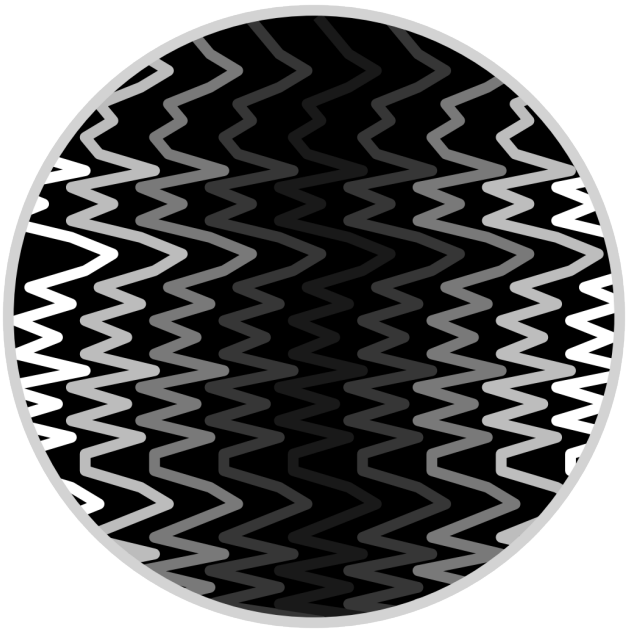
straditize Documentation

Release 0.1.3

Philipp Sommer

Nov 14, 2019

CONTENTS



STRADITIZE (Stratigraphic Diagram Digitizer) is an open-source program that allows stratigraphic figures to be digitized in a single semi-automated operation. It is designed to detect multiple plots of variables analyzed along the same vertical axis, whether this is a sediment core or any similar depth/time series.

Usually, in an age of digital data analysis, gaining access to data from the pre-digital era – or any data that is only available as a figure on a page – remains a problem and an under-utilized scientific resource.

This program tackles this problem by providing a python package to digitize especially pollen diagrams, but also any other type of stratigraphic diagram.

The package is very new and there are many features that will be included in the future. Therefore we would be very pleased to get feedback! To do so, you can contact us or raise an issue on GitHub.

CHAPTER

ONE

DOCUMENTATION

1.1 About straditize

1.1.1 Why straditize?

In an age of digital data analysis, gaining access to data from the pre-digital era – or any data that is only available as a figure on a page – remains a problem and an under-utilized scientific resource. Whilst there are numerous programs available that allow the digitization of scientific data in a simple x-y graph format, we know of no semi-automated program that can deal with data plotted with multiple horizontal axes that share the same vertical axis, such as pollen diagrams and other stratigraphic figures that are common in the Earth sciences. STRADITIZE (Stratigraphic Diagram Digitizer) is a new open-source program that allows stratigraphic figures to be digitized in a single semi- automated operation. It is designed to detect multiple plots of variables analyzed along the same vertical axis, whether this is a sediment core or any similar depth/time series.

1.1.2 About the author

The code and GUI of straditize was developed by Philipp S. Sommer at the Institute of Earth System Dynamics (IDYST) at the University of Lausanne as part of the SNF funded HORNET Project (200021_169598).

The other contributors are Basil A. S. Davis, Manuel Chevalier and Dilan Rech who made significant contributions to the layout, workflow, beta tests and and reviewing of the software.

1.1.3 License

1.2 Installation

1.2.1 How to install

You can either install straditize through a package manager such as *conda* or *pip* or install it *from source*.

Installation using conda

We highly recommend to use conda for installing straditize. Here you can install it via manually via the chilipp channel

After having downloaded and installed anaconda, open a terminal (or the *Anaconda Prompt* on windows) and install straditize from the conda-forge channel. You can choose: We recommend to install straditize into its own environment via:

```
$ conda create -n straditize -c conda-forge straditize
```

and then activate this environment via:

```
$ conda activate straditize
```

In that way you do not mess up your base environment. Nevertheless you can also install it into an existing environment via:

```
$ conda install -c conda-forge straditize
```

In the same terminal, now type `straditize` to start the software.

Note: The latest master branch on github is always available under the `master` label on the chilipp channel. Just type:

```
$ conda install -c chilipp/label/master straditize
```

to install the latest version from the master branch. Note that you then have to add the *conda-forge* channel to your default channels via:

```
$ conda config --add channels conda-forge
```

Installation using pip

If you do not want to use conda for managing your python packages and already have python3 installed on your computer, you can also use the python package manager `pip`. To be on the safe side, make sure you have the *Dependencies* installed. If so, open a terminal and install it via:

```
$ pip install straditize
```

To open the software, type `straditize` in the same terminal.

Installation from source

To install it from source, make sure you have the *Dependencies* installed. Download (or clone) the github repository, e.g. via:

```
git clone https://github.com/Chilipp/straditize.git
```

and install it via:

```
pip install .  # or python setup.py install, but pip is recommended
```

from your terminal. To open the software, type `straditize` in the same terminal.

1.2.2 Dependencies

Required dependencies

straditize has been tested for python>=3.6. Furthermore the package is built upon multiple other packages, mainly

- psyplot-gui>1.2.0: The graphical user interface for psyplot
- PyQt5: Pythons Qt bindings that are required by psyplot-gui (note that PyQt4 is **not** supported!)
- numpy, scipy and pandas: for the data management and compuations
- matplotlib>=2.0: **The** python visualiation package
- pillow: for reading and writing images
- scikit-image: For image recognition features
- openpyxl: For exports to Excel files
- netCDF4: A library for saving and storing netCDF files.

Optional dependencies

We furthermore recommend to use

- tesserocr: for column names recognition. It depends on the tesseract OCR and you can install both (on Linux and MacOS) via:

  ```
  $ conda install -c chilipp tesserocr
  ```

  (see *Automatic optical character recognition (OCR)* for more information)

1.2.3 Running the tests

We use pytest to run our tests. So you can either run clone out the github repository and run:

```
$ python setup.py test
```

or install pytest by yourself and run

    $ py.test

Alternatively you can build the recipe in the *conda-recipe* directory via

    $ conda build conda-recipe

which will also run the test suite.

Warning: Running the entire test suite in one single process (such as `python setup.py test`) might be quite memory consumptive because it involves the creation and closing of many PyQt widgets and unfortunately some memory is leaked from one test to another. Therefore we recommend to split the tests into multiple processes, e.g.:

```
# run the test suite but ignore some modules
python setup.py test -a '--ignore=tests/widgets/test_selection_toolbar.py --
↪ignore=tests/widgets/test_samples_table.py --ignore=tests/widgets/test_beginner.
↪py --ignore=tests/widgets/test_hoya_del_castillo.py'
# run the tests for the previously ignored modules
python setup.py test -a 'tests/widgets/test_selection_toolbar.py
tests/widgets/test_samples_table.py'
python setup.py test -a 'tests/widgets/test_beginner.py'
python setup.py test -a 'tests/widgets/test_hoya_del_castillo.py'
```

or equivalently with *py.test* instead of *python setup.py test -a*. Note that *conda build conda-recipe* already splits the session into multiple processes.

Nevertheless, you should expect about ~180 tests to be ran and a total memory usage of about 3 to 4GB RAM.

1.2.4 Building the docs

The online documentation is accessible as PDF, HTML and Epub under https://straditize.readthedocs.io or https://straditize.rtfd.io. Thanks to the free services by readthedocs.org, the online documentation is build automatically after each commit to the github repository.

To build the docs locally on your machine, check out the github repository and install the requirements in `'docs/environment.yml'` and the sphinx_rtd_theme package. The easiest way to do this is via anaconda by typing:

```
$ conda env create -f docs/environment.yml
$ conda activate straditize_docs
$ conda install sphinx_rtd_theme
```

Then build the docs via:

```
$ cd docs
$ make html  # or `make pdf` for a PDF version compiled with Latex
```

1.2.5 Updating straditize

Updating the software depends on how you installed it on your system.

Updating via conda

If you installed straditize via conda (see *Installation using conda*), you can update it via:

```
$ conda update -c chilipp straditize
```

Updating via pip

If you installed it via `pip` (see *Installation using pip*), you can update it via:

```
$ pip install -U straditize
```

Updating from source files

If you installed it via `python setup.py install` from the source repository (see *Installation from source*), just run that command again after having checked out the latest version from github.

1.2.6 Uninstallation

The uninstallation depends on the system you used to install straditize. Either you did it via *conda* (see *Uninstallation via conda*), via *pip* or from the *source files* (see *Uninstallation via pip*).

Anyway, if you may want to remove the psyplot configuration files. If you did not specify anything else (see `psyplot.config.rcsetup.psyplot_fname()`), the configuration files for psyplot are located in the user home directory. Under linux and OSX, this is `$HOME/.config/psyplot`. On other platforms it is in the `.psyplot` directory in the user home.

Uninstallation via conda

If you installed straditize via *conda*, simply run:

```
conda uninstall straditize
```

Uninstallation via pip

Uninstalling via pip simply goes via:

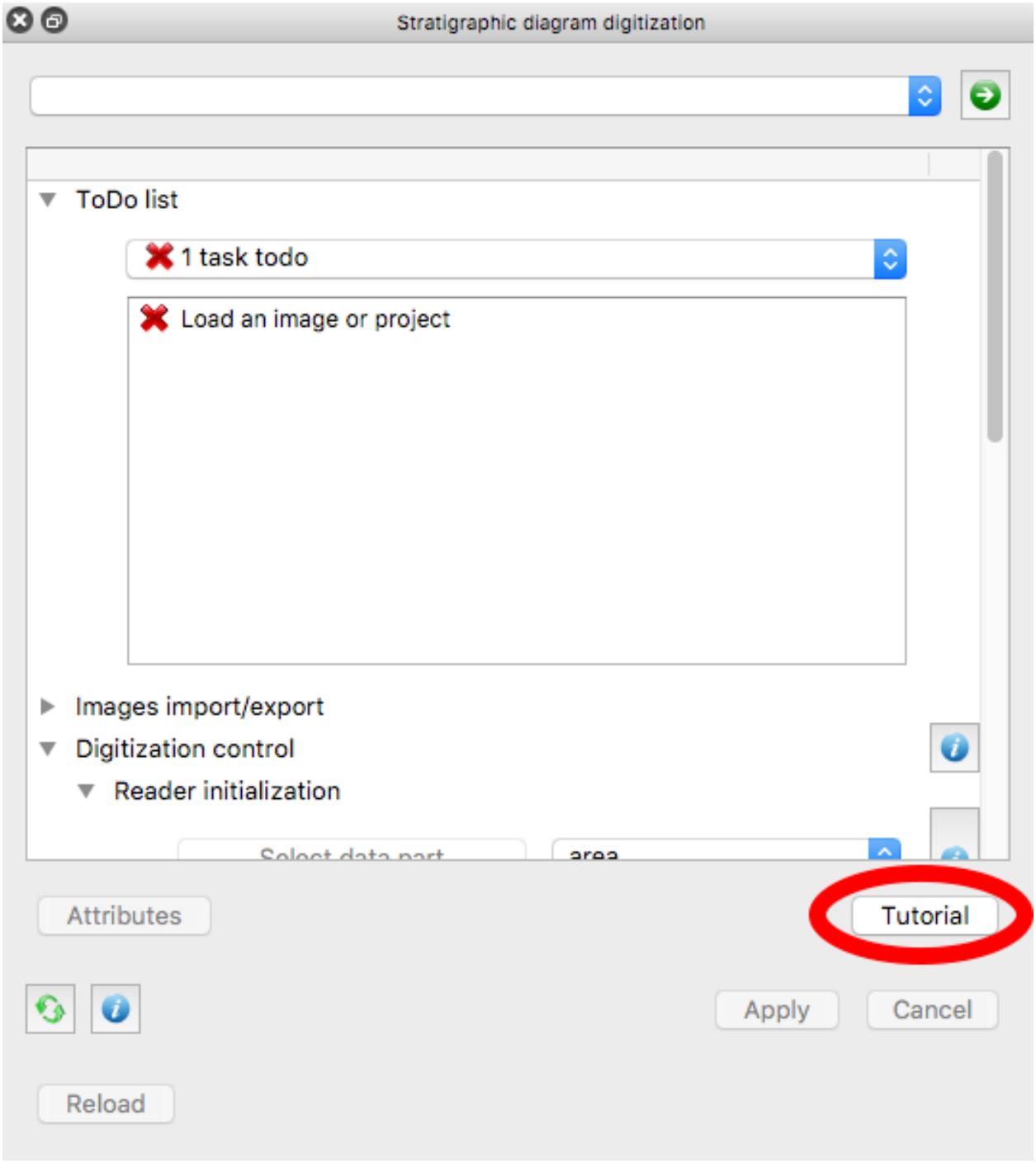
```
pip uninstall straditize
```

Note, however, that you should use *conda* if you also installed it via conda.

1.3 Straditize Tutorial

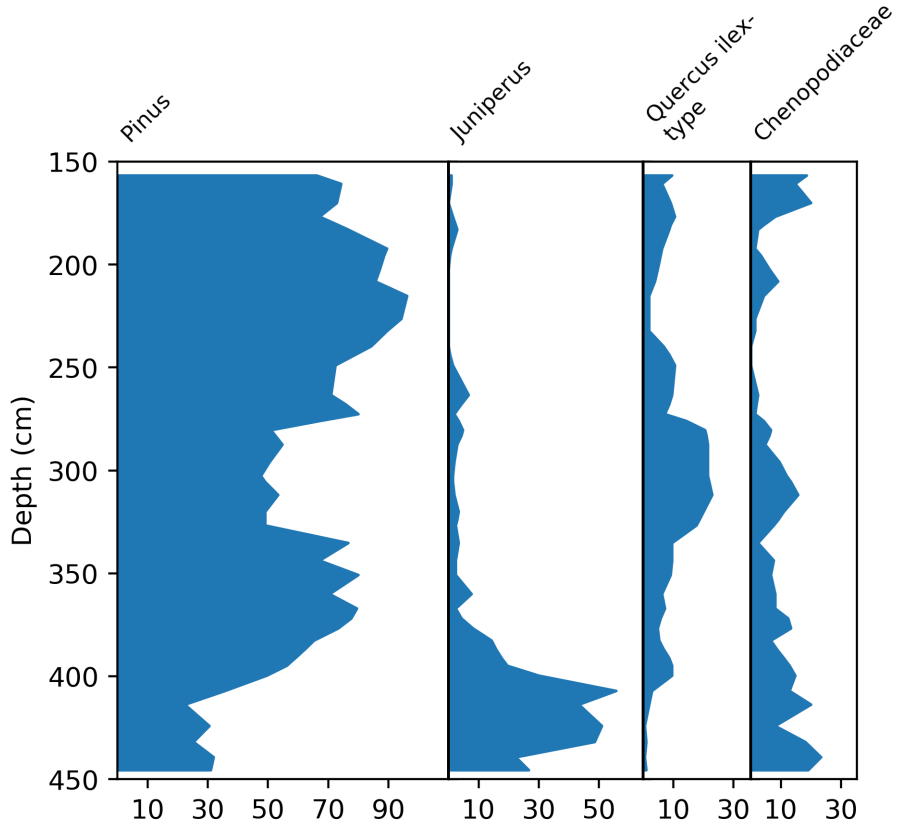
To introduce you into straditize, we implemented a tutorial into the graphical user interface.

To start it in the GUI, click the *Tutorial* button at the bottom of the straditizer control.



1.3.1 Tutorials

Beginners tutorial to introduce straditize



In this tutorial, we digitize a simple small pollen diagram a subset of the Hoya del Castillo dataset from [Basil2007a]. This tutorial will introduce you into the main parts of straditize and the basic workflow.

At the bottom of the straditizer control, you find a navigation panel which helps you to navigate through the tutorial. Click the *Next* button in this control to start the tutorial. You can skip steps by clicking the *Skip* button and you can check what you have done using the *Check* button and then proceed with the next step.

The tutorial will take between 5 and 10 minutes. You can save your current state to a file using *File* → *Save* → *Save straditizer* (see *Saving and loading your project*) and reload it later via *File* → *Open project* → *Open straditizer* → *Project or image*

If you have questions or troubles with this tutorial, please open an issue on

https://github.com/Chilipp/straditize/issues

and we will do our best to assist you.

The straditize workflow

The straditize workflow, i.e. the program for the next few minutes consists of 6 main steps:

1. Load the diagram
2. Select the diagram part (withouth x-axes labels, y-axes labels, etc.)
3. Clean the diagram part. Only the data parts should be left over, i.e. no y-axes, lines, letters, etc.

