UNIVERSITÉ DE LAUSANNE

Summary

Faculty of Geosciences and Environment (FGSE)
Institute of Earth Surface Dynamics (IDYST)

Doctor of Science

Software and Numerical Tools for Paleoclimate Analysis

by Philipp S. SOMMER

Data-model comparisons of Holocene (11,700 years ago to present) climate provide an ideal basis for evaluating climate model performance outside the range of modern climate variability. The Holocene is recent enough so that boundary conditions of the underlying physics and forcing are well known, while paleoenvironmental archives are abundant and dated with enough precision to comprehensively reconstruct climate. To date, efforts to reconstruct the spatial patterns of Holocene climate change have been mainly focused on the mid-Holocene (about 6'000 years ago), but significant discrepancies have already been identified in data-model comparisons.

These data-model discrepancies can be investigated using instrumental datasets covering continental or hemispheric scales which allow us to reconstruct large-scale climatic features, such as atmospheric dynamics or latitudinal temperature gradients. The generation of these datasets for times prior to the 19th century however faces considerable challenge because there are very few direct measurements of climatic variables. We rely on climate proxies as indirect measurements of the paleo climate. The most abundant one is fossil pollen data, i.e. pollen that are produced by vegetation and can be preserved over thousands of years in terrestrial (or coastal) archives (e.g. lake sediments). This proxy is available from all non-glaciated continents over the world in many different climate regimes, and the primary data is becoming increasingly accessible through large publicly available and community-driven relational databases. Our ability to use this proxy for continental-scale climate reconstructions, however, depends on our ability to analyze, explore and find patterns in these rich and heterogeneous databases. In particular, this requires a proper understanding of the uncertainties that are related to the indirect measurement of climate.

In the first part of this thesis, I present three new software tools that tackle the challenge to make this large amount of data accessible, and to build and develop a continental-scale pollen database. These tools cover a wide range of possible applications to leverage our work with site-based proxy data to a continental scale. The first tool I present is a web framework that is built around a map-based interactive database viewer, developed primarily for the Eurasian Modern Pollen Database, EMPD. This new tool makes the database accessible to other researchers and to the general public, and it allows a continuous and stable development of the community-driven database. In addition to the EMPD, I present an extension of this viewer that makes a large northern-hemispheric fossil pollen database accessible and allows its visual exploration.

The second tool tackles the challenge to fill the gaps in certain geographic areas in the pollen database. *straditize* is a digitization software for stratigraphic diagrams, and pollen diagrams in particular. It can be used to generate new data for the pollen

Chapter 1

Introduction

1.1 Motivation

Our understanding of the climate system is based on computational models that operate on large spatial scales and simulate a complex system of closely related environmental parameters. The evaluation of these models poses a considerable challenge because we need data on continental scale to reconstruct large-scale climatic features, such as atmospheric dynamics or latitudinal temperature gradients. This instrumental data, however, is limited to not even the past two centuries and overlaps highly with the *comfort zone* of the models, i.e. the period where they have both been developed and tested.

A comparison of models with paleo-environmental proxy records, i.e. records from past climates prior to the systematic measurement of meteorology and climatology, provides therefore the only possibility to evaluate the predictive skill of our models for climates that are very different from today. The Holocene, ranging from 11'700 years ago to present, provides an ideal basis for it because (1) it is recent enough so that boundary conditions and forcings are well known, and (2) paleoenvironmental archives are abundant and dated with enough precision to comprehensively reconstruct climate. Each of these archives represent the regional climate condition in the surrounding environment and, when grouped together into large databases; they allow an informed estimate of the climate state over a large period of time and vast geographic areas.

A direct comparison of climate model output and proxies is however still challenging because even the climate proxy record, as an indirect measurement of climate, relies on an inverse modelling approach with associated uncertainties that are not always easy to quantify.

Key challenges for large-scale data-model comparisons on past climates are therefore (1) to gather enough climate proxy information from a spatially large area and a variety of climates, and (2) to provide reliable estimates of the uncertainties associated to the indirect measurements.

These challenges will be addressed in this thesis via the development of new software tools that cover flexible data analysis (chapters 2 and 4), a tool for data gathering (chapter 3), as well as new predictive methods for large-scale paleoenvironmental modelling (chapters 5 and 6).

All these tools are open-source with a strong emphasis on a proper software development that includes documentation and reproducibility.

In the following section 1.2, I will describe the paleo-climate of the current epoch and why this is of interest for future climate predictions. In the subsequent section 1.3, I introduce the influence of software development in this paleo-climatic research, and some of the common open-source software development contents. I conclude this chapter by providing an overview on the contents of this thesis in section 1.3.2.

1.2 Learning from the past — Why we study paleo-climates

Mankind is facing large infrastructural challenges during this century, such as the loss of biodiversity, an exponentially growing world population and an acceleration of growth and globalization of markets (e.g. Ceballos et al., 2015; United Nations, 2019; World Bank, 2002). They all interact with a global climate change that may lead to a new environment none of us ever experienced (Collins et al., 2013). Any future global planning has to account highly diverse responses that range from regional to continental scales (Christensen et al., 2013). As such the complex climate system will enter a state that is significantly different from everything we had since the beginning of the satellite era, i.e. the beginning of global meteorological data acquisition, and even different from what has been experienced within the last 2’000 years (Neukom et al., 2019a,b).

Our knowledge about this new climate is therefore mainly based on computational Earth System Models (ESMs). They face the challenge of simulating a new climate based on our present knowledge of the interactions between the different compartments Ocean, Land and Atmosphere. The validation of it becomes conceptually difficult because of the aforementioned transition into a warmer world during the next century. We are entering a new state and it is questionable how well our models perform (Hargreaves et al., 2013; Mauri et al., 2014).

To evaluate the predictive skill, we rely on our knowledge of paleo-climates, i.e. climates before the systematic measurement of temperature, precipitation, etc.. They provide the only opportunity for a large-scale evaluation of ESMs under conditions very different than today. Paleo-climatic research has therefore been an integral part for climate sciences since the 80s (COHMAP Members, 1988; Joussaume and Taylor, 1995), particularly in the Paleoclimate Modelling Intercomparison Project (PMIP) (Braconnot et al., 2012, 2007a,b; Jungclaus et al., 2017; Kageyama et al., 2016; Otto-Bliesner et al., 2017).

The Holocene interglacial period (11’700 years ago to present) (Walker et al., 2009) is particularly important because it is sufficiently close in time to provide paleo-climate archives and the forcings and boundary conditions are well known (Wanner et al., 2008). With the end of the Younger Dryas around 11’700 years ago, the Earth experienced a climate warming due to changes in orbital precession and obliquity of the Earth, as well as the disappearing residual ice sheets of the Last Glacial Maximum (LGM) (Berger and Loutre, 1991; Peltier, 2004). This results in a multitude of large-scale effects in the atmospheric circulation, such as an increasing amplitude and frequency of the El Niño–Southern Oscillation (ENSO) (Donders et al., 2008), stronger westerly circulation in winter indicating a more positive AO/-NAO over mid-latitudes and the arctic (Funder et al., 2011; Mauri et al., 2014) and changes in the polar amplification and a weakening of the latitudinal temperature gradient (Davis and Brewer, 2009).

Hence, this epoch is of particular interest because the continental setup is comparable to nowadays while still having a climate that is significantly different from present day.

1.2.1 Pollen as a climate proxy

Before 1850, there is almost no instrumental measurement of temperature. Instead we rely on archives such as lake sediments, glaciers, peat bogs, or speleothems that

(Böttinger and Röber, 2019; Nocke, 2014; Nocke et al., 2008; Phillips, 1956; Rautenhaus et al., 2018), although the visualization methods significantly differ due to the differences in data size and data heterogeneity.

