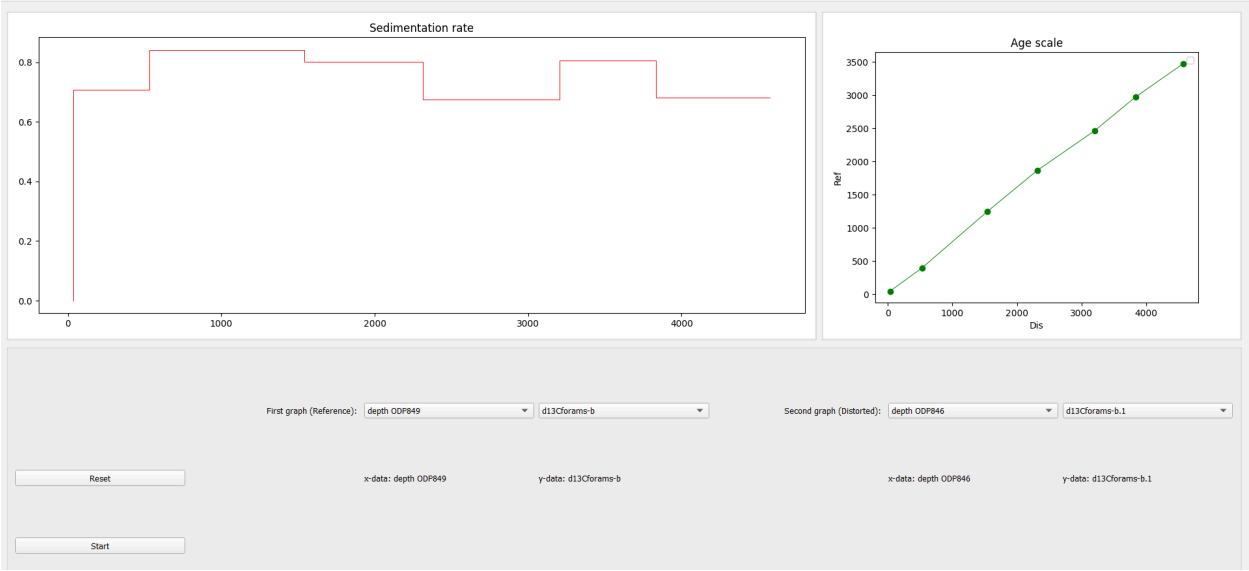

Figure 34: The data d13 is selected on the main window. We can see the selection of new data as well as the current state of the age scale function and the sedimentation rate.

When the user proceeds with the start button, the correlation plot updates containing the previously added pointers as well as seen in Figure 35.