4. Digitize the diagram
5. Find and edit the samples
6. Export the data to Excel or CSV

Most of these tasks can be done in an automatic way but you should always review and edit the outcome to make sure that what you do is scientifically reliable.

The interface to these semi-automatic steps is the straditizer control

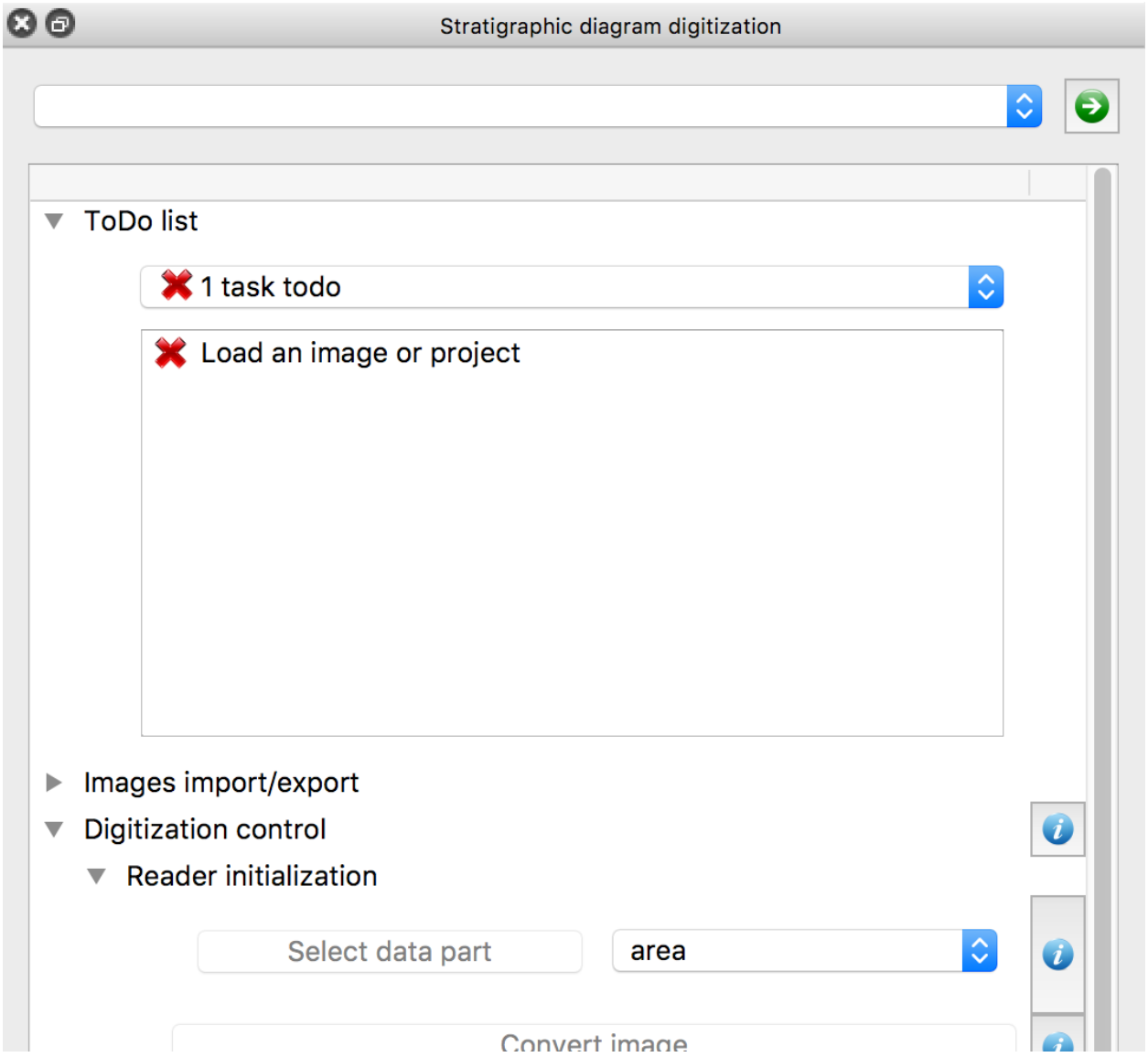


Fig. 1: Click the image to enlarge it

You can expand the submenus using the ▸ buttons in the control and you can access the user manual using the buttons on the right side.

Before we start with the diagram, we will highlight some of the sections in the GUI:

The ToDo list The list at the top of the control guides you through the *straditization* process. It shows the open tasks you should consider.

Most of them will be marked as done automatically, but if you think you are done with one of the tasks, just right-click the task and mark it as done.

The Plot control This section at the bottom of the control gives you some visual explorations of your diagram and the digitization process. You should, whenever possible, use it's functionalities, most of the errors can be spotted visually

The Help explorer This separate widgets can be used to access the user manual and provides guidance for the different functionalities. This should always be your first point of reference if you have any problems.

Load the image of the diagram

The first step is to load the image of the diagram into straditize.

1. Click the  button at the top of the straditizer control.
2. Select the image file `beginner-tutorial.png` from your hard-disk and click Open

Navigation inside the plot

Now you see the stratigraphic diagram (by default) centered at the top of the graphical user interface. Additionally you find a separate window to it's right with a magnified version of the diagram. It follows your mouse movements inside the diagram and shows you a magnified version of it.

To navigate inside the plot, you can use the default navigation toolbar that is provided by matplotlib

Especially the Pan/Zoom button  and the zoom-to-rectangle button  are of interest for you. You can enable and disable them by clicking on the corresponding button in the toolbar.

The matplotlib docs provide further guidance:

The `Pan/Zoom` button This button has two modes: pan and zoom. Click the toolbar button to activate panning and zooming, then put your mouse somewhere over an axes. Press the left mouse button and hold it to pan the figure, dragging it to a new position. When you release it, the data under the point where you pressed will be moved to the point where you released. If you press 'x' or 'y' while panning the motion will be constrained to the x or y axis, respectively. Press the right mouse button to zoom, dragging it to a new position. The x axis will be zoomed in proportionately to the rightward movement and zoomed out proportionately to the leftward movement. The same is true for the y axis and up/down motions. The point under your mouse when you begin the zoom remains stationary, allowing you to zoom in or out around that point as much as you wish. You can use the modifier keys 'x', 'y' or 'CONTROL' to constrain the zoom to the x axis, the y axis, or aspect ratio preserve, respectively.

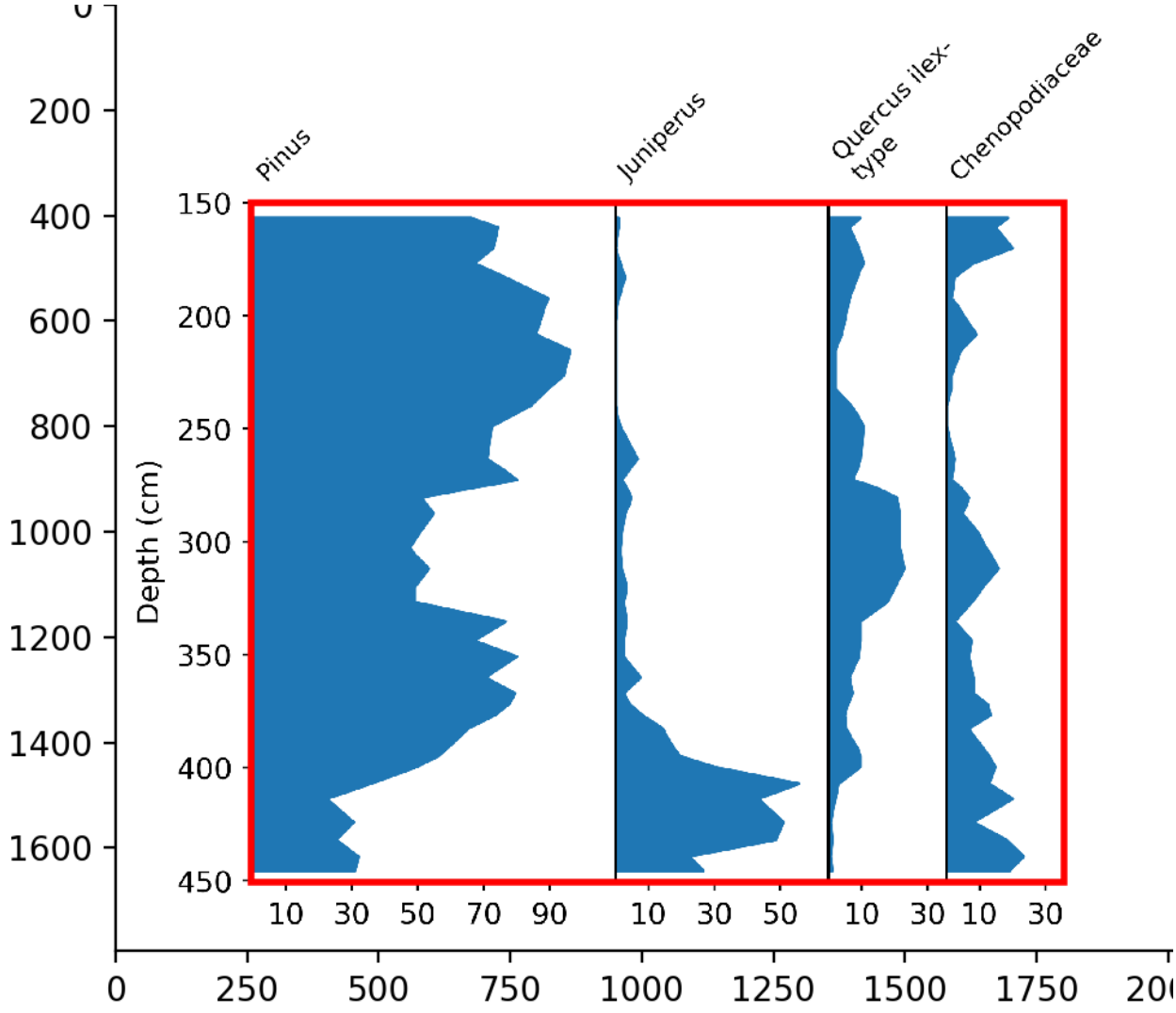
With polar plots, the pan and zoom functionality behaves differently. The radius axis labels can be dragged using the left mouse button. The radius scale can be zoomed in and out using the right mouse button.

The `Zoom-to-rectangle` button Click this toolbar button to activate this mode. Put your mouse somewhere over an axes and press a mouse button. Define a rectangular region by dragging the mouse while holding the button to a new location. When using the left mouse button, the axes view limits will be zoomed to the defined region. When using the right mouse button, the axes view limits will be zoomed out, placing the original axes in the defined region.

More information can be found in the matplotlib documentation: https://matplotlib.org/users/navigation_toolbar.html

Select the diagram part

The second step involves selecting the diagram part. This is the part where your data is displayed, but without the vertical and horizontal axes descriptions.



Being exact in this step significantly reduces the later work and simplifies the automatic digitization.

1. Click the *Select data part* button in the digitization control.
2. Straditize automatically recognizes the data part. But you can change this by moving the crosses around with your mouse (see the note below).
3. Click the *Apply* button at the bottom
4. Done! You now see a red rectangle in your diagram the marks the data part. You can hide it using the *Plot control* section in the straditizer control. For the sake of this tutorial, expand the *Plot control* section and remove the rectangle by clicking the  icon for the *Diagram part*

Note:

These marks are very common in straditize. In general, you can left-click on a mark to move it around, right-click to delete it, and Shift+left-click on the plot to create a new one.

Note that you need to disable the  and  tools in the navigation to interact with the marks, by clicking on the corresponding button!

If you want to change the appearance of the marks, see the *Marker control* section in the straditizer control panel.

Create the diagram reader

Now you have to select, which type of diagram you are digitizing. In the example, it is an area diagram which is already selected in the dropdown menu next to the *Select data part* button.

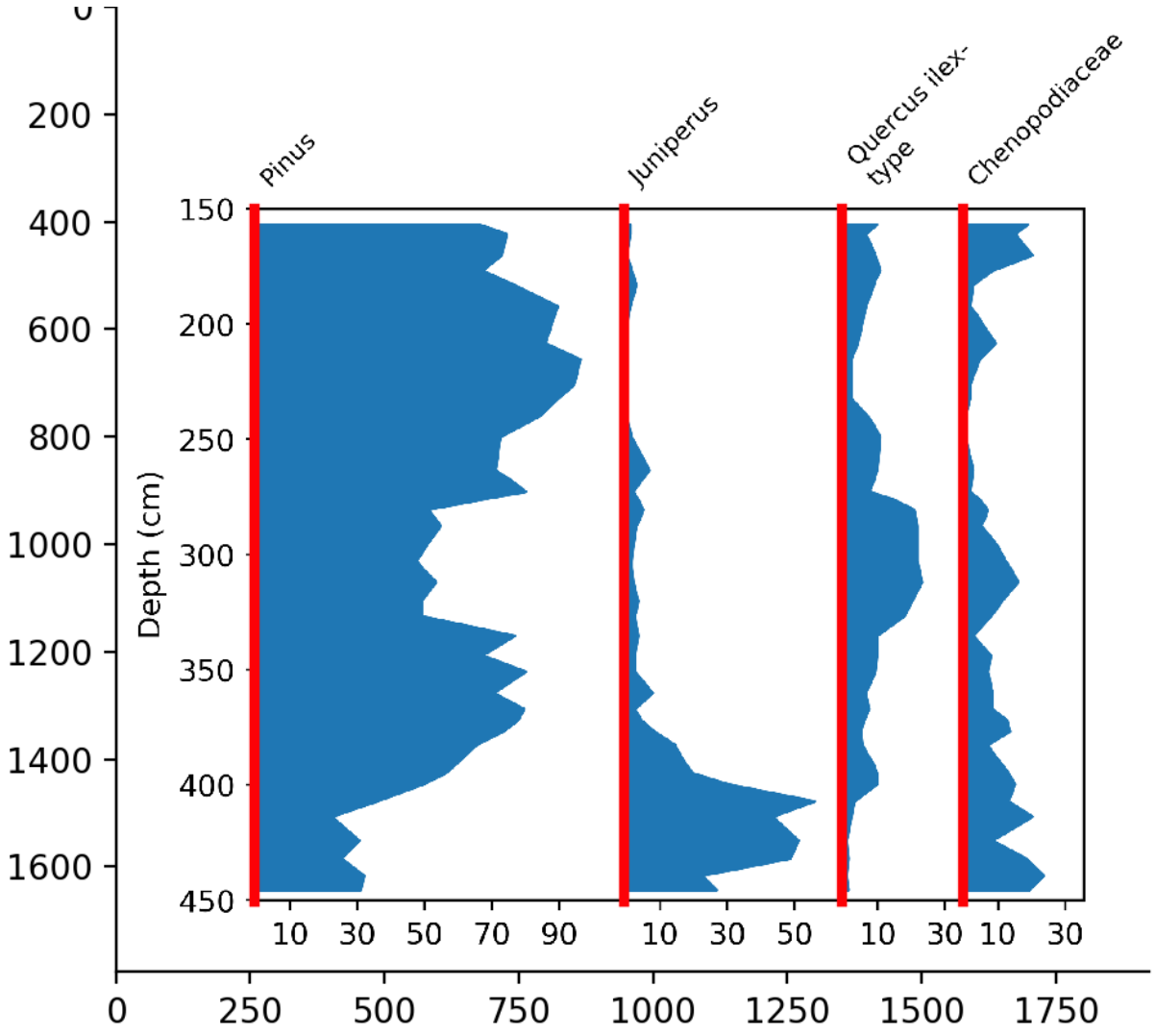
Click on *Convert image* to start the digitization.

See also:

Selecting the reader

Selecting the columns

The next step is, to separate the columns, i.e. to tell straditize where each of the sub diagrams start.



This is simple in our case, since straditize can separate them automatically.

1. Click the *Column starts* button. You will see several vertical blue lines appearing on the plot that you can drag and drop such as you did it when selecting the diagram part. You can change their colors using the *marker control*
2. straditize recognizes these columns automatically and in our example, you do not have to edit them. Therefore hit the *Apply* button at the bottom of the control and you are done.