The second important aspect for software and paleoclimate is the distribution of data to make it accessible to other researchers, the community and policy makers, which is commonly established through online accessible data archives and recently also through map-based web interfaces (Bolliet et al., 2016; Williams et al., 2018).

The following sections provide an overview on the different techniques used by palynologists to visualize and distribute their data and concludes with an introduction into Open-Source Software Development, which forms the basis of all the software solutions that are presented later in this thesis.

1.3.1 Sofware for Proxy Data Analysis, Visualization and Distribution

Due to the nature of stratigraphic data, proxies, especially pollen assemblages, are often treated as a collection of multiple time-series (one-dimensional arrays). The size of one dataset is generally small (in the range of kB) and can be treated as plain text files. Traditionally, numerical and statistical analysis are separated from the visualization.

In palynology, standard analytical tools are Microsoft Excel[1] and the R software for statistical computing (R Core Team, 2019). The latter also involves multiple packages for paleoclimatic reconstruction, such as `rioja` (Juggins, 2017) and `analogue` (Simpson, 2007; Simpson and Oksanen, 2019), and bayesian methods exists in a variety of programming languages ~~(**Tipton2017**; e.g. Haslett et al., 2006; Holmström et al., 2015; Nolan et a~~ (e.g. Haslett et al., 2006; Holmström et al., 2015; Nolan et al., 2019; Parnell et al., 2014; Tipton, 2019) . Alternatively, desktop applications exist, such as Polygon[2] by Nakagawa et al., 2002 or the CREST software presented by Chevalier et al., 2014 and Chevalier, 2019.

It is a long-standing tradition to visualize stratigraphic data, and especially pollen data, in form of a stratigraphic (pollen) diagram (Bradley, 1985; Grimm, 1988). Especially during the 20th century, when it was not yet common to distribute data alongside a peer-reviewed publication, pollen diagrams have been the only possibility to publish the entire dataset (see also chapter 3). The generation of these diagrams is usually based on desktop applications such as C2 (Juggins, 2007) or Tilia[3] (Grimm, 1988, 1991). A more recent implementation into the psyplot framework (Sommer, 2017, chapter 4) is also provided with the psy-strat plugin[4] (Sommer, 2019).

Raw pollen data is at present made available through web archives, such as PANGAEA[5] or the National Climatic Data Center (NCDC) by the National Oceanic and Atmospheric Administration (NOAA)[6]. Collections of data, such as regional pollen databases or project specific collections (e.g. Davis et al., 2013; Whitmore et al., 2005) are usually published in one of the above-mentioned archives or associated with a publication. A different approach has been taken by Bolliet et al., 2016 who developed a small web application as an interface into the data collection, the *ClimateProxiesFinder* (Brockmann, 2016, chapter 2).

Outstanding compared to the previous data interfaces is the new infrastructure for the Neotoma database (Williams et al., 2018). It consists of the map-based web

[1]`https://products.office.com/en/excel`
[2]`http://polsystems.rits-paleo.com`
[3]`https://www.tiliait.com/`
[4]`https://psy-strat.readthedocs.io`
[5]`https://pangaea.de/`
[6]`https://www.ncdc.noaa.gov/data-access/paleoclimatology-data`

Chapter 3

Straditize

A digitization software for pollen diagrams

Straditize *is published in the Journal of Open Source Software:*

Sommer, Philipp S., Dilan Rech, Manuel Chevalier, and Basil A. S. Davis (Feb. 2019). “straditize: Digitizing stratigraphic diagrams”. In: *Journal of Open Source Software* 4.34, p. 1216. DOI: `10.21105/joss.01216`. URL: `https://doi.org/10.21105/joss.01216`.

Abstract. The conversion of printed diagrams or figures into numerical data has become extremely important in ensuring that scientific work, especially from the pre or early digital age, is not lost to science. One of the most common figures used in the paleo-sciences is the stratigraphic diagram, where the results of the analysis of samples are plotted against a common y-axis, usually representing age or depth. Currently this type of diagram is laborious and error prone to digitize using current software designed for simple x/y graphs. Here we present a new open source software written in python that is specifically designed to quickly and accurately digitize stratigraphic diagrams based on a user controlled semi-automatic process. The software is optimized for use with pollen diagrams, but will work well with many other types of similar diagram. The software is fully documented and includes integrated help and tutorials.

3.1 Introduction

As with almost all areas of science, the digital capture, storage, manipulation and sharing of data has almost completely transformed the way that paleo-science is now undertaken compared to just 20-30 years ago. This digital transformation has created entirely new types of datasets, analysis, collaborations and visualizations, but it has also created a profound divide between the science that is available in digitized form, and that which is only available in analogue or paper form. This is a particular problem for science that was undertaken and published before this digital revolution, or where the original digital version of a dataset is unavailable, perhaps through retirement or other personnel changes, accidental damage to data storage devices, incompatible or out of date hardware storage or data file formats.

This data however may still be available as a published or printed diagram, which can be turned into numerical data by digitization, either manually or often using graph digitizing software such as *Graphclick*, *Engage Digitizer*, *Plot Digitizer*, *g3data*, *DigitizeIt* and *WebPlotDigitizer* (Rohatgi, 2019). While this approach may be optimal for simple graphs with a single x and y axis, it can rapidly become extremely multiple variables plotted in the same column do not overlap either. The software can process multiple variables in a column so long as these are stacked or additive, and therefore they never overlap. An example in a pollen diagram could be where multiple species of, for instance Betula, are plotted as a stacked diagram in a single column so that the sum of the species also shows the total ~~Betula~~*Betula*.

At the top of each column it is usual to find the name of the variable plotted in the column. This may be shown at a variety of angles, but usually either vertically, or at an angle or rotated to make it easier to read and fit within the diagram (fig. 3.1 c). This label can be automatically read by *straditize* and the name assigned to the respective column data, although care should be taken as the label is sometimes offset from the column it represents.

Variables are also often grouped together and the group labelled, for instance in pollen diagrams into *trees & shrubs*, and *herbs* (fig. 3.1 d). For pollen data these groups usually share the same x-axis units/scaling (as in fig. 3.1 b). In *straditize* the units/scaling can be set and applied to a whole group of columns/variables, or set and assigned for each individual column/variable (see fig. 3.1 e).

3.2.1.2 Diagram types

A number of different diagram types are shown in figure 3.1 that are commonly used in pollen diagrams, as well as other stratigraphic diagrams. These can all be identified and read by *straditize*. One of the most commonly found diagrams in pollen diagrams are line diagrams (fig. 3.1 f), or line diagrams where the area underneath of the line is filled to make an area diagram (fig. 3.1 h). Data is also often commonly presented as bar diagrams that make it clearer where the individual samples are located (fig. 3.1 g). Both bar and line/area diagrams may also be stacked, where (as we have already mentioned) columns may contain multiple variables stacked one upon the other (fig. 3.1 i). These various diagram types require different digitization strategies, which we discuss in the digitization section below (see section 3.2.2).

3.2.1.3 Informative features

Other more specialized features can also be found in pollen diagrams that provide additional information for the reader, but are more difficult to interpret for the software. For instance the taxa or variables in a pollen diagram are usually all plotted on the same scale even if they are in different columns, so that direct visual comparison can be made between them. However, whilst this works well for large percentage values, it can often be difficult to see changes in low percentage values, which may still be ecologically important. To help the reader see these changes in low percentages, pollen diagrams often include a vertical exaggeration. This means that the percentages for a pollen taxa in a column will be plotted with two lines, one showing the percentage value shown on the scale, and the other showing the percentage value multiplied or exaggerated by a certain factor (usually 5 or 10) (fig. 3.1 j). A different approach to the same problem is to mark the low percentages with a symbol or marker. For instance, a common method is to mark all samples with less than 0.5% or 1.0% with a "+" symbol (fig. 3.1 k).