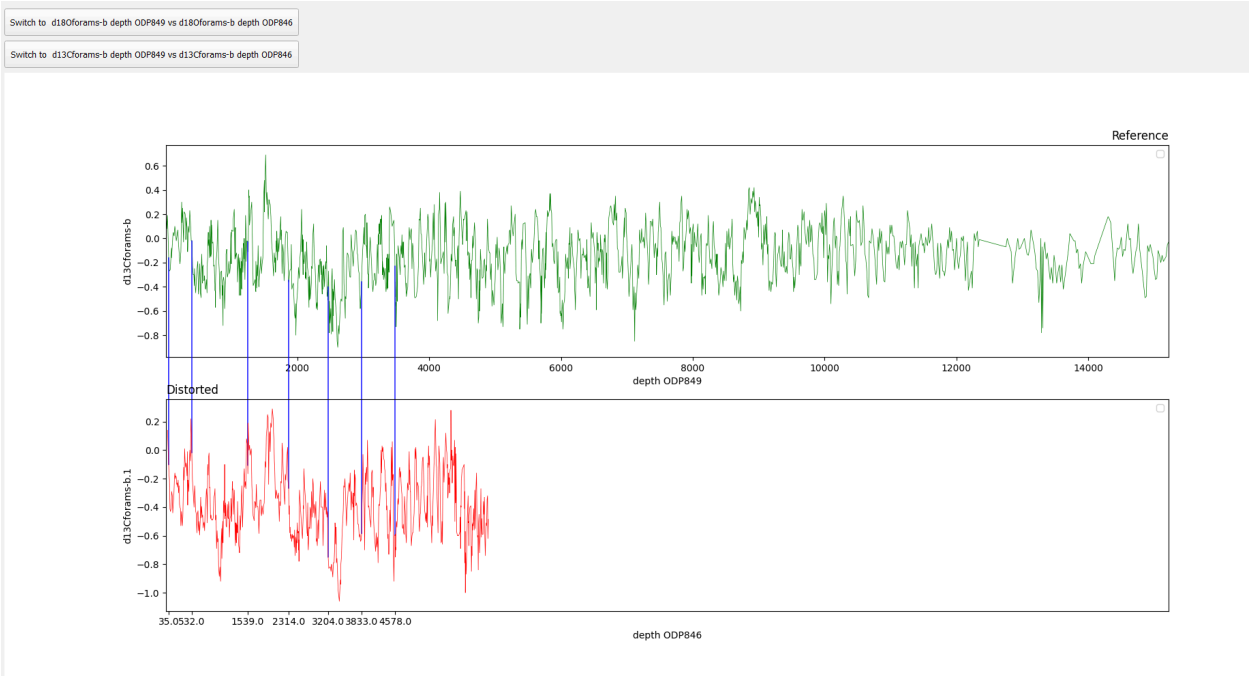

Figure 35: The updated correlation window after adding another data point and proceeding with the start.

The data that is plotted in this new update is d13, but the difference is that it has already been distorted by the previous pointers. Once the new pointers are added with d13 the age scale relation updates. In the

implemented version the user can also switch between d13 and d18 in the correlation window, as discussed earlier.

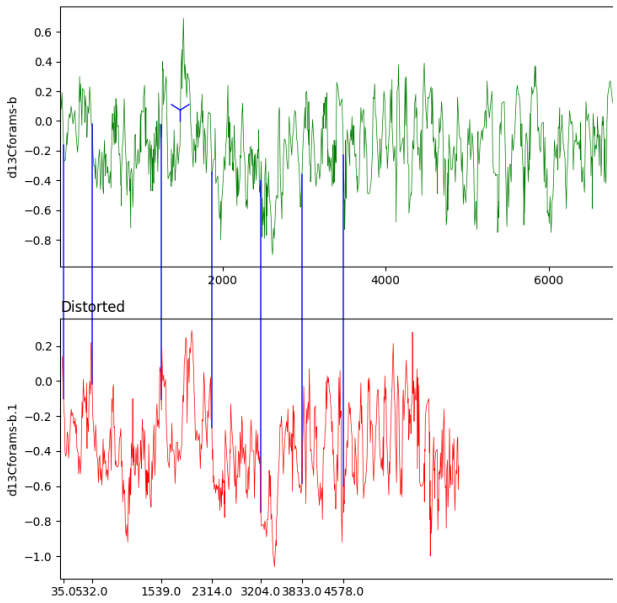

Figure 36: A new pointer is added in d13 at roughly 1500 m.

The age scale function was updated adding a new data point at the ordinate 1500 m.