See also:

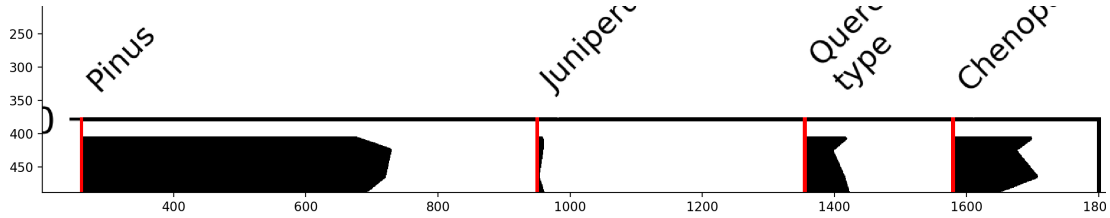
Select the starts for each column

Cleaning the image

The next step is to clean the data image to make sure that everything that is not representing real data is removed. In our case, these are y-axes, x-axes and the outer frame. You can also use an external image editing software such as Adobe Photoshop, but we will do this now inside straditize and use some of the automatic recognition functionalities.

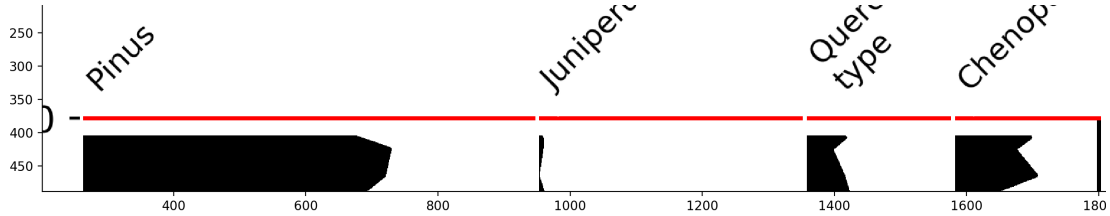
1. Expand the *Remove features* tab

2. Click the y-axes button, the y-axes in the plot will be highlighted



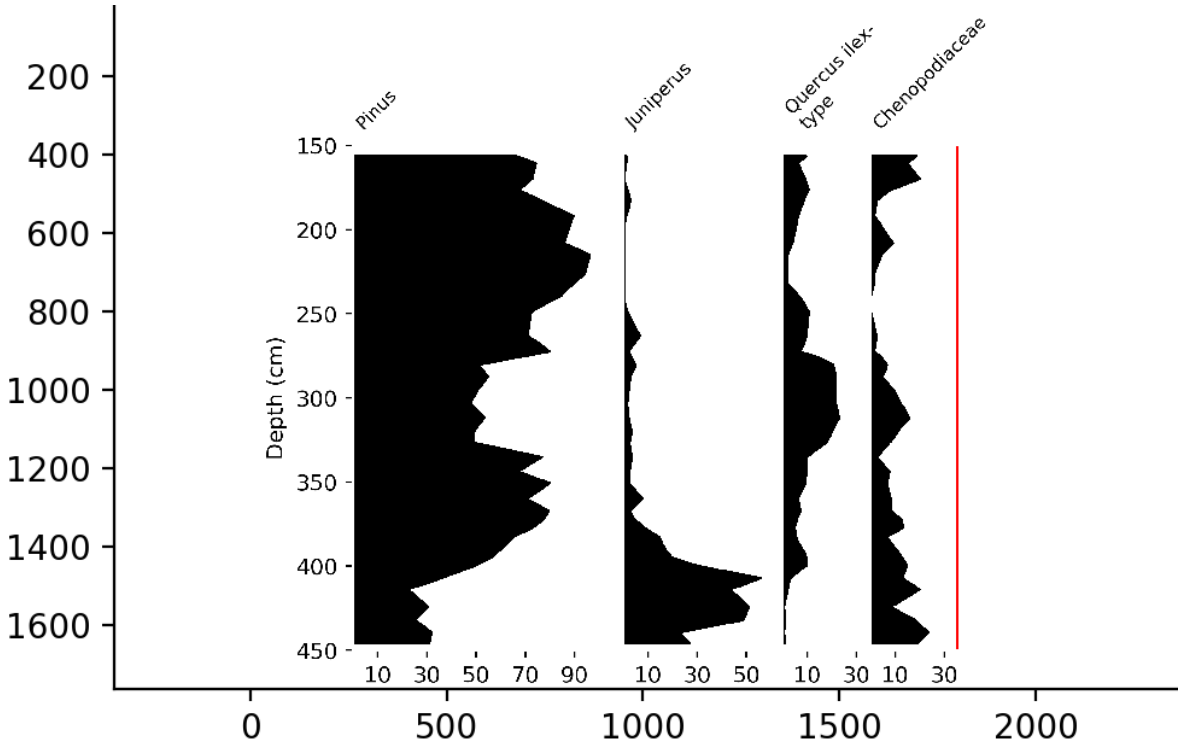
3. Click *Remove* to remove them

4. 2. Click the x-axes button, the x-axes in the plot will be highlighted



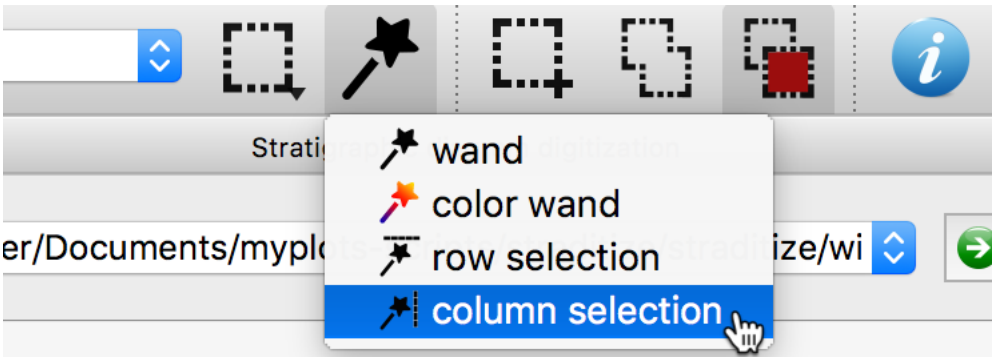
5. Click *Remove* to remove them

6. Finally there is the right part of the diagram frame left

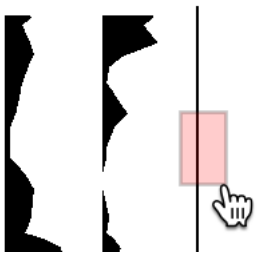


For this, we use the selection toolbar:

i. from the menu, select the *column selection* tool (click and hold the button to the right of the button in the selection toolbar to open the menu)



ii. Draw a rectangle around the right line



and make sure you have the mode activated

iii. Click the *Remove* button when you are done

Digitizing the data

The next step, after we cleaned up the image, is the digitization of the diagram. Click the *Digitize* button.

You now see the lines that result from the digitization. You can remove them in the *Plot control* section by clicking the button of the *Full digitized data* row.

Finding and editing samples

So far, we have one data point per pixel in the image. However, we have to identify the locations of the samples in order to reproduce the original data.

Straditize assists you with this through automatic sample finding algorithms, or you can load the sample locations from an external files or you just add and edit the samples manually.

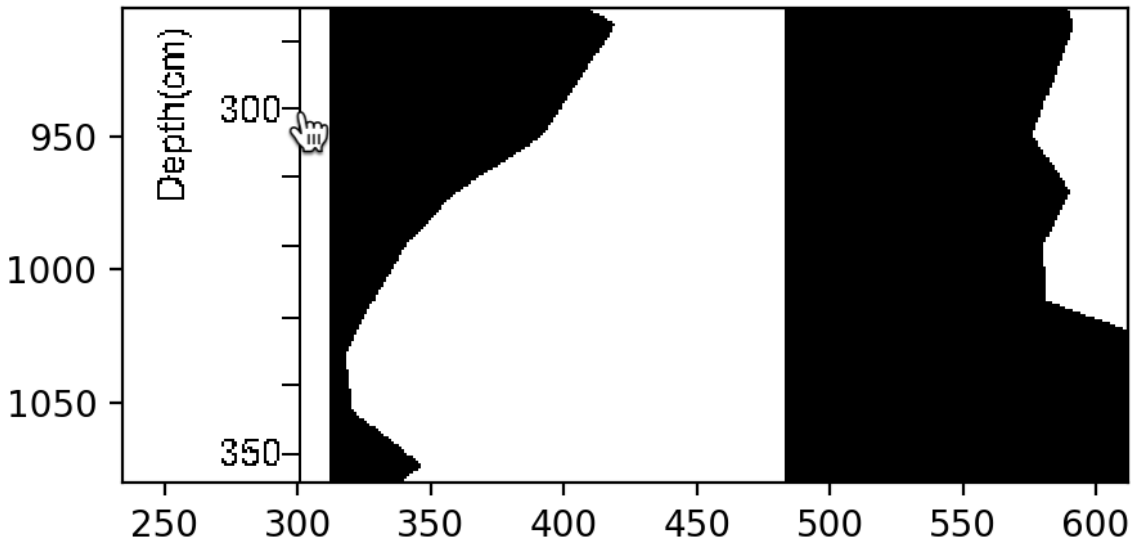
1. Expand the *Samples* item in the digitization control
2. Find the samples by clicking the *Find samples* button. straditize now identified the sample locations based on the extrema in the columns (see *Automatic samples identification* for an explanation of the algorithms).
3. To visualize the samples, you can again use the items in the *Plot control* tab. However, you can (and should!) also edit them by clicking the *Edit samples* button.
4. Now you see one horizontal line per sample that you can drag around (left-click), delete (right-click) or you can also add new samples (Shift + left-click). See the *Editing samples* section for more details.
5. Finally, click the *Apply* button or the *Cancel* button to stop the editing of the samples.

You're almost done! Your diagram is now digitized and the data could already be exported.

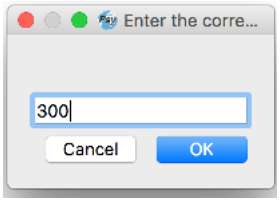
Translating the y-axis

To correctly reproduce the diagram data, there are now only to more things to know, that is the scaling of the y- and x-axes in the diagram.

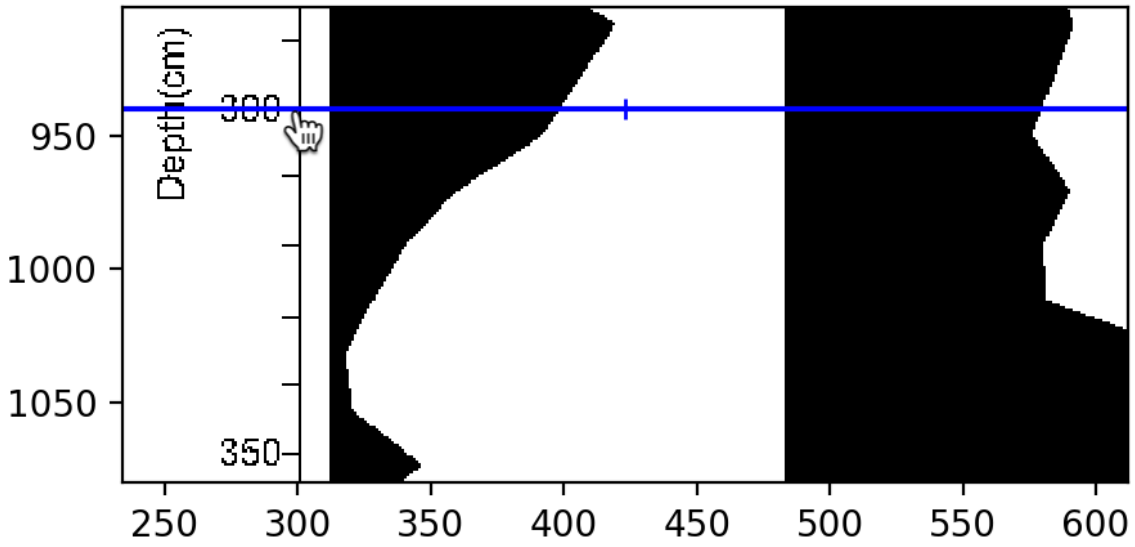
1. Expand the *Axes translations* tab in the digitization control
2. Click the *Insert Y-axis values* button in the *Axes translations* section of the straditizer control
3. Shift-leftclick on the plot to enter the corresponding y-value.



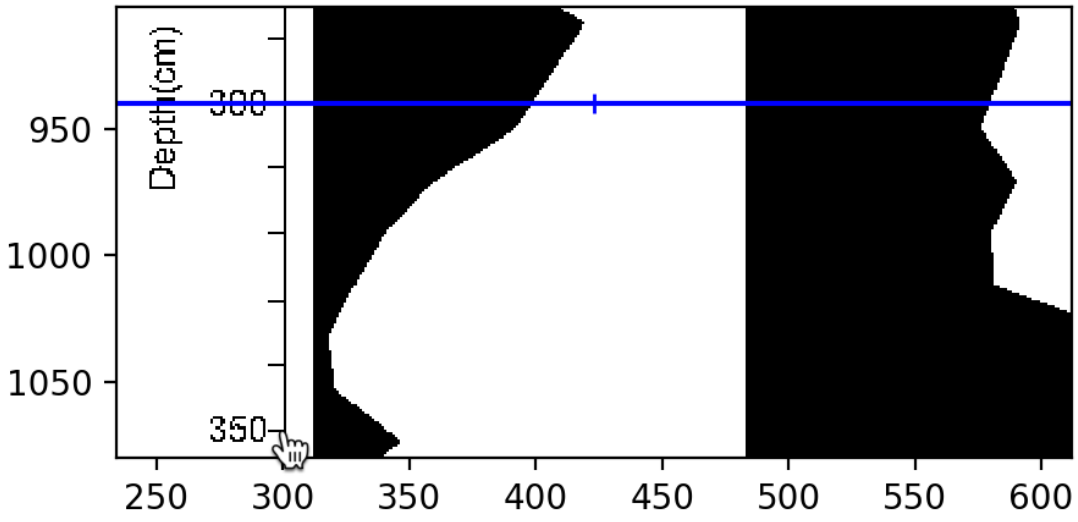
4. A small dialog will appear where you should enter the y-value to use (in this case, `300`)



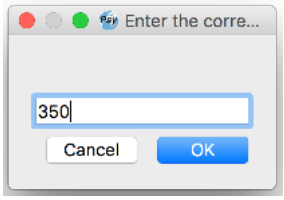
5. After hitting the *Ok* button, you will see a mark on the plot (blue line). You can select the mark via leftclick and drag it to a different location or you can delete it via rightclick.



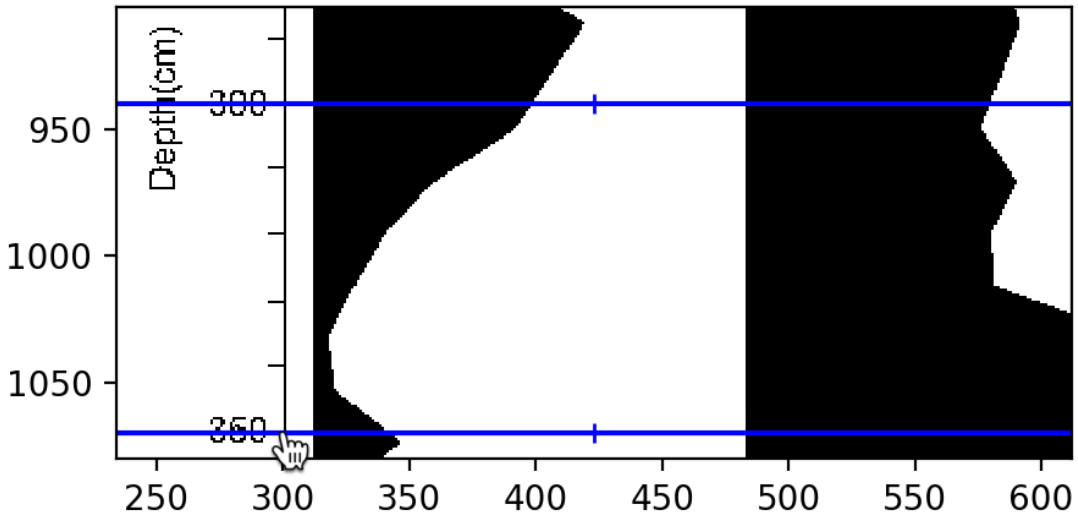
6. now repeat steps 2-4 on a second point on the y-axis
   - Select another point



   - Enter the corresponding value (here `350`)



   - A new mark is created that you can modify



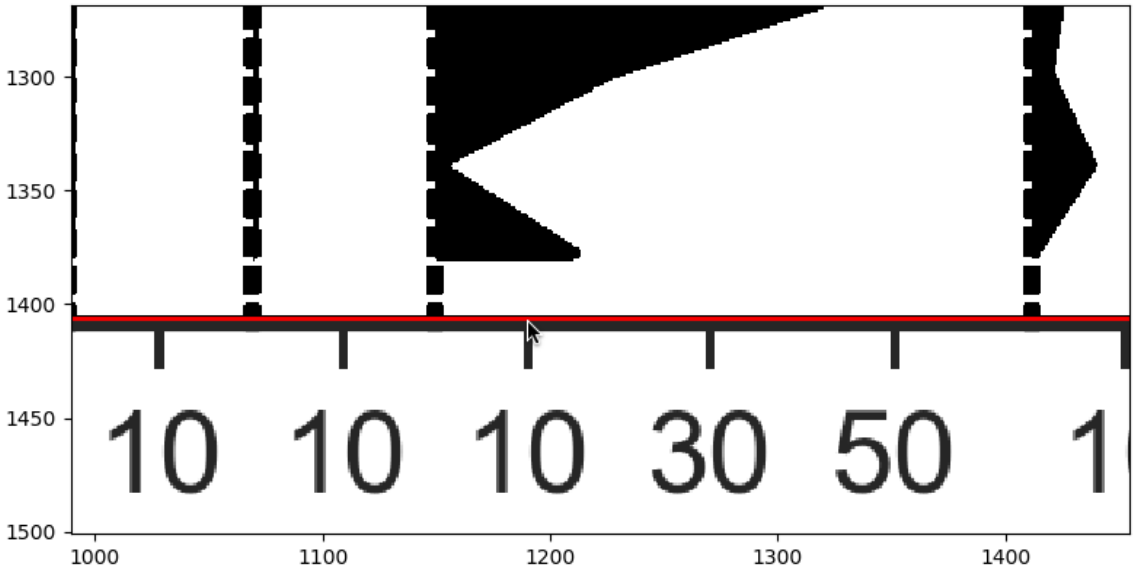
7. Click the *Apply* button at the bottom of the straditizer control when you are done.