Other features that are often added to pollen diagrams and other stratigraphic diagrams are vertical and horizontal lines. Vertical lines often denote the start of a column, representing the baseline of the y-axes. These are often a continuous or discontinuous dashed line, and when it is quite a thick line it can be difficult to define

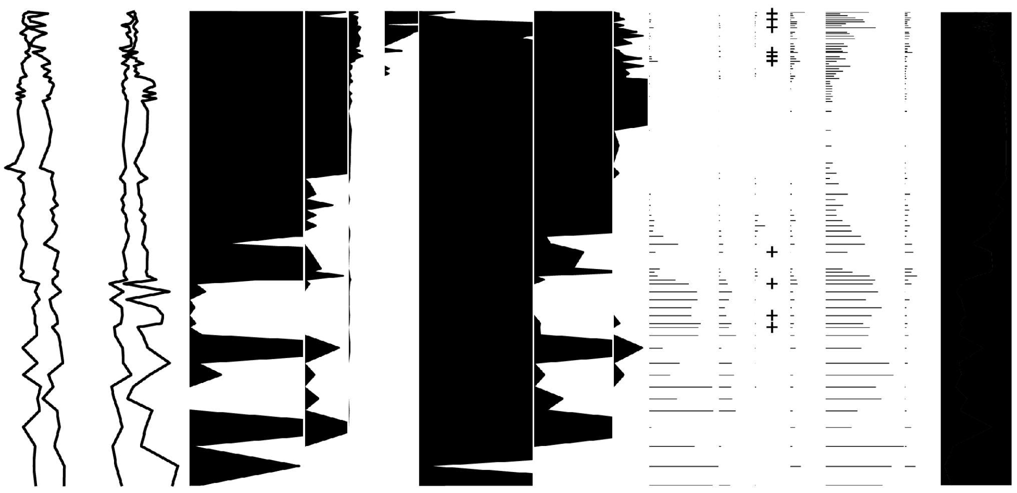


FIGURE 3.2: Cleaned binary image of the data part of figure 3.1. Informative features (Y-axes and horizontal lines) have been removed. Exaggerations and occurrences are still in the binary image and are considered separately in section ??3.2.2.4.

Straditize therefore has an automatic algorithm to detect vertical axes that tries to minimize the removing of real data pixels. This detection is described by the following algorithm:

Let $C(i)$ be the most frequent color in the pixels of a pixel column i, and $D(i)$ the amount of data pixels in this column. A pixel column that is covered by data with more than a user-defined threshold (by default 30 percent of the data part height, see section 3.2.2.1) is considered as part of a y-axis if

- it is either the first pixel column in the subdiagram with data (i.e. $D(j) = 0$ for $j < i$ and j being larger or equal than the column start of the diagram),
- or the dominant color of the pixel column is the same as for the previous pixel column (i.e. $C(i) = C(i-1)$) and the number of data points is approximately the same (i.e. $D(i) \approx D(i-1)$).

This procedure results in one vertical line per column and works independent of whether it is a dotted, dashed or solid line. However, the line width is critical and may vary a lot. If a diagram column contains a filled line (fig. 3.1 h) that merges with the vertical line defining the y-axis, then the algorithm could potentially overestimate the width of the y-axis line. Therefore, we use the median of all of the estimated lines from the various columns as the width of the y-axis line, and reduce the width of each of the lines to this amount. The algorithm then looks for informative features or other features that appear to be part of the data columns but are not part of the data behind the image. These include small features such as axis tick marks for example, and lighter pixels (i.e. close to white) that are usually a result from the rasterization of the diagram image. *straditize* then moves the start of the column because it assumes that the starting point for the x-axis (for pollen taxa it would be the 0% line) is in the middle of the vertical line marking the y-axis.

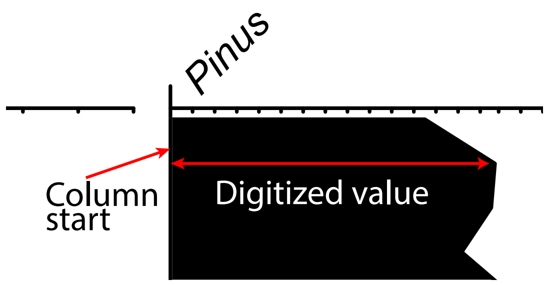


FIGURE 3.3: Illustration of the basic digitization strategy of *straditize*. For each diagram column in the binary image (fig. 3.2), we use the pixel that is located furthest to the right on the curve and take the distance to the column start as the digitized value. This is then transformed from the pixel scale into the data units based on the user input.

(4) Handling low taxon values

One particular feature often associated with pollen diagrams is the use of vertical exaggeration to help visualize changes in low percentages. Ordinarily, low percentages could be viewed better by changing the x-axis scale for the taxa with low percentages, but with pollen data it is also important to be able to make a visual comparison across all of the taxa listed, including those with high pollen percentages. Therefore, a common scale for all taxa is important. The exaggeration (fig. 3.1 j) usually takes the form of a second line outside of the first, usually representing x5 or x10 exaggeration of the scale presented on the x-axis. This line could be expected to have greater precision than the first for any given pixel resolution, but problems emerge when the value of the exaggerated value exceeds the width of the scale on the x-axis, so that the line marking the exaggerated values is truncated at high values above a certain threshold. Another common way to help the reader identify low percentages in pollen diagrams is to use a marker (often a "+" symbol, fig. 3.1 k) for values below a certain threshold. This can be particularly useful for very low counts where the author wants the reader to be aware that pollen of a certain taxa was found, even if the pollen counted was very low. This is often used for instance with "important" taxa such as ~~Cereals~~ *Cereals* that can indicate human agricultural activity, and taxa like ~~Larix~~ *Larix* (Larch) that have notoriously low pollen productivity and where <1.0% in a pollen diagram may actually represent 20% of ~~Larix~~ *Larix* trees in the surrounding landscape. For the purpose of digitization, the *straditize* user can either remove these exaggerations, or use some of the functions available in *straditize* to consider both the non-exaggerated and the exaggerated information in the diagram. In the case of the use of symbols to represent values below a threshold, it needs to be decided what value the symbol will represent once it is digitized and turned into numerical data. In any case, exaggerations and occurences are automatically removed from the image before the diagram is digitized (see next step 3.2.2.5).

(5) Digitizing the diagram

After removing the informative features (see section 3.2.2.3) and exaggerations (see section 3.2.2.4), *straditize* can automatically digitize the various columns on a pixel basis. In general *straditize* treats every column of the diagram separately and uses different algorithms for the various plotting types:

Area and line diagrams such as those shown in the fig. 3.1 f and fig. 3.1 h are digitized based on the pixel located furthest to the right on the curve in any particular column (illustrated in figure 3.3).

Bar diagrams as in figure 3.1 g are also digitized based on the the pixel located furthest to the right on the bar in any particular column. Additionally, *straditize* distinguishes between two adjacent bars by using a user defined threshold (by default two pixels). Additionally, it identifies bars that are significantly wider than the others (which would indicate two or more overlapping bars) and then the user can split them manually.

Stacked diagrams as in figure 3.1 i have to be digitized manually. The user has to manually distinguish the different areas using the selection tools in *straditize*.

These procedures each result in one value per pixel row in each variable column in the data part. The next step is then to extract only the rows that are necessary to regenerate the diagram, that is, the pixel rows that correspond to the samples.

(6) Finding the samples

A key function of *straditize* is its ability to identify sample levels in the data, so that measurements of x-axis values in each column for each variable are assigned to the appropriate y-axis sample depth or age across all columns and variables. The search and assignment of sample levels can be done either automatically or manually, and if done automatically, this can also be later edited following manual checking.