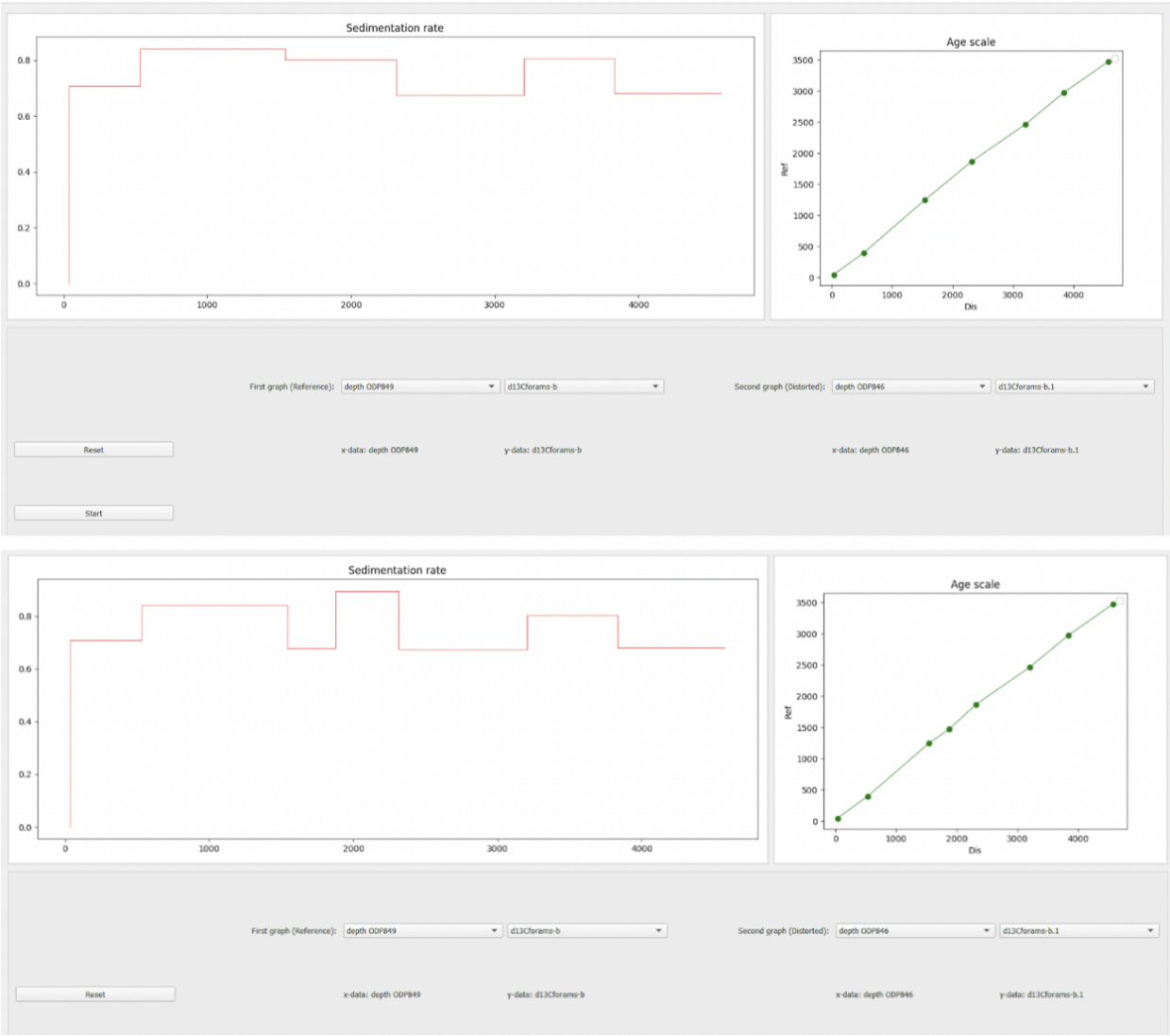

Figure 37: The upper image shows the version of the age scale before introducing d13. In the lower graph, a new point was added (2000 m, 1500 m).

5. Discussion

5.1 Results

To align the software with the specific needs of the LSCE and to further modernize the software to enable it to become an open-source, several improvements can be concluded from the results presented in the sections above. The focus of the project was to enhance user-friendliness, efficiency, and the breadth of the software's features and flexibility.