Note: If you drag a mark and hold the `Shift` button while releasing the mouse button, the dialog in point 3 from above will not pop up.

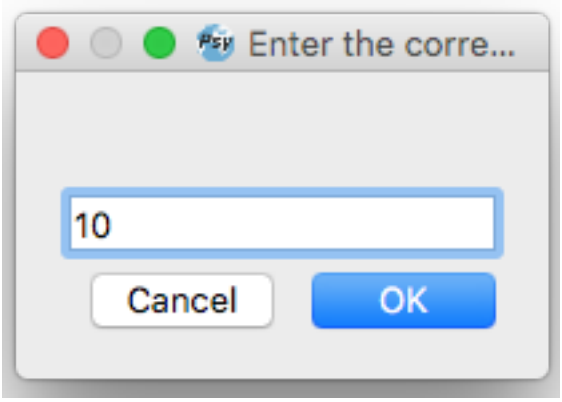
Translating the x-axis

As we did for the y-axes, we do it for the x-axes.

1. Click the *Insert X-axis values* button in the *Axes translations* section of the straditizer control
2. A small dialog will appear where you should enter the value at the column start (in our case, 0)
3. Shift-leftclick on the plot to enter another x-value.



4. A small dialog will appear where you should enter the x-value to use



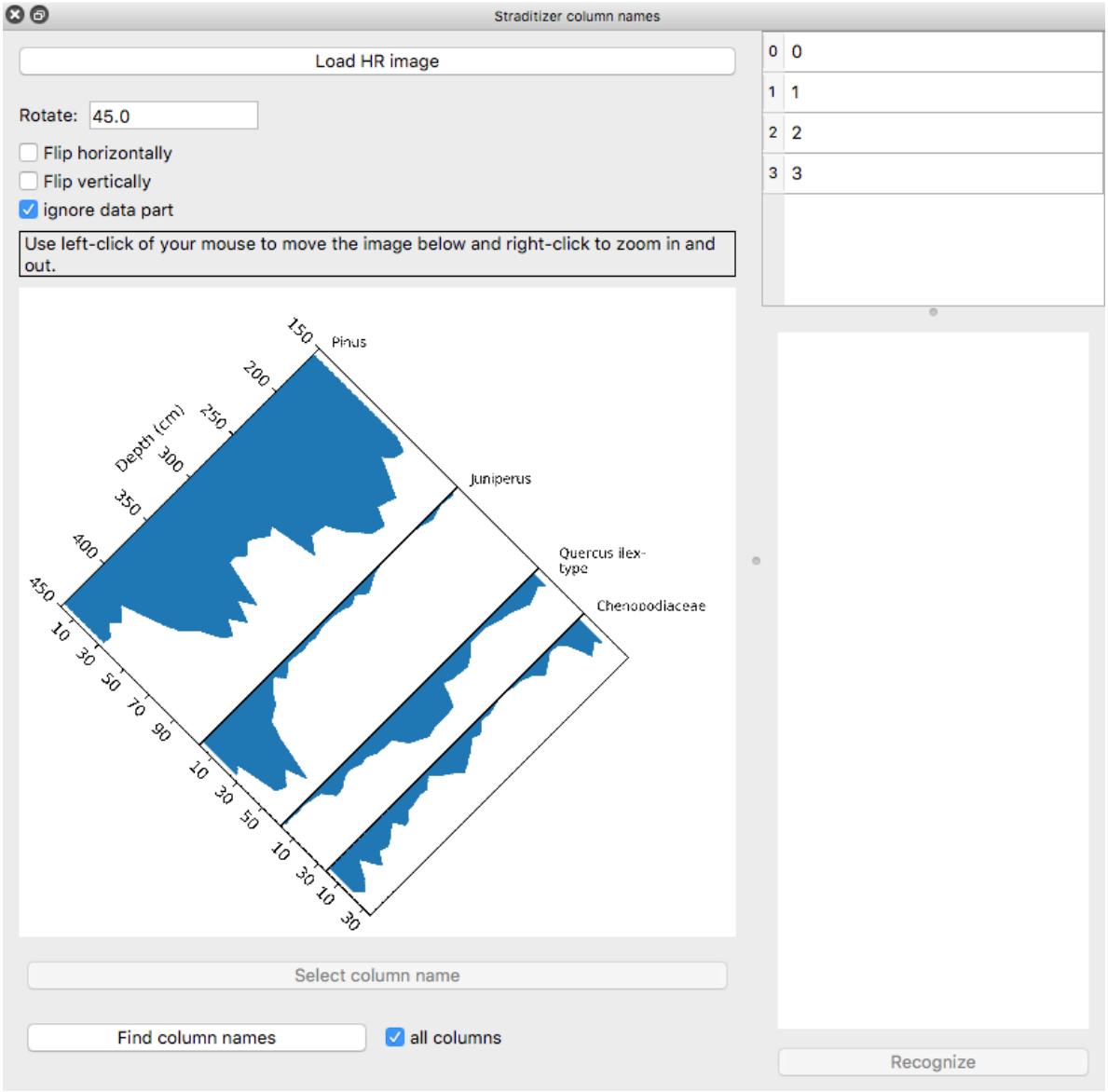
5. Click the *Apply* button at the bottom of the straditizer control when you are done.

Column names

Finally, we select the column titles and save them in the project.

Each of the sub diagrams has a title, in our case, *Pinus*, *Juniperus*, *Quercus ilex-type* and *Chenopodiaceae*. These names should be used in the final digitization result and therefore you can include them in straditize:

1. Expand the *Column names* item in the straditizer control and click the *Edit column names* button
2. In the appearing widget, the colnames editor

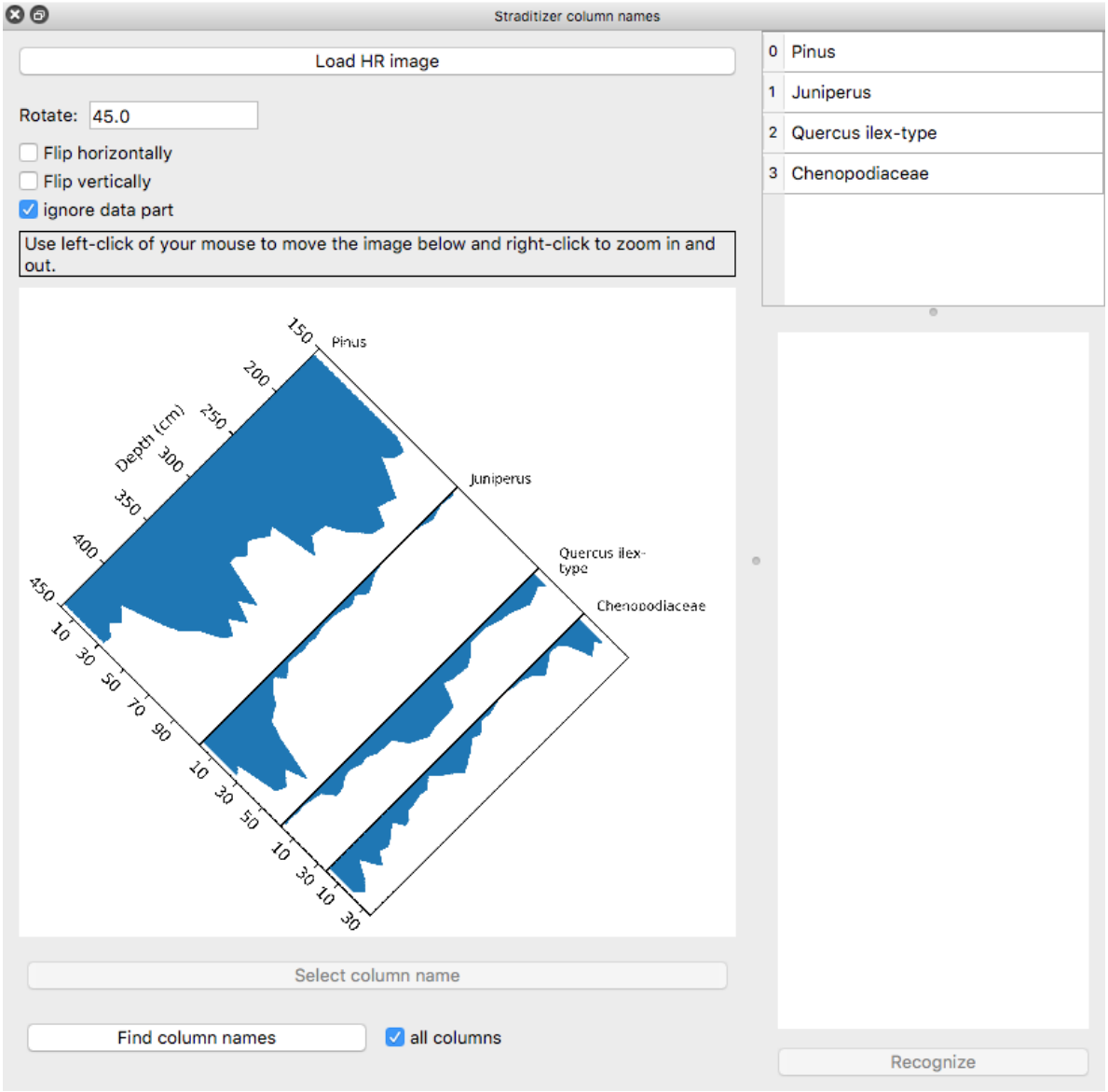


you find a table where you can edit the column names. The plot on it's left also shows a rotated version of the diagram, to help you identifying the column names. You can navigate in this plot using leftclick and zoom in and out using right-click (see matplotlibs docs on interactive navigation with the Pan/Zoom-tool).

Hint: If you have tesserocr installed (see the *user manual*), you can also just click the *Find column names* button and you are done.

---

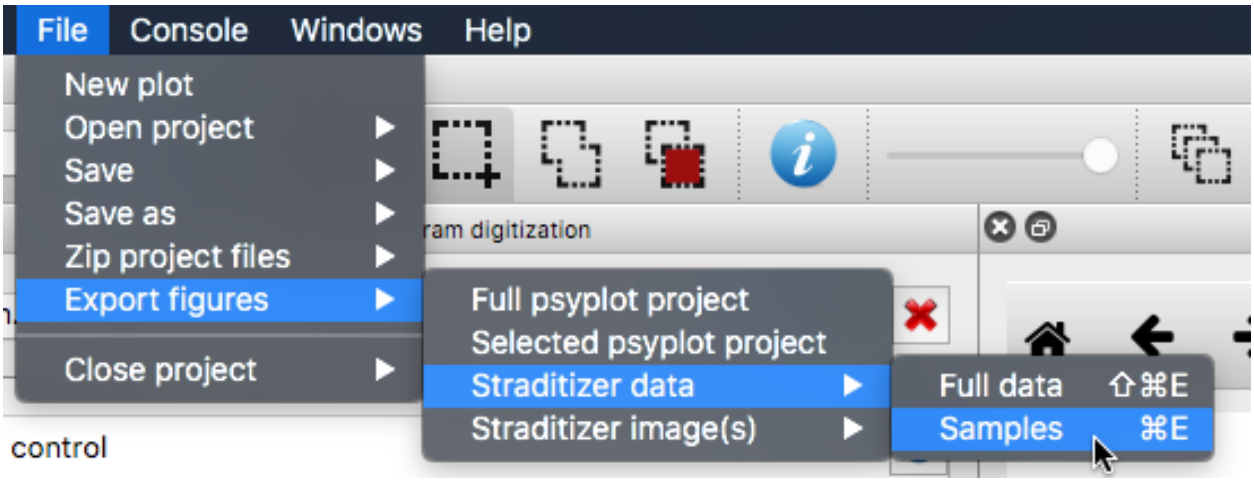
3. When you entered the correct names in the table (see image below), click the *Edit column names* button again to hide the dialog.



Done!

That's it! You can export the samples (see *Export your results*) to an Excel or CSV file via

File → *Export figures* → *Straditizer data* → *Samples*



Now start with your own diagram!

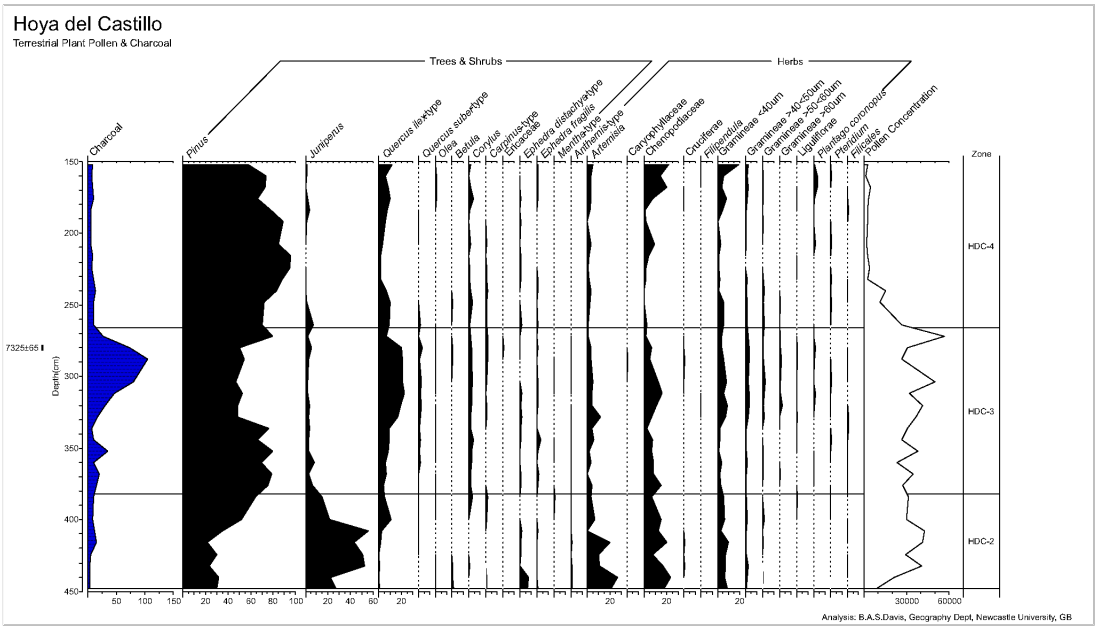
If you have questions or troubles, please open an issue on

https://github.com/Chilipp/straditize/issues

Thanks for using **straditize** and happy digitizing!

References

Advanced tutorial: The Hoya del Castillo pollen diagram



In this tutorial, we digitize a stratigraphic diagram based on the data in [Basil2007] and introduce to you how to extract the samples out of an area diagram.