The algorithm is thereby split into two steps:

1. For each column: Identify the intervals that contain exactly one sample (the rough locations, i.e. certain consecutive pixel rows in every column where we know that there is a sample, but we do not know where exactly)

2. Align the overlapping intervals between the columns to estimate the exact location

The implementation of step 1 necessarily differs between bar and area/stacked ~~/~~or line diagrams. With bar diagrams *straditize* uses the bars identified in the previous digitization step (section 3.2.2.5) to define these rough sample locations, while for the other diagram types the algorithm looks for local extrema in the graph line, i.e. intervals that are lower or higher than the surrounding areas. This implies that each sample is associated with a local minimum or maximum in at least one of the diagram columns. This holds well for pollen diagrams that usually sum up to 100% across all of the columns or variables in the diagram, but it is however not generally true for all stratigraphic diagrams.

Step 2 then aligns these rough intervals and uses the overlapping information from the different columns to estimate the exact location of the sample. This is described by the following procedure, where we focus on a simple case of only two diagram columns. Assume that the two columns i and j have a sample in the corresponding overlapping intervals I_i and I_j (i.e. $I_i = \{r_{i,1}, r_{i,2}, \ldots\}$, and $I_j = \{r_{j,1}, r_{j,2}, \ldots\}$ with $I_i \cap I_j \neq \emptyset$). To find the exact location, *straditize* distinguishes the following cases:

If one of the intervals contains only one pixel row (i.e. $|I_i| = 1$ or $|I_j| = 1$), *straditize* sets the sample at exactly this location self-describing Common Data Model on which the network Common Data Form (netCDF) is built (Brown et al., 1993; Hoyer and Hamman, 2017; Rew and Davis, 1990). The package integrates with standard python from the python environment, such as the computing and analysis packages numpy (Oliphant, 2006), scipy (Jones et al., 2001; Oliphant, 2007), pandas (McKinney, 2010) and statsmodels (Seabold and Perktold, 2010), but also offers intuitive interfaces for other packages, such as a package for empirical orthogonal functions (EOFs, Dawson, 2016), CDOs (Müller, 2019), fourier transforms (Uchida et al., 2019) and many more[3]. This large potential for extension distinguishes psyplot from other high-level visualization software, such as ParaView or Vapor, as such python packages can be implemented as a so-called *formatoption* (without hyphen, see below) or used in a pre-processing step.

Psyplot core structure

The core structure of psyplot consists of five base classes that interact with each other, the visualization objects *Plotter* and its *Formatoptions*, the data objects *DataArray*, an *InteractiveList* of them, and a collection of all of them, the psyplot *Project*. It is schematically visualized in figure 4.1.

The most high-level API object is the psyplot project that consists of multiple data objects that are (or are not) visualized. The main purpose is a parallel handling of multiple plots/arrays that may also interact with each other (e.g. through the sharing of ~~formatoptions~~*formatoptions*). It mainly spreads update commands to its contained objects, but also serves as a filter for the data objects. Furthermore, one project may be split up into sub projects which then only control a specific part of the main project, e.g. for a specific formatting of only a small part of the data.

The next level is the *DataArray* from the xarray package (or more explicitly, its accessor, the *InteractiveArray*[3]), that holds the data of one (or more) variables (e.g. temperature) and its corresponding coordinates (e.g. time, latitude, longitude, etc.). It may be one or multidimensional depending on the chosen visualization method. psyplot offers several methods to provide the coordinates for the plotting of different grids to make the visualization easier. The software can interpret CF Conventions[4] and UGRID conventions for unstructured grids (Jagers et al., 2018).

Multiple of these arrays can also be grouped together into an *InteractiveList* that shall be visualized by the same plot method (e.g. multiple lines or a scalar field with overlying vector field).

The visualization part in the framework is managed by the *Plotter* class, a collection of multiple *Formatoptions*. Each plotter subclass is designed to visualize the data in a specific manner (e.g. via line plots, violin plots, or map plots) and is completely defined through it's ~~formatoptions~~*formatoptions*.

Formatoptions are the core of the psyplot structure. The standard functionality of a formatoption is to control the visual appearance of one aspect of the plot (e.g. through the colormap, figure title, etc.). It is, however, completely unlimited and can also do data manipulations or calculations. The psy-reg plugin for example (see section 4.3.2) implements a formatoption that performs a regression through the data that is then visualized. As mentioned earlier, each plotter is set up through its ~~formatoptions~~ *formatoptions* where each formatoption has a unique formatoption key inside the plotter. This formatoption key (e.g. *title* or *cmap*) is what is used for

[3] several packages related to xarray are listed in the docs at `http://xarray.pydata.org/en/stable/related-projects.html` and psyplots integration (accessors) in particular is shown at `https://psyplot.readthedocs.io/en/latest/accessors.html`.

[4]`http://cfconventions.org`

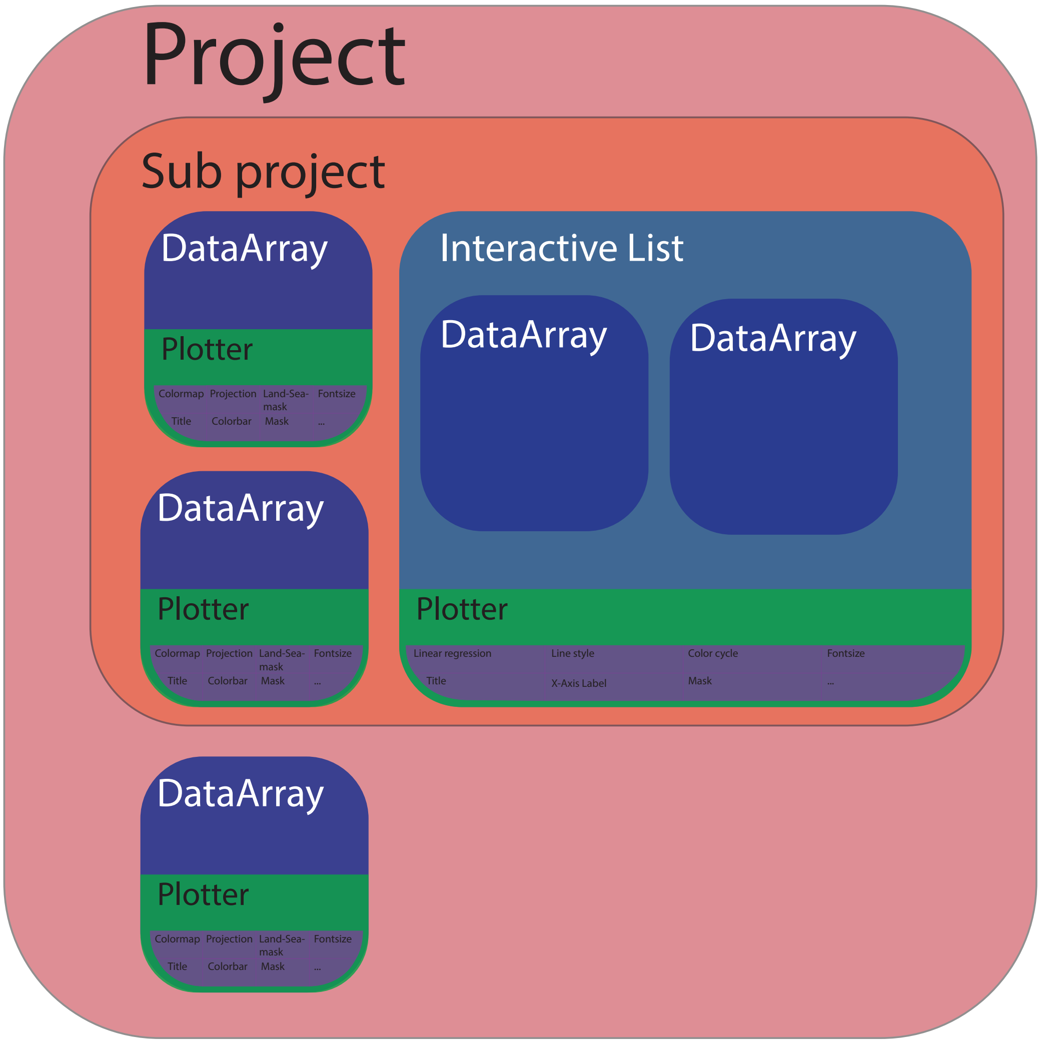


FIGURE 4.1: The psyplot core framework. A (sub) project consists of n-dimensional data arrays or a list of these that are each visualized by a plotter. Each plotter consists of a set of ~~formatoptions~~ *formatoptions* that control the appearance of the plot or performs data manipulation.