Firstly, in the user interface, the "worksheet" feature that was introduced offers more user-friendliness than the spring version since it offers user data management and saving capabilities, it also offers more flexibility. Furthermore, the new application menu offers a more intuitive selection of functions for the user, making it more user-friendly. The implementation of multi-window plotting offers more flexibility for the user than the previous version, where a newly opened plot would overwrite the old one. Moving on to data representation and file processing, two prominent enhancements were implemented. The elimination of the inadequate data representation was an important enhancement since it takes away the confusion that can occur when analyzing, offering a more user-friendly base and making the application more efficient to use since the probability of errors is less so researchers can move forward. The updated version's capability to process ".xlsx" files also offers more flexibility and efficiency. In a research landscape, different sources will be applied, and offering a solution that can process several types without having to manually convert the files helps the researchers to move faster in their procedure. From the perspective of turning the application into open-source, this is also a significant factor, since it offers a more seamless tool for anyone to use, no matter what format most researchers use in different geographical areas and so on.

Another significant enhancement that was done during the project was a large amount of code optimization. Through refactoring the code, the design of the software system is ensured to stay simple to fit the purpose without unnecessary complexity or duplication. This helps prevent errors and facilitates future changes to the system (Balaban G, Grytten I, Rand KD, Scheffer L, Sandve GK, 2021). It will also be faster to change, execute, and download the software, and when we have bugs in the code they will be easier to find and understand (Veerraju *et al.*, 2010). This lays out a better foundation to make the application open-source and for future developments. Furthermore, discussing the enhancements applied to the functionalities within the software the correlation function now provides more user-friendliness, breadth, and flexibility since the updated version can handle multiple files simultaneously, as well as display up to four plots at the same time. Moreover, a significant improvement was the introduction of an automatic adjustment feature that has been applied both to the correlation and the histogram function. It offers much more flexibility, and user-friendliness, which is further enhanced by the warning window that appears when the feature is performed so that the user is aware. It is also more efficient for the user, that the software automatically standardizes the length since they don't have to perform this manually.

The Linage function had already been given a lot of time and effort by the previous team in the spring, but some modifications, mostly to add the Multiseries feature, still contributed to improvements. Updating the Linage window, and offering a separate correlation window with a switch action, offers more breadth and flexibility for the user. Furthermore, the Linage function has become more user-friendly since the correlation plot only runs when the user decides to proceed with a "start" button. Lastly, the enhancement opening up for new opportunities and breadth of the software is the added Multiseries feature. Offering the same user-friendliness, and flexibility as the single Linage function, it is a great addition for future development. The feature offers the possibility to use more than one data point in each ice core when plotting their correlation. This may be an important contribution to the future, where emerging techniques are based on multi-ice-core multi-parameter correlations (Sneed, Mayewski, and Dixon, 2011). Lastly, the functionalities that have remained the same features as in the initial version, although deployed into Python, offer more breadth and flexibility for the user than the latest version presented in spring.

5.2 Challenges

5.2.1 Software Development

The software development of Analyseries presented multifaceted challenges besides just the task itself, to recreate complex mathematical functions in a new programming language while keeping it user-friendly. A primary concern was to ensure the application's portability across different operating systems to enable it to be open-source and offer more breadth in future development. Therefore, standardized interfaces have been implemented and platform-specific features have been avoided to ensure the portability of the software, additionally, the usage of Python as a programming language ensures portability (Foo, 2023). Furthermore, another considerable challenge was the initial lack of comprehensive metadata documentation for the ice core data. Metadata is essential for understanding the context and intricacies of datasets, and at the beginning of the project, the absence of it posed a large barrier to initially understanding the software. Therefore, a metadata file was created to help for further development of the application in the future (the file can be found in the annex). Developing the new Multiseries feature also brought its own set of challenges, managing the complexity of multiple data visualizations. The main challenge to overcome was to make sure that the user interface would not become cluttered due to the multiple open data windows. By keeping the new feature close to the single Linage function, this challenge could be overcome.