At the bottom of the straditizer control, you find a navigation panel which helps you to navigate through the tutorial. Click the *Next* button in this control to start the tutorial. You can skip steps by clicking the *Skip* button and you can check what you have done using the *Check* button and then proceed with the next step.

The tutorial will take between 15 and 30 minutes. You can save your current state to a file using *File* → *Save* → *Save straditizer* (see *Saving and loading your project*) and reload it later via *File* → *Open project* → *Open straditizer* → *Project or image*

If you have questions or troubles with this tutorial, please open an issue on

https://github.com/Chilipp/straditize/issues

and we will do our best to assist you.

Load the image of the diagram

The first step is to load the image of the diagram into straditize.

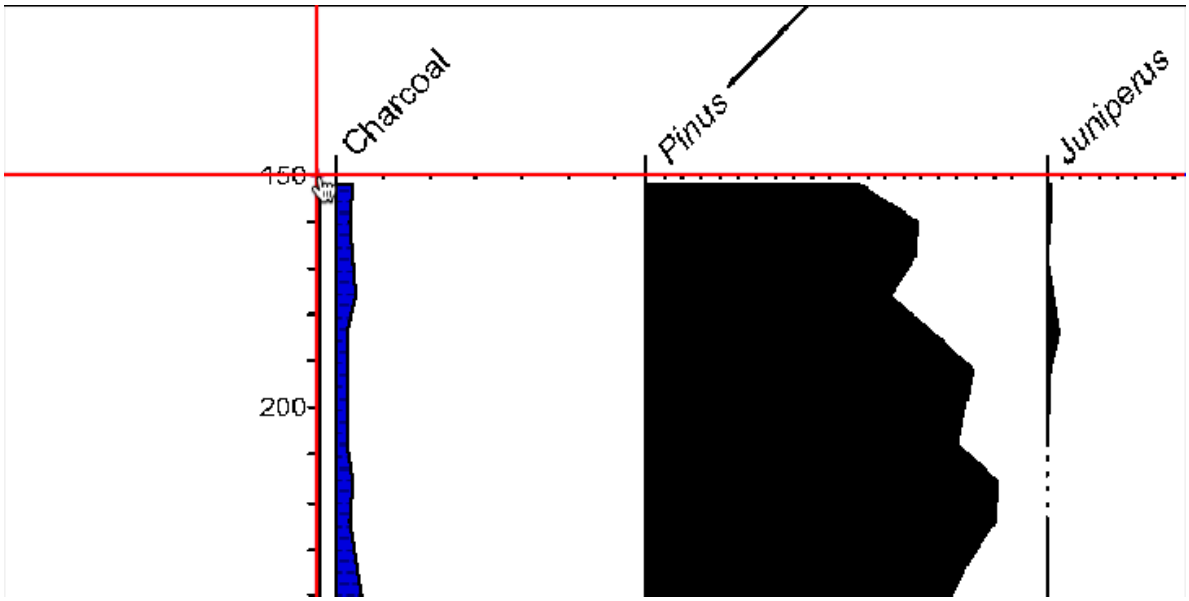
1. Click the  button at the top of the straditizer control.
2. Select the image file `hoya-del-castillo.png` from your hard-disk and click Open

Select the diagram part

The second step involves selecting the diagram part. This is the part where your data is displayed, but without the vertical and horizontal axes or axes descriptions.

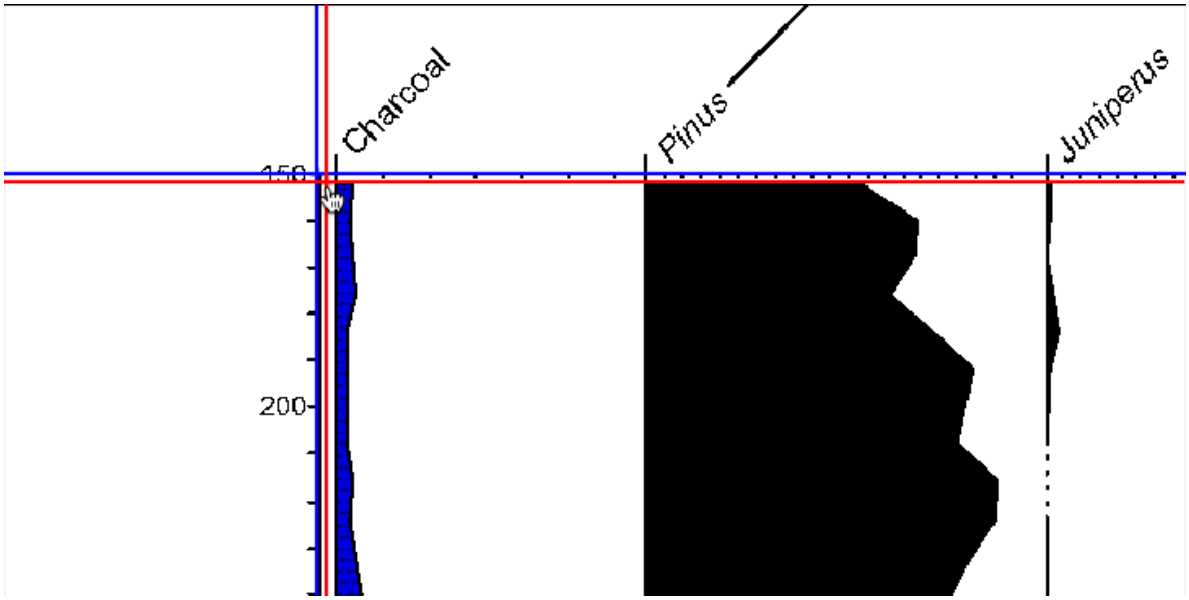
Being exact in this step significantly reduces the later work and simplifies the automatic digitization.

1. Click the *Select data part* button in the digitization control.
2. Click on the upper left cross to select it

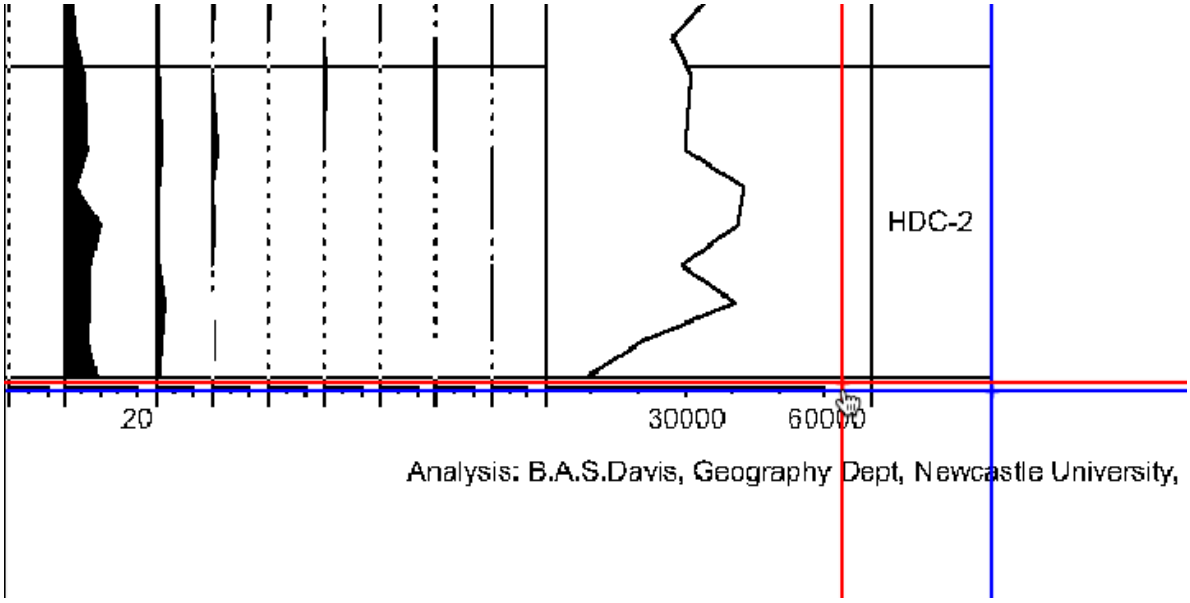


3. Hold the left mouse button and drag the cross to the upper left corner of the diagram. Make sure you don't include the y-axis and x-axis ticks.

   You can also use the zoom and navigation tools in the figure toolbar for navigation.



5. Now select the cross at the lower right corner of the diagram and move it such that you don't include the x-axis ticks (i.e. the numbers on the x-axis)



6. Click the *Apply* button at the bottom

If you want to change the appearance of the marks, see the *Marker control* section in the straditizer control panel.

Create the diagram reader

Now you have to select, which type of diagram you are digitizing. In the example, it is an area diagram which is already selected in the dropdown menu next to the *Select data part* button.

Click on *Convert image* to start the digitization.

See also:

Selecting the reader

Identifying the columns

The next step is, to automatically separate the columns from each other. This is simple in our case, since straditize can separate them automatically.

1. Click the *Column starts* button. You will see several vertical blue lines appearing on the plot that you can drag and drop such as you did it when selecting the diagram part. You can change their colors using the *marker control*
2. straditize recognizes these columns automatically and in our example, you do not have to edit them. Therefore hit the *Apply* button at the bottom of the control and you are done.

See also:

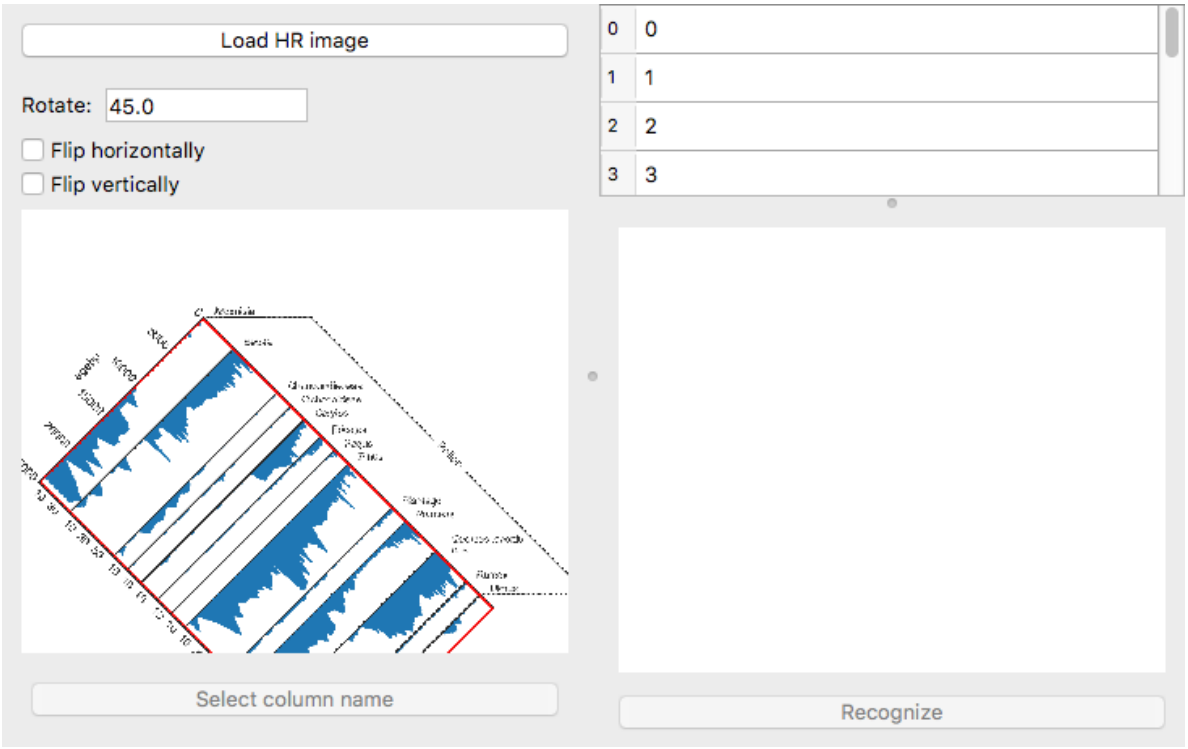
Select the starts for each column

Specifying the column names

Straditize can handle column names which will then be included in the final export. It interfaces with the tesserocr package and we will use this to minimize our typing amount. However, we have a complex diagram so manual corrections are unavoidable.

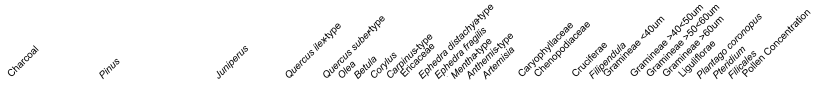
1. Expand the *Column names* item in the straditizer control and click the *Edit column names* button.

   In the appearing widget, the colnames editor



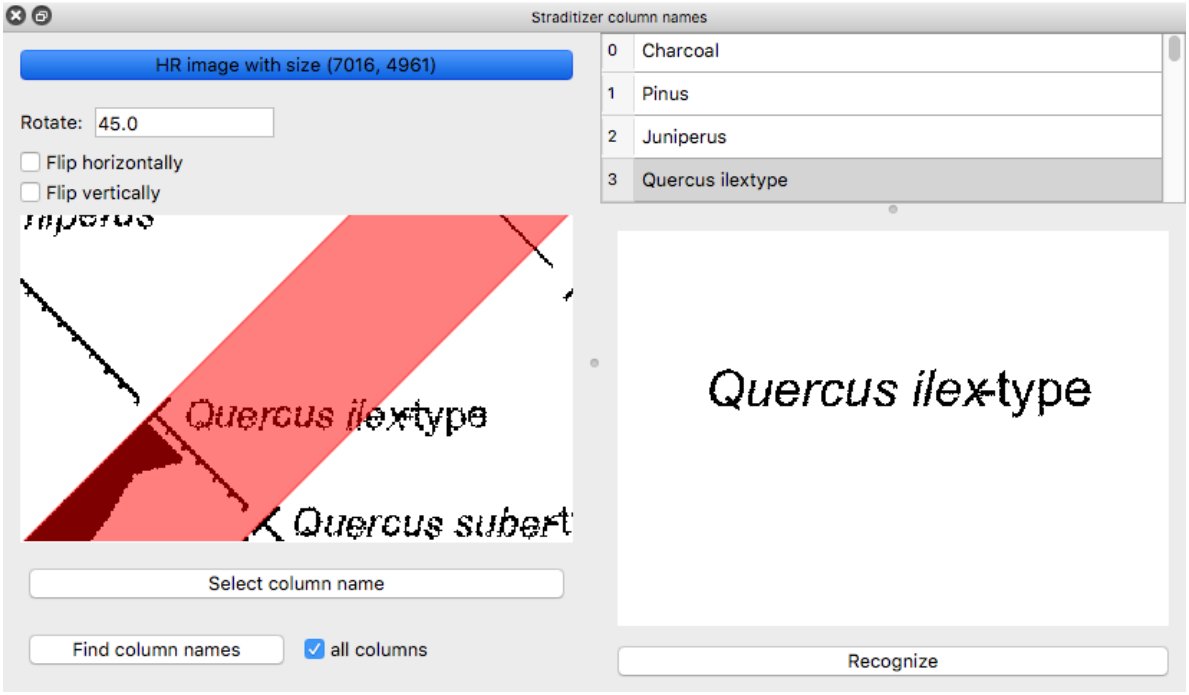
   you find a table where you can edit the column names. The plot on it's left also shows a rotated version of the diagram, to help you identifying the column names. You can navigate in this plot using leftclick and zoom in and out using right-click (see matplotlibs docs on interactive navigation with the Pan/Zoom-tool)

2. To improve the text recognition, it is highly recommended to have a clean image with only the column names on it and a sufficient resolution. We have something prepared for you:



Click the *Load HR image* button and select the *hoya-del-castillo-colnames.png* image file.

3. We will now use the automatic finding of column names. For this, click the *Find column names* button. This will search for the column names in the plot to the left of the table
4. There will probably be some errors in the column names. Therefore, go through each row in the table and check the name. You can use the *Hint* button to help you and/or zoom to the column name to see the name in the original



5. When you entered the correct names in the table, click the *Edit column names* button again to hide the button.

Remove artifacts

Stratigraphic diagrams, and especially pollen diagrams, often have a lot of artifacts in it, that are informative for the reader.

In our case, the diagram is splitted into three temporal *zones* (HDC-2, HDC-3 and HDC-4, see the *Zone* column on the right part of the diagram) which are visually separated with horizontal lines. Additionally, the diagram has vertical, dashed lines at each column start (the y-axes for each column).