updating the plot, manipulating the data, etc.. ~~Formatoptions~~ *Formatoptions* might also interact with other ~~formatoptions~~ *formatoptions* inside the plotter or from other plotters. This concept of ~~formatoptions~~ *formatoptions* allows to use the same formatoption with all different kinds of plotters and the interaction of multiple plots with each other. Common plot features, such as the figure title, colormap, etc., therefore do not have to be implemented explicitly for every plotter but can be used from existing implementations. This framework also allows a very easy integration and development of own ~~formatoptions~~ *formatoptions* with a low or high level of complexity.

4.3.2 Psyplot plugins

The psyplot package provides the core of the data management described in the previous section 4.3.1. The real visualization is implemented in external plugins. The advantage of this approach is an increased flexibility of the entire framework (collaborations can evolve through dedicated plugins) and of managing the various dependencies of the packages. As such, the dependencies of psyplot are rather week (only xarray is needed), but the dependencies of the plugins can be more extensive (e.g. for geo-referencing or advanced statistics).

Each plugin defines new *Plotters* and *Formatoptions* that are specific to the purpose of the visualization/analysis task. The plotters can also be implemented as a plot method (see supplements 4.B to 4.E) and accessed through the psyplot core API (see supplements 4.A for an example).

The current lists of plugins include *psy-simple* for rather simple and standard visualization tasks, *psy-maps* for geo-referenced plots, *psy-reg* for statistical analysis visualization, and *psy-strat* for stratigraphic diagrams.

psy-simple: The psyplot plugin for simple visualizations

Much of the functionality that is used by other plugins is developed in the psy-simple plugin. This package targets simple visualizations and currently includes plot methods for one-dimensional data: line plots, bar plots and violin plots; for two-dimensional data: scalar plots, vector plots and combined scalar and vector plots; and plots that do not require complex data manipulation: a density plot and a plot of the weighted geographic mean. A full list of examples is provided in the supplementary material, section 4.B.

This package also implements most of the functionality to handle unstructured grids in 2D visualizations and defines most of the commonly used ~~formatoptions~~*formatoptions*. The latter include text manipulation (such as plot title, figure title, x- and y-axis labels, etc.), data masking, x- and y-axis tick labeling and positioning, as well as color coding for 2D plots (colormap, colormap sections, etc.).

psy-maps: The psyplot plugin for visualizations on a map

psy-maps builds on top of the psy-simple plugin and extends its functionality for visualizations on a map using the functionalities of the cartopy package (Met Office, 2010 - 2015) (see supplements 4.C for examples). It simplifies as such the automated generation of maps for climate model data through the flexibility of the psyplot framework.

psy-maps currently implements additional ~~formatoptions~~ *formatoptions* for choosing the projection of the map, selecting the geographic region, drawing the contintens or shaded reliefs of land and ocean, and more. One feature that distinguishes psy-maps from other visualization software, even from pure cartopy, is the ability to visualize unstructured geo-referenced grids on the map. For this purpose, triangles are projected in a pre-processing step to the target projection, prior to the visualization with matplotlib. This drastically increases the performance and makes it possible to visualize even very large data sets. As such, psy-maps visualizes a global scalar field on a hexagonal grid of roughly 4.4 million grid cells ($\approx$ 13 km resolution) in roughly 3.5 minutes. The interactive usage of such a large dataset is however limited by the functionalities of matplotlib to handle such an immense amount of data.

psy-reg: The psyplot plugin for visualizing and calculating regression plots

psy-reg performs regression analysis on 1D variables using the methods of the statsmodels (Seabold and Perktold, 2010) and scipy (Jones et al., 2001; Oliphant, 2007) packages, and visualizes the results with the functionalities of the psy-simple plugin. As such, it implements ~~formatoptions~~ *formatoptions* for univariate regressions, confidence intervals via bootstrapping, and combined plots of the data density and the fitted model (see also supplements 4.D). The necessity for this package arose from the need to visualize a regression model, compare it (visually) with the original data and to use it afterwards. Other python packages either focus only on the generation of the regressions (such as statsmodels or scipy), or on their visualization (such as seaborn (Waskom et al., 2018)). The psyplot plugin makes it possible to generate the visualization and to access the underlying regression model parameters and uncertainties.

psy-reg has been heavily used for the parameterization of the weather generator in chapter 6 which also gave the initial motivation for the package.

psy-strat: A psyplot plugin for stratigraphic plots

psy-strat (Sommer, 2019a) is the latest plugin for psyplot that has been developed for stratigraphic diagram visualization. It is particularly designed for the straditize software (Sommer et al., 2019, chapter 3) and was motivated by the need for an automated creation of pollen diagrams. One example of such a diagram is provided in the supplementary material, section 4.E.

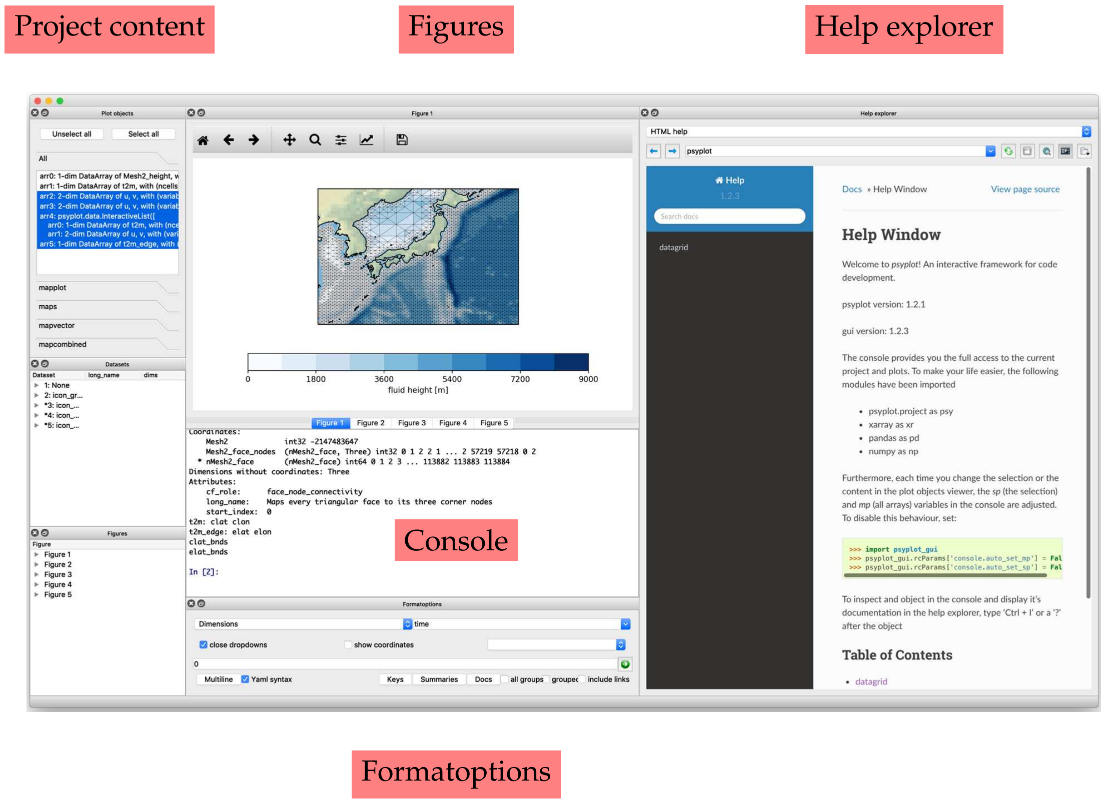
As the psy-reg and psy-maps plugins, psy-strat uses the functionalities of the psy-simple plugin for a visualization of multiple variables in separate diagrams that share one common vertical axis (usually age or depth)[5]. Additionally, besides the integration that is common for every psyplot plugin (see next section 4.3.3), psy-strat contains additional functionalities for the psyplot GUI. This implementation allows the user to select and reorder the variables (pollen taxa) that are shown in the stratigraphic diagram.