5.2.2 Project management

Disregarding all the technical and development challenges, the project is a management challenge itself. During the project, the team encountered technical challenges, especially interpreting paleoclimatological data that had to be considered when managing the project. The gap in expertise between the team and the researchers using the software necessitated a deep dive into research before starting the coding process, highlighting the importance of a well-structured and research-oriented approach to handling complex subjects. Another significant challenge was the balance between the team's enthusiasm for learning and exploring within the data science field, without overshadowing the immediate needs and expectations of the client. Furthermore, an important aspect of the project was the challenge of ensuring user acceptance and adoption of the update of the software, imposed from the importance of the reliability of the application. To address these concerns, the team has tried to prioritize transparency with the client, explaining the algorithms and keeping an ongoing dialogue throughout the process. Lastly, the team consists of diverse skills and motivations, which pose both challenges and strengths when ensuring all members are equally engaged and contributing. The team has tried to harness each member's unique abilities and fill any skill gaps through collaborative learning and teamwork.

6. Conclusion

6.1 Project impact on the Laboratory

The project creates value for the researchers at the LSCE in several ways based on the improvements that have been made and that the initial version now has been implemented into Python, modernizing the software substantially. Firstly, several features enhancing user-friendliness have been implemented such

as the introduction of a more intuitive application menu, a “worksheet” feature for data management, and multi-window plotting. User-friendliness can lead to increased productivity and efficiency, as well as reduce the learning curve for new users. Furthermore, the updated software offers more flexibility and efficiency than the previous versions. By enabling the processing of “.xlsx” files and eliminating inadequate data representation, the software allows researchers to work with a variety of data formats without the need for manual conversion of files. This can potentially both save time and reduce errors. The refactoring of the code enhances the software’s stability and reliability. It offers a simpler, more efficient codebase, which can reduce the likelihood of errors and make it more efficient when executing. Moreover, the enhancements to the correlation function, such as handling multiple files and displaying multiple plots simultaneously also offer more flexibility and breadth in the analysis, since it enables easier comparison between different plots. Lastly, the automatic adjustment feature for the Histogram and Correlation function makes the software more user-friendly and the researchers can save time, not having to standardize the lengths manually.

6.2 The Future Development of the Software

In terms of the contributions from the updated version to facilitating the future development of the software, as well as making it open-source, the project during the fall has had a large impact. Since all functionalities that the initial version contained now have been implemented in Python, it is a significant improvement since this was the overall objective when modernizing the software. However, there is a minor issue that needs to be addressed. In the current version, when new data is uploaded and ice cores are detected, the worksheet window will appear (as intended), but it is currently not as useful as it is meant to be. On the original version on Mac, the worksheet could be used to select the data that the user wanted to work on before selecting Linage (or another functionality). This was the intention when implementing the worksheet feature for the updated version. In the current state of the software, the user cannot do that but has to select the file directly on the corresponding menu that the user works on. However, it works as intended regarding saving capabilities. Beyond this minor fix, the goal should be to enhance the user interface further, bringing it beyond its current basic functionality and design.

This involves making it more intuitive and user-friendly, while also incorporating a design that offers a more visually engaging experience. Another opportunity is to integrate advanced features such as B-Tukey or Maximum Entropy. These features could improve the software's capacity for even more in-depth climatological analysis, offering more breadth when adding a range of new analytical tools to researchers. Furthermore, the Multiseries function should be aimed to expand, since it seems to be an emerging area for further development in the future of climatological research.

However, the user-friendliness offered in the current version has made the software more accessible to a broader audience, which is a requirement if it is going to be adopted as open-source. The more intuitive and user-friendly features, as well as code optimization, have improved the software’s readability and made it easier to understand for future development. The refactoring of the code also makes the software more portable across systems, which further makes it more suitable for open-source. Moreover, some enhancements presented for the functionalities may not be useful currently for the researchers at LSCE, but could potentially be useful in the future, and useful for other researchers globally that will use the application. Offering techniques, such as the Multiseries, goes hand in hand with the emerging technique

of the multi-ice-core multi-parameter correlations, and puts the software at the forefront for future purposes. In conclusion, the overall improvements in user-friendliness, data handling, and analytical capabilities, combined with a streamlined and efficient code, prepare the software well for adoption to open-source and future development.