Before we digitize our diagram, those *informative features* have to be removed. You can do this in an external image editing software (e.g. Photoshop) but we also implemented several automated algorithms to detect common features and remove them easily.

For our tutorial, we use the *Remove lines* feature to detect and remove the vertical and horizontal lines.

Horizontal lines

1. Expand the *Remove features* tab in the *Digitization control*
2. Expand the item with the *vertical lines* and *horizontal lines* button by clicking on the small arrow on their left
3. Set the minimum line width to 1 pixel
4. Click the *horizontal lines* button. In the plot you will see, that the horizontal lines are red now (if necessary, go with the mouse over the plot and you will see it in the zoom window). You could edit the selection now using the selection toolbar (see *The selection toolbar*), but for our tutorial, this is not necessary.
5. Click the *Remove* button to remove the lines

Vertical lines (y-axes)

1. Enable the maximum line width and set it to 2 pixel
2. Set the minimum fraction to 30%
3. Click the *vertical lines* button and the vertical lines turn red and are marked to be removed.
4. Click the *Remove* button to remove the y-axes

Additional automated removal tools are available and fully described in the documentation (see *Removing features*). But here, we can continue with the digitization.

Digitizing the data

The next step, after we cleaned up the image, is the digitization of the diagram. Click the *Digitize* button.

You now see the lines that result from the digitization. You can remove them in the *Plot control* section by clicking the  button of the *Full digitized data* row.

Finding and editing samples

So far, we have one data point per pixel in the image. However, we have to identify the locations of the samples in order to reproduce the original data.

Straditize assists you with this through automatic sample finding algorithms, or you can load the sample locations from an external files or you just add and edit the samples manually.

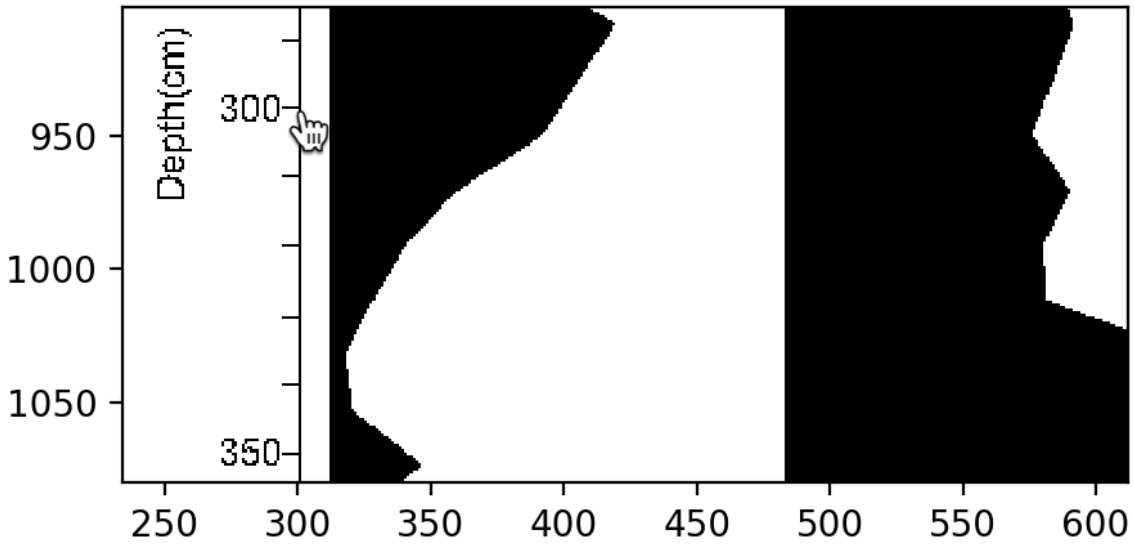
1. Expand the *Samples* item in the digitization control
2. Find the samples by clicking the *Find samples* button. straditize now identified the sample locations based on the extrema in the columns (see *Automatic samples identification* for an explanation of the algorithms).
3. To visualize the samples, you can again use the items in the *Plot control* tab. However, you can (and should!) also edit them by clicking the *Edit samples* button.
4. Now you see one horizontal line per sample that you can drag around (left-click), delete (right-click) or you can also add new samples (Shift + left-click). See the *Editing samples* section for more details.
5. Finally, click the *Apply* button or the *Cancel* button to stop the editing of the samples.

You're almost done! Your diagram is now digitized and the data could already be exported.

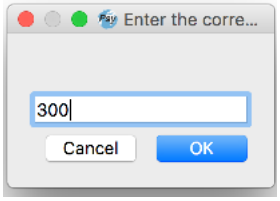
Translating the y-axis

To correctly reproduce the diagram data, there are now only to more things to know, that is the scaling of the y- and x-axes in the diagram.

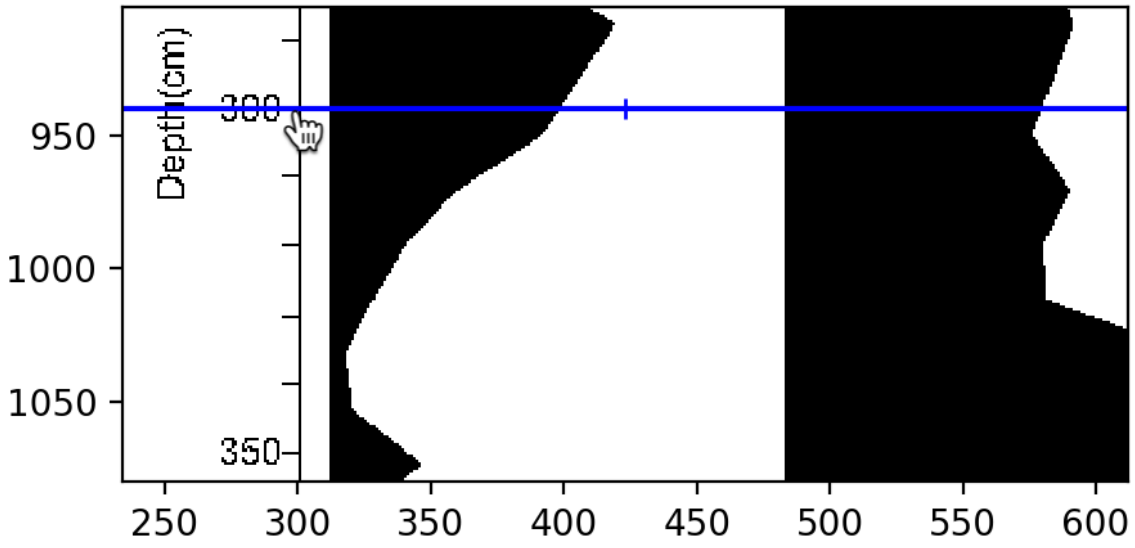
1. Expand the *Axes translations* tab in the digitization control
2. Click the *Insert Y-axis values* button in the *Axes translations* section of the straditizer control
3. Shift-leftclick on the plot to enter the corresponding y-value.



4. A small dialog will appear where you should enter the y-value to use (in this case, `300`)

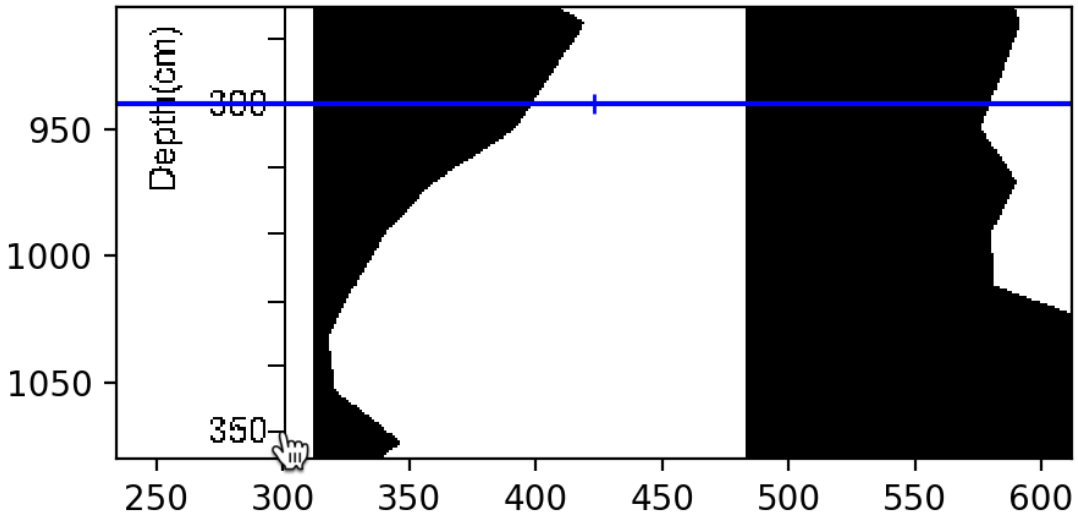


5. After hitting the *Ok* button, you will see a mark on the plot (blue line). You can select the mark via leftclick and drag it to a different location or you can delete it via rightclick.

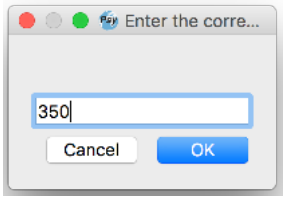


6. now repeat steps 2-4 on a second point on the y-axis

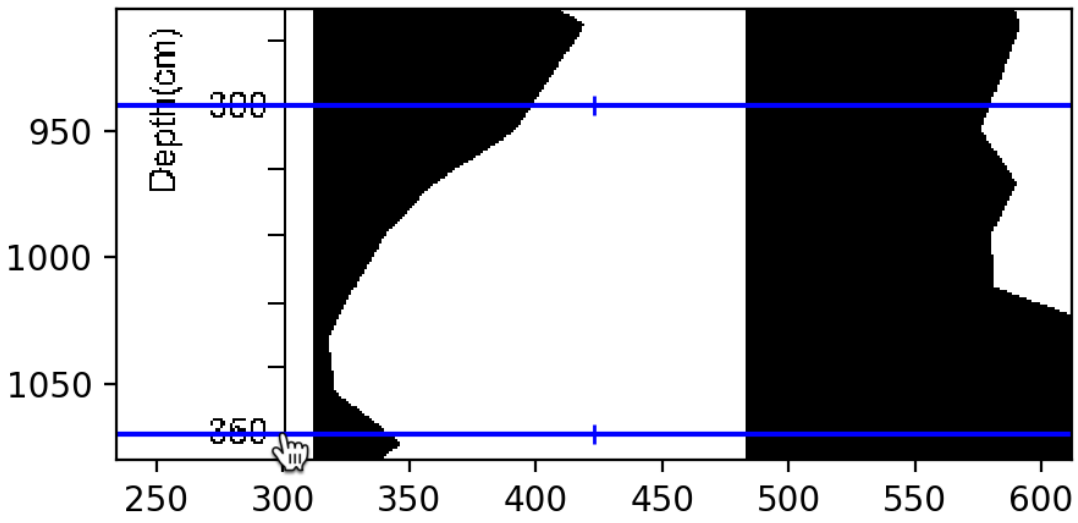
- Select another point



- Enter the corresponding value (here `350`)



- A new mark is created that you can modify



7. Click the *Apply* button at the bottom of the straditizer control when you are done.

Note: If you drag a mark and hold the `Shift` button while releasing the mouse button, the dialog in point 3 from above will not pop up.

Translating the x-axes

Additionally to the vertical axis, we have to tell straditize how to interprete the x-axes of the stratigraphic diagram.

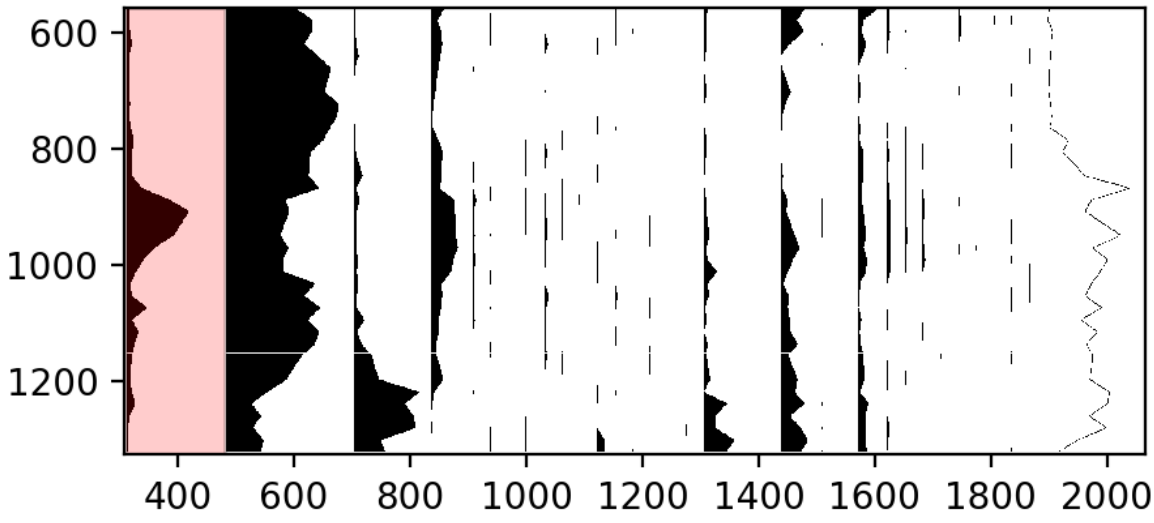
Here we have to deal different units of the x-axes in the diagrams. The first (*Charcoal*) and last (*Pollen Concentration*) columns are counts, the pollen taxa in between are in percent.

Therefore we will create column specific readers, one for the first and one for the last column. Then we will insert the x-axis data.

The steps down below are repetitive, but we will describe them for each step in detail.

The *Charcoal* column

1. Expand the *Current reader* tab in the digitization control
2. Click the + button to start the selection for a new reader
3. Click in the plot on the first column. It will turn red



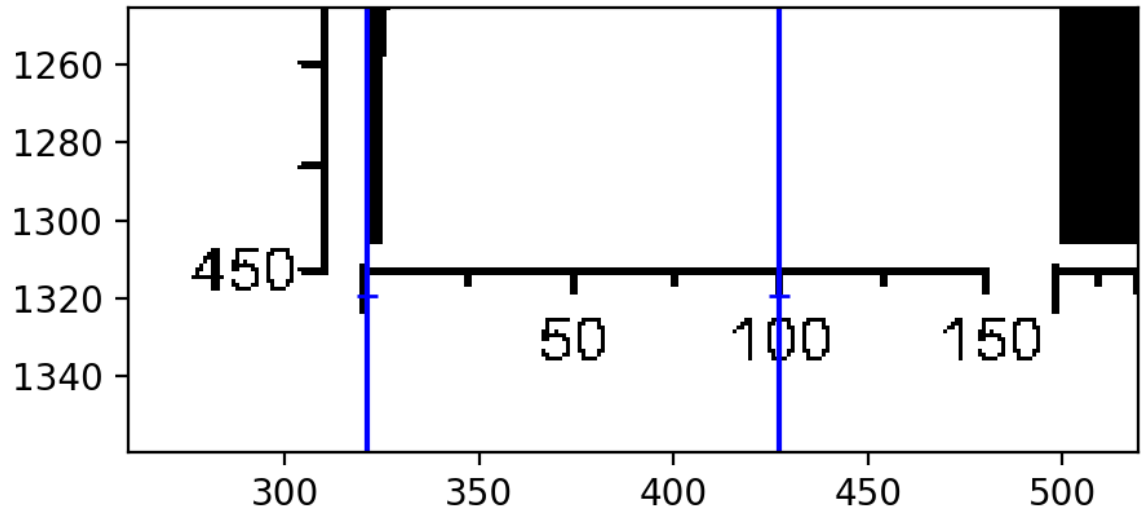
4. Click the *Apply* button and select the *area* reader in the appearing dialog
5. In the dropdown menu in the *Current reader* tab, select the reader for *Column 0* to make it the current reader.

Now, we select the values for interpreting the x-axis of this column. The procedure is more or less the same, as before with the y-axis:

1. Click the *Insert X-axis values* button in the *Axes translations* section of the straditizer control
2. A small dialog will appear where you should enter the x-value of the start of the column (here `0`).

Note: The dialog only appears, if the start of one of the selected columns is visible. If this is not the case, Shift-Leftclick on the plot to create a mark.