4.3.3 The psyplot Graphical User Interface

Psyplots objective of providing a platform for flexible and convenient data analysis is further approached with the *psyplot-gui* package. This extension to the framwork provides a GUI for simplified access to the plotting features in psyplot.

A strong focus of this interface is, again, the flexibility. psyplot-gui is based on the cross-platform PyQt5 library[6], a very flexible and frequently used package for

[5]See psy-strat.readthedocs.io for an example of psy-strat.
[6]PyQt5 can be accessed via `https://riverbankcomputing.com/software/pyqt/intro`.



arr0: 1-dim DataArray of Mesh2_height, w
arr1: 1-dim DataArray of t2m, with (ncells
arr2: 2-dim DataArray of u, v, with (variab
arr3: 2-dim DataArray of u, v, with (variab
arr4: psyplot.data.InteractiveList([
  arr0: 1-dim DataArray of t2m, with (nce
  arr1: 2-dim DataArray of u, v, with (vari
arr5: 1-dim DataArray of t2m_edge, with (

Dataset	long_name	dims
1: None		
2: icon_gr...		
*3: icon_...		
*4: icon_...		
*5: icon_...		

```
Coordinates:
    Mesh2            int32 -2147483647
    Mesh2_face_nodes  (nMesh2_face, Three) int32 0 1 2 2 1 ... 2 57219 57218 0 2
  * nMesh2_face      (nMesh2_face) int64 0 1 2 3 ... 113882 113883 113884
Dimensions without coordinates: Three
Attributes:
    cf_role:      face_node_connectivity
    long_name:    Maps every triangular face to its three corner nodes
    start_index:  0
t2m: clat clon
t2m_edge: elat elon
clat_bnds
elat_bnds

In [2]:
```

Docs » Help Window   View page source

Help Window

Welcome to *psyplot*! An interactive framework for code development.

psyplot version: 1.2.1

gui version: 1.2.3

The console provides you the full access to the current project and plots. To make your life easier, the following modules have been imported

- psyplot.project as psy
- xarray as xr
- pandas as pd
- numpy as np

Furthermore, each time you change the selection or the content in the plot objects viewer, the *sp* (the selection) and *mp* (all arrays) variables in the console are adjusted. To disable this behaviour, set:

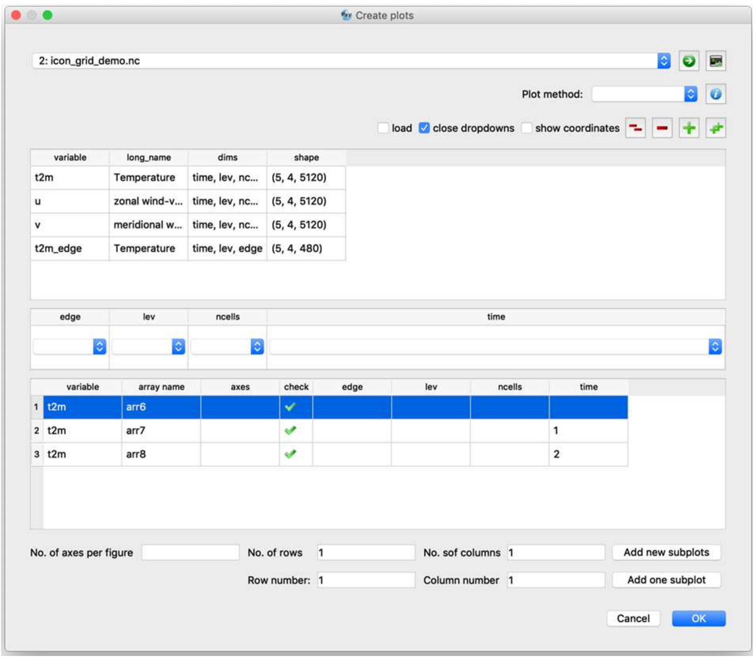
```
>>> import psyplot_gui
>>> psyplot_gui.rcParams['console.auto_set_mp'] = Fal
>>> psyplot_gui.rcParams['console.auto_set_sp'] = Fal
```

To inspect and object in the console and display it's documentation in the help explorer, type 'Ctrl + I' or a '?' after the object

Table of Contents

- datagrid

FIGURE 4.2: Screenshot of the psyplot GUI. The left part shows the content of the psyplot project, the upper center the plots, and the right part contains the help explorer. Below the plots, there is also the IPython console for the usage from the command line and a widget to update the ~~formatoptions~~ formatoptions of the current project.



variable	long_name	dims	shape
t2m	Temperature	time, lev, nc...	(5, 4, 5120)
u	zonal wind-v...	time, lev, nc...	(5, 4, 5120)
v	meridional w...	time, lev, nc...	(5, 4, 5120)
t2m_edge	Temperature	time, lev, edge	(5, 4, 480)

	variable	array name	axes	check	edge	lev	ncells	time
1	t2m	arr6		✓				
2	t2m	arr7		✓				1
3	t2m	arr8		✓				2

No. of axes per figure		No. of rows 1	No. sof columns 1
		Row number: 1	Column number 1

FIGURE 4.3: Plot creation dialog to generate new figures from an xarray dataset.

graphical user interfaces. This enables other software to develop additional features for the package (see psy-strat in the previous section 4.3.2, for instance, or straditize in chapter 3) and to flexibly change the layout of the application. The GUI is complemented with an interactive console to provide a fully integrated python environment for data analysis.

The next paragraphs provide an overview on the various widgets, that are also displayed in figure 4.2 and 4.3.

Console

The central aspects to guarantee flexibility of the application is an in-process IPython console, based on the qtconsole package[7] that provides the possibility to communicate with the psyplot package via the command line and to load any other module or to run any other script or notebook, or even to run commands in different programming languages, such as R (R Core Team, 2019) or Julia (Bezanson et al., 2017). The console is fully integrated both ways into the GUI. The documentation of every python object in the terminal, for instance, can be viewed in the help explorer of the GUI. And vice versa: a change of the current project through the project content widgets, also changes the corresponding python variable in the shell.

Help explorer

As a complement to the console, the GUI contains a help explorer to provide immediate and dynamic access to the documentation of python objects in the console, rendered as an HTML webpage[8]. Furthermore, the help explorer is connected to multiple other widgets of the GUI in order to provide a dynamically generated documentation. The documentation of available ~~formatoptions~~ *formatoptions* in the psyplot project, for instance, are rendered as HTML upon request, in order to make the various plot methods more accessible. The same principle works for the plot methods that are accessible in the plot creator.