6.3 For Us

The project has significantly contributed valuable skills and insights about data science, project management, and paleoclimatological research for all team members. The development of complex features like the Multiseries function and the transition of software functionalities into Python have contributed to enhanced technical skills. Furthermore, addressing challenges such as the recreation of mathematical functions in a new programming language, and creating a user-centric design has further developed the team's ability to solve problems. Another important aspect is the focus on user-friendliness which has deepened the team member's understanding of a user-centric approach, which is a very useful skill when developing software. The need to address several varying challenges, from technical issues to understanding the subject of research, has taught the team to be adaptable and flexible and adjust the approach when needed to meet the project requirements. Lastly, it has taught the members a lot about project management, collaboration, and teamwork. Working in a diverse team with varying skills and motivations while managing a balance between meeting the client's needs and exploring the field of data science, has emphasized the importance of collaboration and teamwork. These values contribute to the team members' personal and professional growth and will enhance the ability to deliver high-quality work in future projects.

7. References

Al-Saqqa, S., Sawalha, S. and Abdel-Nabi, H. (2020) 'Agile Software Development: Methodologies and Trends,' *International Journal of Interactive Mobile Technologies*, 14(11), p. 246. https://doi.org/10.3991/ijim.v14i11.13269.

Balaban G, Grytten I, Rand KD, Scheffer L, Sandve GK. (2021) 'Ten simple rules for quick and dirty scientific programming,' *PLoS Comput Biol*, 11;17(3). doi: 10.1371/journal.pcbi.1008549.

Foo, D. (2023) 'Design your software for portability - Daniel Foo - Medium,' *Medium*, 20 November. https://danielfoo.medium.com/design-your-software-for-portability-00abcfbf5970.

iotafinance.com (no date) *Formule de calcul: Interpolation linéaire*. https://www.iotafinance.com/Formule-Interpolation-Lineaire.html.

Noor, N.M. *et al.* (2014) 'Comparison of linear interpolation method and mean method to replace the missing values in environmental data set,' *Materials Science Forum*, 803, pp. 278–281. https://doi.org/10.4028/www.scientific.net/msf.803.278.

R. Wise, L. Sheppard, J. Huggins and D. Broadwell, "A methodology and heuristics for re-architecting a legacy system," *2015 Annual IEEE Systems Conference (SysCon) Proceedings*, Vancouver, BC, Canada, 2015, pp. 382-389, doi: 10.1109/SYSCON.2015.7116781.

Sauer, T., (2012) 'Numerical Analysis,' *George Mason University.* https://eclass.aueb.gr/modules/document/file.php/MISC249/Sauer%20-%20Numerical%20Analysis%202e.pdf

Schober, P., Boer, C. and Schwarte, L.A. (2018) 'Correlation Coefficients: appropriate use and interpretation,' *Anesthesia & Analgesia*, 126(5), pp. 1763–1768. https://doi.org/10.1213/ane.0000000000002864.

Sneed, S.B., Mayewski, P.A. and Dixon, D.A. (2011) 'An emerging technique: multi-ice-core multi-parameter correlations with Antarctic sea-ice extent,' *Annals of Glaciology*, 52(57), pp. 347–354. https://doi.org/10.3189/172756411795931822.

Thompson, L.G. (2000). 'Ice core evidence for climate change in the Tropics: implications for our future,' *Quaternary Science Reviews*, 19(1–5), pp. 19–35. https://doi.org/10.1016/s0277-3791(99)00052-9.

Veerraju, R.P.S.P. *et al.* (2010) 'Refactoring and its benefits,' *AIP Conference Proceedings* [Preprint]. https://doi.org/10.1063/1.3516393.