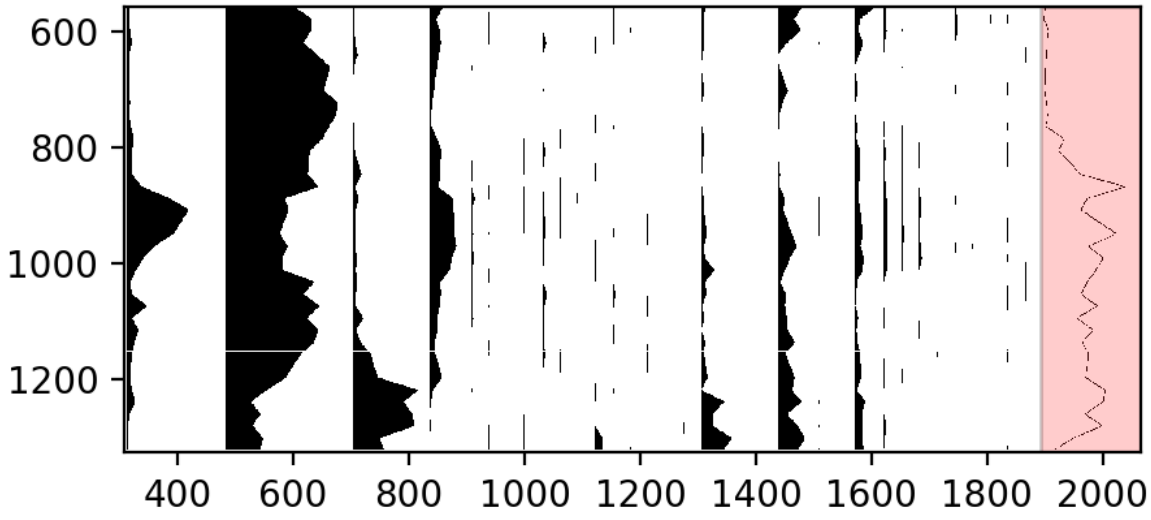
3. After hitting the *Ok* button, you will see a mark on the plot (blue line). You can select the mark via leftclick and drag it to a different location or you can delete it via rightclick.
4. now repeat steps 2-4 on a second point in the same column. Your diagram should now look something like this:



5. Click the *Apply* button at the bottom of the straditizer control and we can continue with the last column

The *Pollen Concentration* column

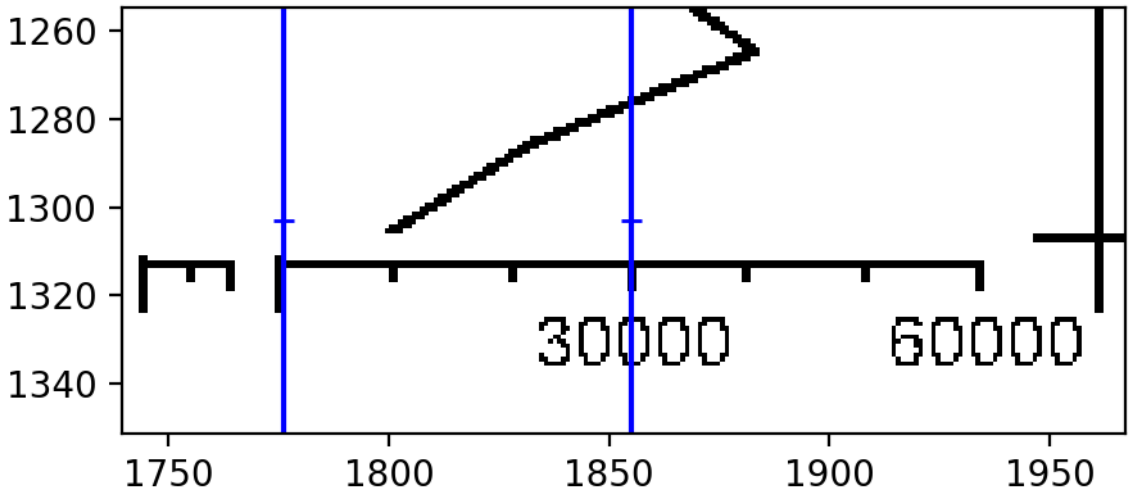
1. In the *Current reader* dropdown menu, select the reader for the columns 1-27
2. Click the + button to start the selection for a new reader
3. Click in the plot on the last column. It will turn red



4. Click the *Apply* button and select the *line* reader in the appearing dialog.
5. In the dropdown menu in the *Current reader* tab, select the reader for *Column 27* to make it the current reader.

Now, we select the values for interpreting the x-axis of this column. The procedure is the same as above:

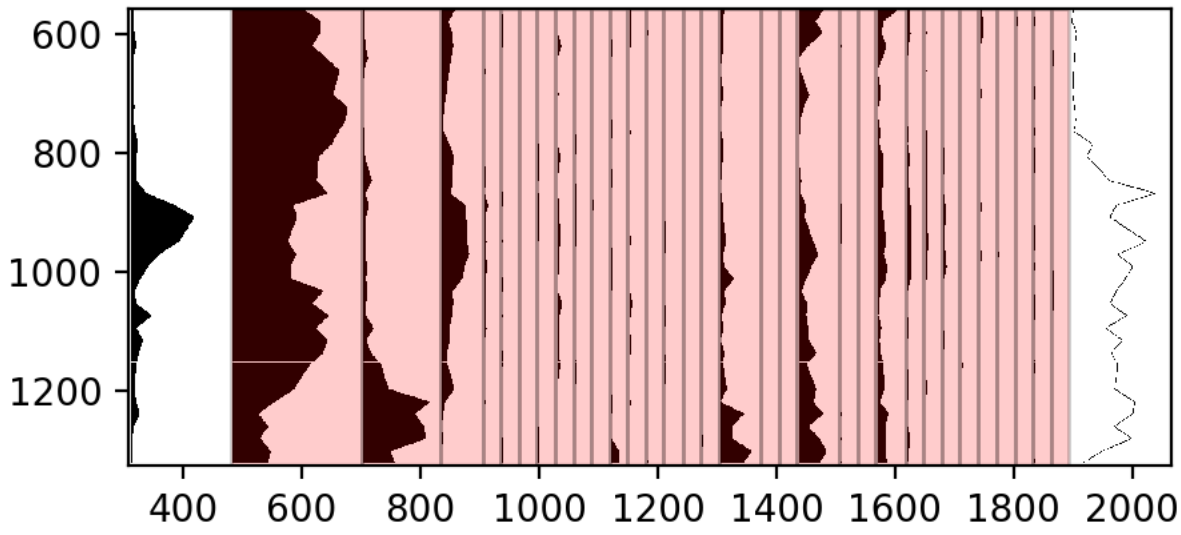
1. Click the *Insert X-axis values* button in the *Axes translations* section of the straditizer control
2. In the appearing dialog, enter the x-value of the column start (here again `0`) or Shift-Leftclick on the plot and then enter it (see note above).
3. After hitting the *Ok* button, you will see a mark on the plot (blue line). Again, you can select the mark, drag it or delete it as before.
4. now repeat steps 2-4 on a second point in the same column. Your diagram should now look something like this:



5. Click the *Apply* button at the bottom of the straditizer control and we can continue with the pollen taxa.

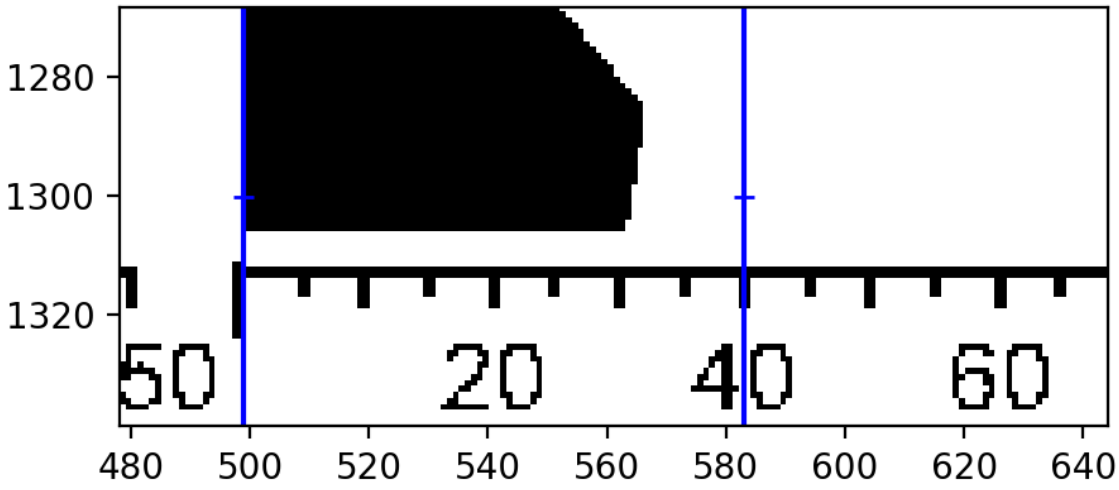
The pollen taxa columns

Last but not least, we translate the x-axes informations for the pollen taxa.



Luckily, as it is common for pollen diagrams, they all have the same scaling. Therefore it is enough to perform the above steps just for one of the columns.

1. In the dropdown menu in the *Current reader* tab, select the reader for *Columns 1-26* to make it the current reader.
2. Click the *Insert X-axis values* button in the *Axes translations* section of the straditizer control
3. In the appearing dialog, enter the x-value of the column start (here again `0`) or Shift-Leftclick on the plot and then enter it (see note above).
4. After hitting the *Ok* button, you will see a mark on the plot (blue line). Again, you can select the mark, drag it or delete it as before.
5. now repeat steps 2-4 on a second point in the same column. Your diagram should now look something like this:



6. Click the *Apply* button at the bottom of the straditizer control.

Editing the meta attributes

To keep the overview on your project, you can save some meta informations to it, for example, where you have the diagram from and what it represents.

Here for example, we know that the digitized data represents a pollen core from *Hoya del Castillo* and has been described in detail in [Basil1980].

1. Click the *Attributes* button at the bottom of the straditizer control. You know see a table with attributes, that you can rename, delete and fill according to your needs.
2. Fill the data table:

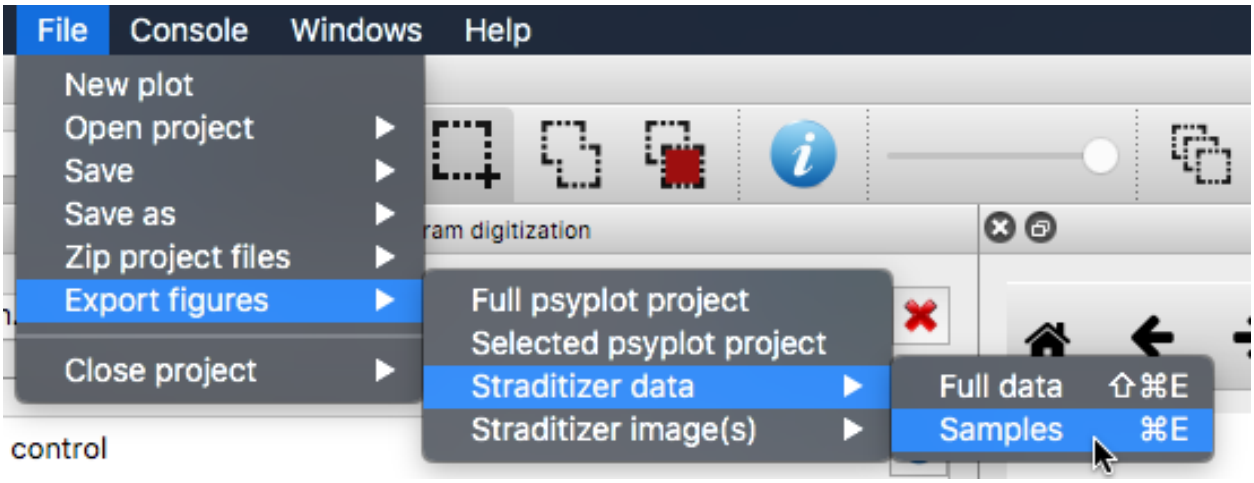
Digitized by	Your name
sitename	Hoya del Castillo
Lon	-0.5
Lat	41.25
Archive	Pollen
Country	Spain
Restricted	No
Reference	Davis, Basil A. S., and A. C. Stevenson. "The 8.2ka Event and Early-Mid Holocene Forests, Fires and Flooding in the Central Ebro Desert, NE Spain." Quat. Sci. Rev. , vol. 26, no. 13-14, 2007, pp. 1695-712
DOI	10.1016/j.quascirev.2007.04.007

Note: You can also add new attributes by clicking *Rightclick → Insert 1 row* inside the table

Done!

That's it! You can export the samples (see *Export your results*) to an Excel or CSV file via

File → Export figures → Straditizer data → Samples



Now start with your own diagram!

If you have questions or troubles, please open an issue on

https://github.com/Chilipp/straditize/issues

Thanks for using **straditize** and happy digitizing!

References

1.4 Contributing, asking for assistance and reporting bugs

First off, thanks for taking the time to contribute!

The following set of guidelines for contributing to straditize are mostly guidelines, not rules. Use your best judgment, and feel free to propose changes to this document in a pull request.

Table of Contents

1.4.1 Code of Conduct

This project and everyone participating in it is governed by the straditize Code of Conduct. By participating, you are expected to uphold this code.

1.4.2 How Can I Contribute?

Reporting Bugs

This section guides you through submitting a bug report for straditize. Following these guidelines helps maintainers and the community understand your report, reproduce the behavior, and find related reports.

Before creating bug reports, please check existing issues and pull requests as you might find out that you don't need to create one. When you are creating a bug report, please *include as many details as possible*. Fill out the required template, the information it asks for helps us resolve issues faster.

> **Note:** If you find a **Closed** issue that seems like it is the same thing that you're experiencing, open a new issue and include a link to the original issue in the body of your new one.

How Do I Submit A (Good) Bug Report?

Bugs are tracked as GitHub issues. Create an issue on straditize repository and provide the following information by filling in the template.

Explain the problem and include additional details to help maintainers reproduce the problem:

- **Use a clear and descriptive title** for the issue to identify the problem.
- **Describe the exact steps which reproduce the problem** in as many details as possible.
- **Provide specific examples to demonstrate the steps**. Include links to files or images, or copy/pasteable snippets, which you use in those examples. If you're providing snippets in the issue, use Markdown code blocks.
- **Upload the project file**. The best is, you you only take the part of the stratigraphic diagram that causes the problems.
- **Describe the behavior you observed after following the steps** and point out what exactly is the problem with that behavior.
- **Explain which behavior you expected to see instead and why.**
- **Include screenshots and animated GIFs** which show you following the described steps and clearly demonstrate the problem. You can use this tool to record GIFs on macOS and Windows, and this tool or this tool on Linux.
- **If the problem is related to your data structure**, include a small example how a similar data structure can be generated

Include details about your configuration and environment:

- **Which version of straditize and psyplot are you using?** You can get the exact version by running `straditize -V` and `psyplot -aV` in your terminal , or by starting the psyplot-gui and open Help->Dependencies.
- **What's the name and version of the OS you're using**?
- **Use conda's diagnostics!** If you installed straditize through anaconda, include the output of `conda info -a` and `conda list` in your description

Suggesting Enhancements

If you want to change an existing feature, use the change feature template, otherwise fill in the new feature template.

How Do I Submit A (Good) Enhancement Suggestion?

Enhancement suggestions are tracked as GitHub issues. Create an issue in the straditize repository and follow these steps:

- **Use a clear and descriptive title** for the issue to identify the suggestion.
- **Provide a step-by-step description of the suggested enhancement** in as many details as possible.
- **Provide specific examples to demonstrate the steps**. Include copy/pasteable snippets which you use in those examples, as Markdown code blocks.
- **Describe the current behavior** and **explain which behavior you expected to see instead** and why.
- **Include screenshots and animated GIFs** which help you demonstrate the steps or point out the part of psyplot which the suggestion is related to. You can use this tool to record GIFs on macOS and Windows, and this tool or this tool on Linux.

- **Explain why this enhancement would be useful** to most straditize users.
- **List some other analysis software or applications where this enhancement exists.**
- **Which version of straditize and psyplot are you using?** You can get the exact version by running `straditize -V` and `psyplot -aV` in your terminal , or by starting the psyplot-gui and open Help->Dependencies.
- **Specify the name and version of the OS you're using.**

Pull Requests

- Fill in the required template
- Do not include issue numbers in the PR title
- Include screenshots and animated GIFs in your pull request whenever possible.
- Document new code based on the *Documentation Styleguide*
- End all files with a newline and follow the PEP8, e.g. by using flake8

Adding new examples

You have new examples? Great! straditize can only be improved through new tutorials. You can either implement it in the graphical user interface (see the straditize.widgets.tutorial module) or you just add them to the sphinx documentation. We are also looking forward to assist you in the implementation and the sharing of your experiences.