Plot creator

The plot creator (figure 4.3) is the starting point of the GUI into the psyplot framework (at least, if one does not use the console or a script to generate the plots). It loads data from the disk or the in-process console, and essentially provides a wrapper around the psyplot plotting call (see suppl. section 4.A). It additionally displays the documentation of the method and its associated ~~formatoptions~~*formatoptions*. This widget creates new plots, that are appended to the psyplot project and are accessible through the console and the project content widgets.

Project content

The psyplot project is the most high-level API element in the psyplot framework (see section 4.3.1) and is displayed in the project content widgets of the GUI. All other elements, such as the ~~formatoptions~~ *formatoptions* widget or the plot creator, are interfering with the project, and it is accessible as a variable in the console. The project content widget can be used to see the various items in the project, but it

[7] `https://github.com/jupyter/qtconsole`

[8] The help explorer widget has been originally motivated by the *Help* widget of the Scientific PYthon Development EnviRonment, Spyder (`https://www.spyder-ide.org/`) and uses the sphinx package (Hasecke, 2019) to convert restructured Text into HTML.

is also used to select the specific items for the so-called *current* sub-project. The latter is dynamically set in the console through the sp variable and it is used by the ~~formatoptions~~ *formatoptions* widget to update the plotting parameters of the selected items.

Formatoptions

As mentioned in section 4.3.1, ~~formatoptions~~ *formatoptions* are the core elements in psyplot that control the figure aesthetics of the plots and/or perform data manipulations. The generic ~~formatoptions~~ *formatoptions* widget provides access to these parameters, in order to update them for the selected items in the current project. The formatoption itself (i.e. the python object) can in turn generate a widget that is implemented in the ~~formatoptions~~ *formatoptions* widget, to make the available options more accessible. The *title* formatoption, for instance, generates a drop-down menu to select variable attributes (e.g. variable name, variable units, etc.) which is then embedded in the ~~formatoptions~~ *formatoptions* widget. The modifications of the ~~formatoptions~~ *formatoptions* via this widgets, updates the figures of the selected items.

Figures and plots

The plots generated by the plotting methods are displayed in dedicated widgets inside the GUI and can be dynamically adjusted using the ~~formatoptions~~ *formatoptions* widget or the console. The underlying library of the current implemented psyplot plugins, matplotlib, implements multiple backends to display the data interactively, or to export them as PDF, PNG, etc. The psyplot GUI has implemented a backend on top of the PyQt5 backend of matplotlib, which embeds the figures in the GUI. psyplot can, however, work with any backend of matplotlib and does not depend on the specific implementation.

4.4 Conclusions

psyplot (Sommer, 2017e) is a new data visualization framework that integrates rich computational and mathematical software into a flexible framework for visualization. It differs from most of the visual analytic software such that it focuses on extensibility in order to flexibly tackle the different types of analysis questions that arise in pioneering research. The design of the high-level API of the framework enables a simple and standardized usage from the command-line, python scripts or jupyter notebooks. A modular plugin framework enables a flexible development of the framework that can potentially go into many different directions. The additional enhancement with a flexible GUI makes it the only visualization framework that can be handled from the conveniently command-line, and via point-click handling. It also allows to build further desktop applications on top of the existing framework.

The plugins of psyplot currently provide visualization methods that range from simple line plots, to density plots, regression analysis and geo-referenced visualization in two dimensions. The software is currently entirely based on the visualization methods of matplotlib (Hunter, 2007), the most established visualization package in the scientific python community. However, the framework itself is agnostic to the underlying visualization method and can, as such, leverage a variety of existing analytical software.

4.5 Outlook

The possibilities for further development of the psyplot framework are numerous, due to its intrinsic generality. The core of the psyplot framework will, in the future, be extended with a standardized algorithm for the generation of animations. Psyplot projects already have the functionality of being saved to a file and reloaded, but they will also be exportable as python scripts for a more flexible reusability and adaptability. The update process within a psyplot project (currently every item in the project is updated in parallel) also has potential for improvement by using a single-threaded scheduler approach that better reflects if one formatoption depends on the ~~formatoptions~~ *formatoptions* of another plotter.

The GUI has especially high potential for further development, as it still lacks widgets to quickly and intuitively modify the visual appearance of the plots. The only possibility inside the GUI (besides the console) is to use the ~~formatoptions~~ *formatoptions* widget whose main focus however is on flexibility, rather than usability and has, as such, limited possibilities for adaptation to specific use cases.

Another focus will be the development of new plot methods inside the psyplot framework. The major aspect will be on the development of 3D visualization methods of geo-referenced data, using recently published software that builds on top of the visualization toolkit VTK (Sullivan and Kaszynski, 2019; Sullivan and Trainor-Guitton, 2019), see Sommer, 2019b. psyplot has the unique potential to generate 3D visualizations conveniently from the command line, a distinguishing feature, compared to other visualization software packages, such as ParaView or Vapor. Further potential enhancements for visualizations can involve standard interactive visual analytic tools, e.g. such that the interactive selection of features in one plot affects the visualization in another plot (so-called brushing and linking).

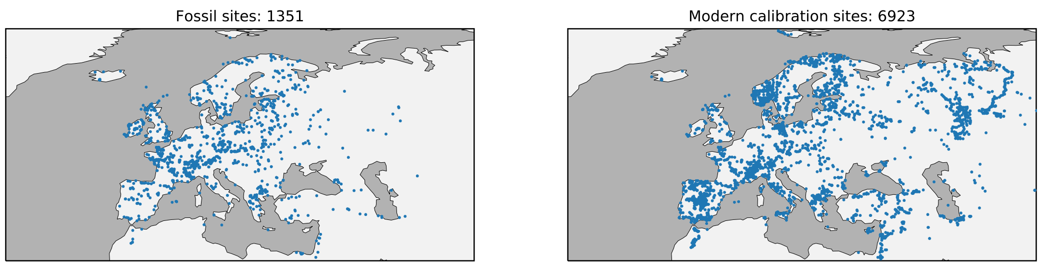


FIGURE 5.1: Site locations of the (left) fossil and (right) modern pollen database.

Our new method applies a probability approach to data integration, whereby the full uncertainty of each sample is considered in the gridding process, rather than just the sample mean used in previous methods (Mauri et al., 2015). From this, we can calculate the uncertainties associated with the temporal and spatial distances between our reconstruction samples/sites and the points on the grid network. We do this through an ensemble bootstrapping approach in which we repeatedly grid the data, each time using a different set of samples that are randomly ~~generated~~ selected to reflect the reconstruction and chronological uncertainties of the site network. In this study, we use pollen-data, which provides the most accessible and spatially distributed paleo-climate data available for the late-Quaternary period.

The strength of this new method is that it treats all of the paleo-climate samples and sites as a single integrated paleo-climate record from which it is possible to extract a single regionally coherent climatic reconstruction, complete with uncertainties. Pollen-based reconstructions often have high sample to sample uncertainties, and the vegetation at individual sites can be influenced by non-climatic factors such as soils, disease, fire, migration lag and human impact. Our method allows us to fully utilize the large quantity of pollen-data that is available, to extract the regional background climate signal from what may be locally quite noisy data. It also has a significant advantage over other methods that integrate records using Bayesian approaches (e.g. Holmström et al., 2015), in that it is much more computationally efficient, making it possible to undertake analysis at continental scales with hundreds and thousands of sites and samples.

5.2 Data

The ensemble based gridding method is adapted to paleo-climates. In this study, we describe the method using a large set of western Eurasian fossil pollen assemblages that have been transformed to summer (June, July and August) (JJA) temperatures. We focus on pollen data because it is the spatially most widely available proxy during the Holocene, but it is important to mention that the reconstruction method is agnostic to the climate proxy, because it does not explicitly use the pollen assemblages but rather alters the standard climate reconstruction method under the aspect of its methodological uncertainties. As such, the following sections describe the fossil and modern pollen database for this use case (section 5.2.1) and the associated uncertainties of the temperature reconstruction method (section 5.2.3) and the dating of the fossil pollen samples (section 5.2.4).

al., 2003; Mauri et al., 2015, e.g.). They all, however, have in common that the categorical, multi-modal distribution of the climate of the modern analogues is simplified into a unimodal distribution, represented by the mean of the analogue climates. Therefore, in our ensemble approach, we do not take the mean but sample the climate of the analogues directly. This is further discussed in the methods section 5.3.2 and 5.4.1.

5.2.4 Age uncertainties

In addition to the methodological uncertainties of the climate reconstruction method (previous section 5.2.3), we provide a framework to handle dating uncertainties. During the gridding step (see next section 5.3.3), every sample is weighted by the age difference to the target reconstruction age. The previous studies by Davis et al., 2003 and Mauri et al., 2015 do not take this uncertainty, that can be as high as multiple centuries, into account although they influence the gridded temperature reconstruction.

The reason is a systematic problem of pollen samples that we overcome here with the recent developments in the pollen community. In palynology, each sample in a sediment core is is dated using a so-called age-depth model, a function that maps each depth of the sediment core to an age. This function is based on a few chronological control points where the age has been determined instrumentally (for lake sediments in the Northern Hemisphere, these are commonly radiocarbon (^{14}C dates) and interpolates/extrapolates to the depths of the sample locations. Various methodologies exist to define these age-depth models, ranging from simple linear interpolation methods (Bennett, 1994) to the more recently developed bayesian techniques of the Bchron (Haslett and Parnell, 2008) and BACON (Blaauw and Christen, 2011) models.

The early approaches have been proven to provide unreliable uncertainty estimates (Telford et al., 2004) and there has been no standardized way to report the uncertainties, if they are reported at all. For this reason we (and previous studies) cannot rely on the age uncertainties reported in the pollen database. An alternative approach is to recalculate the chronology for every dataset in the database (see Goring, 2019, for instance), but this also requires parameterization for reliable uncertainties and goes beyond the scope of this study.

Instead, we follow an approach that is based on two aspects: age uncertainties are higher for older samples, and samples that are farther away from the radiocarbon dates (i.e. chronological control points). Additionally, samples behave differently if the sample is surrounded by two chronological points (i.e. the sampe age is interpolated) or not (sample age is extrapolated). To quantify these relationship, we perform a study based on all datasets (ca. 30’000 samples) from the Neotoma paleoecology database (Williams et al., 2018) that have age-depth models estimated with BACON, a model that has been proven to provide more reliable age uncertainty estimates (Trachsel and Telford, 2016). For the sake of implementation (the age sampling in section 5.3.1 assumes a normal distribution), we apply several assumptions and ~~proximations~~ approximations to the Neotoma samples, in particular:

1. We assume that every dataset with a BACON chronology in Neotoma keeps the defaults and reports the limits of the 95% confidence interval (CI)
2. We keep only the maximal distance of the CI limits from the reported age (i.e. we assume a symmetric distribution)

based on all samples with BACON-based chronologies from the Neotoma database under the assumption that they all report the age uncertainties as the 95th percent confidence interval. This strong assumption can be relaxed for future improvements of this method by using the very recent peer-reviewed dataset by Wang et al., 2019 which contains standardized chronologies for more than 500 datasets.

The other side-product, is a probabilistic variant of the site-based modern analogue technique (MAT) that uses a Gibbs sampling algorithm for the ages and analogue climates in order to approximate the joint distribution within a single dataset. This sampling algorithm is constrained for the individual dataset through first, the monotonicity of the sample ages, and second, a climatic threshold that must not be overcome between too temporally neighboring samples. We compare this sampling algorithm to computationally faster algorithms that involve a forward sampling (climate/age of a sample is constrained by its younger predecessor), its inversion, the backward sampling, as well as a combined approach that uses forward and backward sampling. For age sampling we also test an algorithm that starts with an unconstrained sampling of the ages and then applies an a posteriori sorting in order to maintain the correct distribution of samples in the core (sections 5.3.1 and 5.3.2). All of these approaches, however reveal biases when compared to the computationally more demanding, but ~~stationary~~ statistically correct distribution of the Gibbs sampler. The software package *pyleogrid* therefore implements a computationally efficient version of this Gibbs sampling algorithm that efficiently scales from one single dataset to tenth of thousands of realizations of more than a thousand individual datasets, as it has been used in this study.

This sampling successfully reconstructs a probabilistic version of the individual dataset and provides reliable uncertainty estimates. An evaluation of this realization with respect to the number of modern analogues, and the climatic constraint of the sampling algorithm reveals a close linkage of the two parameters. Further developments might therefore explore the usage of other proxy-climate reconstruction methods that provide a more continuous distribution to sample from.

wet days), 3-4) monthly mean daily minimum and maximum temperature, 5) mean cloud fraction, and 6) wind speed. The model outputs are the same variables at daily resolution. This section summarizes the basic workflow in the model which is also shown schematically in Figure 6.1 and Algorithm 2.

The first approximation of the daily variables comes from smoothing the monthly time series using a mean-preserving algorithm (Rymes and Myers, 2001).

For precipitation we then first use the Markov Chain approach (section 6.3.2) to decide the wet/dry state of the day. If it is a wet day, we calculate the gamma parameters using the equations (6.7) and (6.8). The resulting distribution allows us to draw a random number, the precipitation amount of the currently simulated day. If we are above the threshold μ, we draw a second random number from the ~~GP~~ Generalized Pareto (GP) distribution parameterized via equation (6.9) and the chosen GP shape.

The next step modifies the means of temperature, wind speed and cloud fraction depending on the wet/dry state of the day (lines 11 and 15 in algorithm 2). After that, we use the cross-correlation approach described in Richardson, 1981 (lines 18 - 20 and Equation 6.3.2) and calculate the daily values of these variables. Finally we use the quantile-based bias correction described in section 6.3.4 to correct the simulated wind speed.

We restrict the weather generator to reproduce the exact number of wet days (±1) as the input and to be within a 5% range of the total monthly precipitation (with a maximum allowed deviation of 0.5 mm). If the program cannot produce these results, the procedure described above is repeated (see line 4).

6.3 Model development

GWGEN is based on the WGEN weather generator (Richardson, 1981), using the method of defining the model parameters based on monthly summaries described by Geng et al., 1986 and Geng and Auburn, 1987. GWGEN diverges from the original WGEN by using a hybrid-order Markov chain to simulate precipitation occurrence (Wilks, 1999a), and a hybrid Gamma-GP distribution (Furrer and Katz, 2008; Neykov et al., 2014) to estimate precipitation amount. Temperature, cloud cover, and wind speed are calculated following (Richardson, 1981), using cross correlation and depending on the wet/dry-state of the day. We further add a quantile-based bias correction for wind speed and minimum temperature, which improves the simulation results significantly.

In the following subsections, we first describe the global weather station database used to develop and evaluate the model, then describe the underlying relationships that we use to define GWGEN's parameters.

6.3.1 Development of a global weather station database

To parameterize GWGEN, we assembled a global dataset of daily meteorological observations. Precipitation and minimum and maximum daily temperature come from the daily Global Historical Climatology Network (GHCN-Daily) database (Menne et al., 2012a,b). The GHCN-Daily consists of observations collected at ca. 100'000 weather stations on all continents and many oceanic islands. As the GHCN-Daily stations are highly concentrated in some parts of the world, particularly in the conterminous United States, we selected stations for our study using a geographic anti-aliasing filter to avoid an especially strong geographic bias in the generation of the