1.4.3 Styleguides

Git Commit Messages

- Use the present tense ("Add feature" not "Added feature")
- Use the imperative mood ("Move cursor to…" not "Moves cursor to…")
- Limit the first line (summary) to 72 characters or less
- Reference issues and pull requests liberally after the first line
- When only changing documentation, include `[ci skip]` in the commit title

Documentation Styleguide

- Follow the numpy documentation guidelines.
- Use reStructuredText.
- Try to not repeat yourself and make use of the `straditize.common.docstrings`

Example

```
@docstrings.get_sectionsf('new_function')
def new_function(a=1):
    """Make some cool new feature

    This function implements a cool new feature

    Parameters
    ----------
    a: int
        First parameter

    Returns
    -------
    something awesome
        The result"""
    ...

@docstrings.dedent
def another_new_function(a=1, b=2):
    """Make another cool new feature

    Parameters
    ----------
    %(new_function.parameters)s
    b: int
        Another parameter

    Returns
    -------
    Something even more awesome"""
    ...

**Note:** This document has been inspired by `the contribution
guidelines of
Atom <https://github.com/atom/atom/blob/master/CONTRIBUTING.md#git-commit-messages>`__
```

1.5 Command line usage

The *straditize.__main__* module defines a simple parser to parse commands from the command line to load a diagram or a straditize project.

It can be run from the command line via:

```
python -m straditize [options] [arguments]
```

or simply:

```
straditize [options] [arguments]
```

Load a dataset, make the plot and save the result to a file

```
usage: straditize [-h] [-b [backend]] [-ni] [-rc-gui RC_GUI_FILE]
                  [-inc str [str ...]] [-exc str [str ...]] [--offline]
                  [-pwd str] [-s str] [-c str]
                  [-opengl {'software', 'desktop', 'gles', 'automatic'}]
```

(continues on next page)

(continued from previous page)

```
                [-o str] [-xlim val val] [-ylim val val] [-f]
                [-rt { 'area' | 'bars' | 'rounded bars' | 'stacked area' | 'line' }]
                [-V]
                [str]
```

1.5.1 Positional Arguments

str Either the path to a picture to digitize or a previously saved straditizer project (ending with `'.pkl'`)

1.5.2 Output options

Options that only have an effect if the *-o* option is set.

-o, --output The path to the csv file where to save the digitized diagram

1.5.3 Gui options

Options specific to the graphical user interface

-b, --backend The backend to use. By default, the `'gui.backend'` key in the `rcParams` dictionary is used. If used without options, the default matplotlib backend is used.

Default: False

-ni, --new-instance If True/set and the *output* parameter is not set, a new application is created

Default: False

-rc-gui, --rc-gui-file The path to a yaml configuration file that can be used to update the `rcParams`

-inc, --include-plugins The plugin widget to include. Can be either None to load all that are not explicitly excluded by *exclude_plugins* or a list of plugins to include. List items can be either module names, plugin names or the module name and widget via `'<module_name>:<widget>'`. Default: None

-exc, --exclude-plugins The plugin widgets to exclude. Can be either `'all'` to exclude all plugins or a list like in *include_plugins*.. Default: []

Default: []

--offline If True/set, psyplot will be started in offline mode without intersphinx and remote access for the help explorer

Default: False

-pwd The path to the working directory to use. Note if you do not provide any *fnames* or *project*, but set the *pwd*, it will switch the *pwd* of the current GUI.

-s, --script The path to a python script that shall be run in the GUI. If the GUI is already running, the commands will be executed in this GUI.

-c, --command Python commands that shall be run in the GUI. If the GUI is already running, the commands will be executed in this GUI

-opengl, --opengl-implementation Possible choices: software, desktop, gles, automatic

OpenGL implementation to pass to Qt. Possible options are 'software', 'desktop', 'gles' and 'automatic' (which let's PyQt decide).

1.5.4 Straditizer options

Options specific pollen diagram digitization

-xlim The x-limits of the data part of the diagram

-ylim The y-limits of the data part of the diagram

-f, --full If True, the image is digitzed and x- and ylim are set to the entire share of the array

Default: False

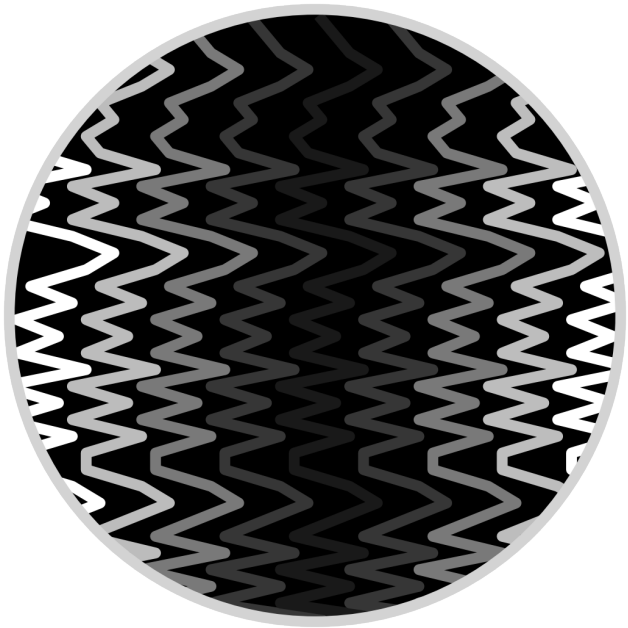
-rt, --reader-type Possible choices: area, bars, rounded bars, stacked area, line

Specify the reader type. Default: "area"

Default: "area"

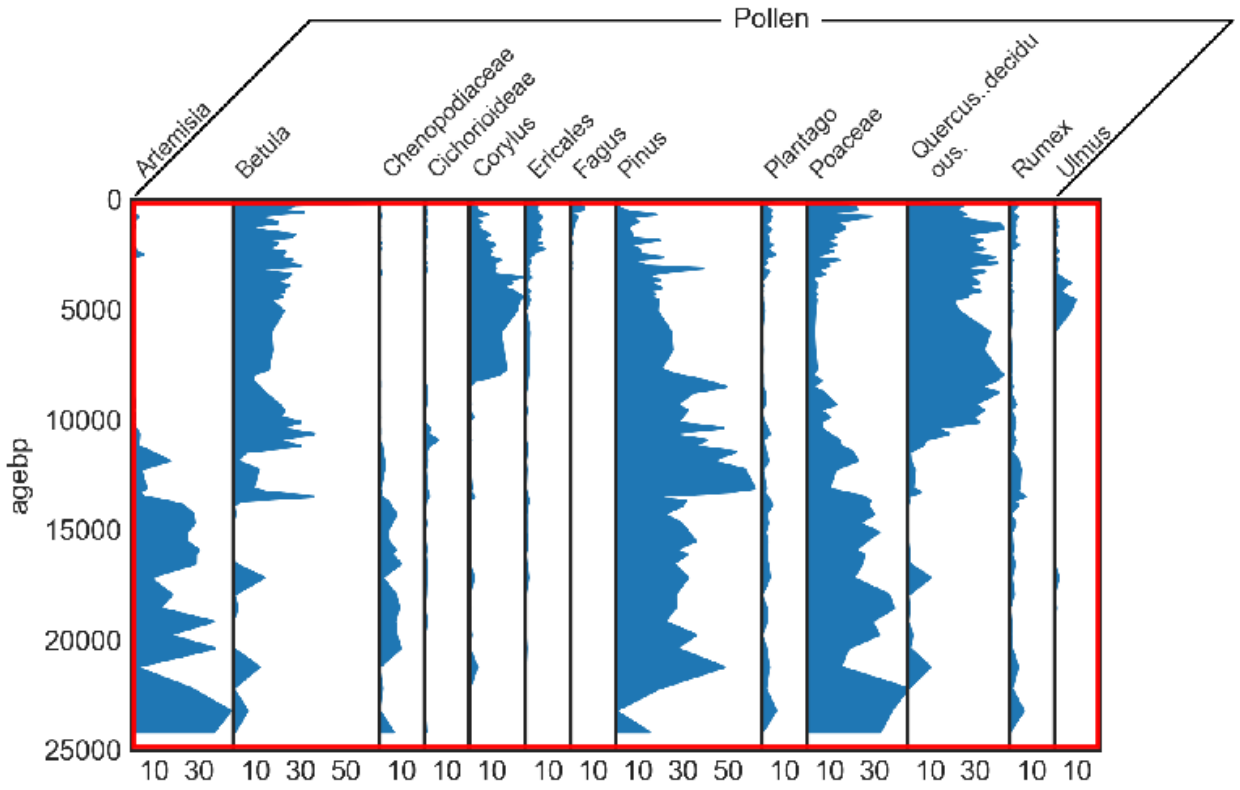
-V, --version show program's version number and exit

1.6 Straditize GUI



Welcome to the help explorer of the **Stra**tigraphic **Di**agram Digi**tize**r **straditize**. Here we guide you through the main functionality of the graphical user interface (GUI) and the necessary steps to digitze your stratigraphic diagram.

1.6.1 Basics and Terminology



A stratigraphic diagram consists of multiple diagrams that are vertically or horizontally aligned, i.e. that have either the same x- or y-axis.

This software assumes, that all the subdiagrams share the y-axis, i.e. the vertical axis. If this is not the case for your specific diagram, just rotate your image by 90°.

column One column in the diagram means one of the subdiagrams (e.g. *Betula* in the example diagram above). The total number of columns is the number of subdiagrams in your plot.

pixel column One pixel column is the horizontal pixel index (x-axis), ranging from 0 to the width of your image. Pixel column 0 represents all pixels on the very left side.

row or pixel row One pixel row (or sometimes only *row*) is the vertical pixel index (y-axis), ranging from 0 to the height of your image. Row 0 represents all pixels at the top.

diagram part This is the part where the data is shown (red rectangle in the figure above).

binary image The binary image is the diagram part converted to black (1) and white (0). Every black pixel in this binary image represents data that should be digitized.

digitize Digitization means transforming each pixel into a value (see the blue line in the image above). Therefore, in contrast to the samples, you get one data value for each pixel row in the binary image. We also refer to this as the *full data*.

sample The samples (red lines in the image above) are a subset of the digitization result. In a very basic sense, the samples represent the original data that was used to create the diagram. In a sediment core for a pollen diagram, it would be one slice of the core that was analyzed under the microscope.

straditizer Everything that refers to the straditizer refers to the full image of your stratigraphic diagram. Speaking in a programmatically sense, it is the top-level interface to digitize the diagram, an instance of the *straditize.straditizer.Straditizer* class. You can always access the current straditizer through the `stradi` variable in the ipython console of the GUI

reader or data reader This is the object that digitizes the diagram part (red area above). For different diagram types (area plot, bar plot, etc.) you have different readers with different functionalities.

The reader of your diagram can be accessed via typing `stradi.data_reader` in the ipython console of the GUI.

1.6.2 Straditization steps

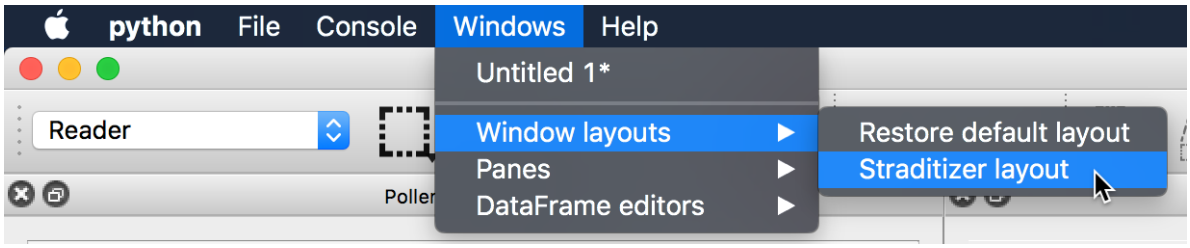
Straditize provides you lot's of possibilities for customizing the semi-automatic digitization of your diagram. However, there are certain steps that you should follow. You may not need all of them and you may exchange them to a certain degree as well.

Load a diagram

The first step is to load the picture of your diagram.

1. If you are starting straditize from the psyplot GUI (i.e. not via typing `straditize` in the terminal or through the app), you should first switch to the straditizer layout via

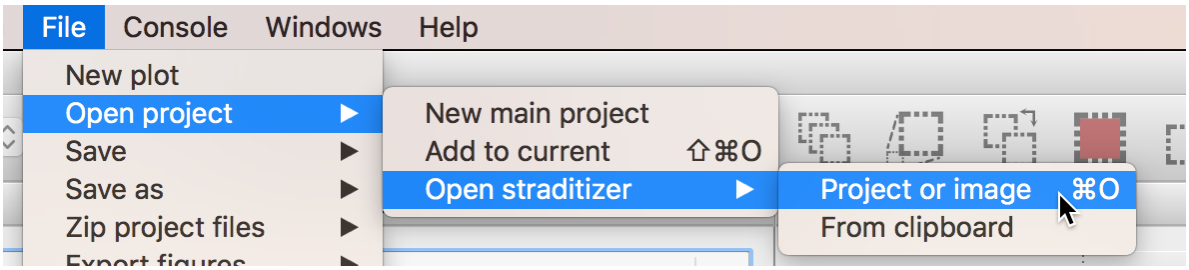
   Windows → Window layouts → Straditizer layout.

   This will show you the straditizer control panel.

2. To load the picture of your diagram, select

   File → Open project → Open straditizer → Project or image



and select the image file you want to load.

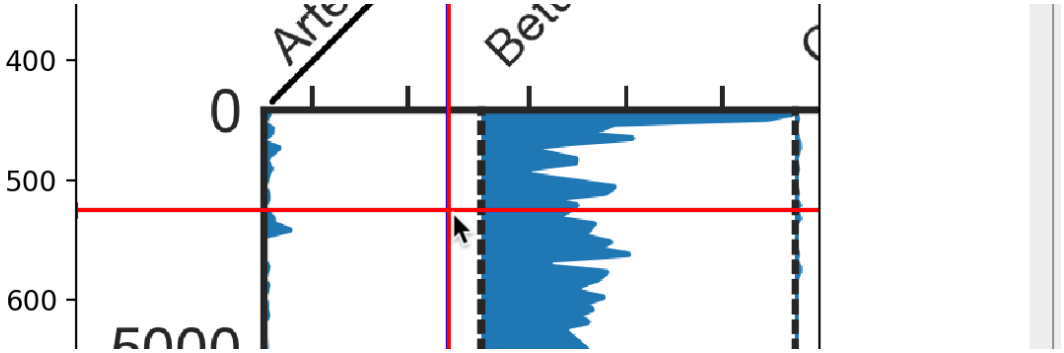
Alternatively, you can use the `Control-O` (`Command-O` on MacOS) keyboard shortcut or the button at the top of the straditizer control.

Select the diagram part

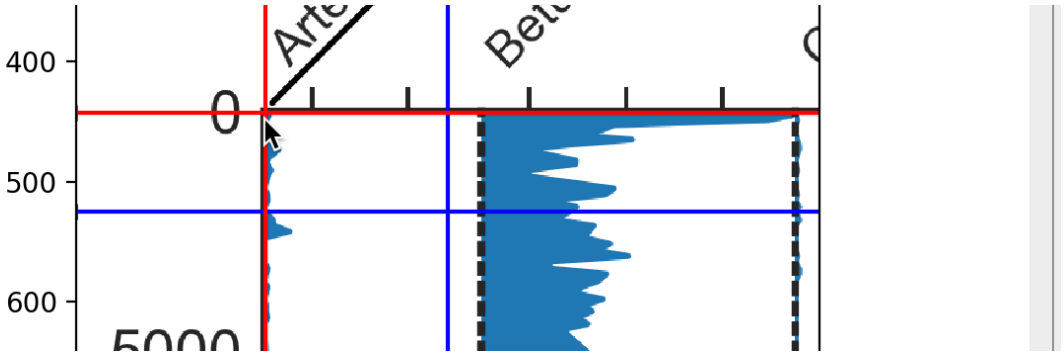
The second step involves selecting the diagram part (see the *Basics and Terminology*) by clicking the *Select data part* button in the digitization control.

The goal is to select the two outer corners of the diagram and try to avoid vertical and horizontal axes or axes description.

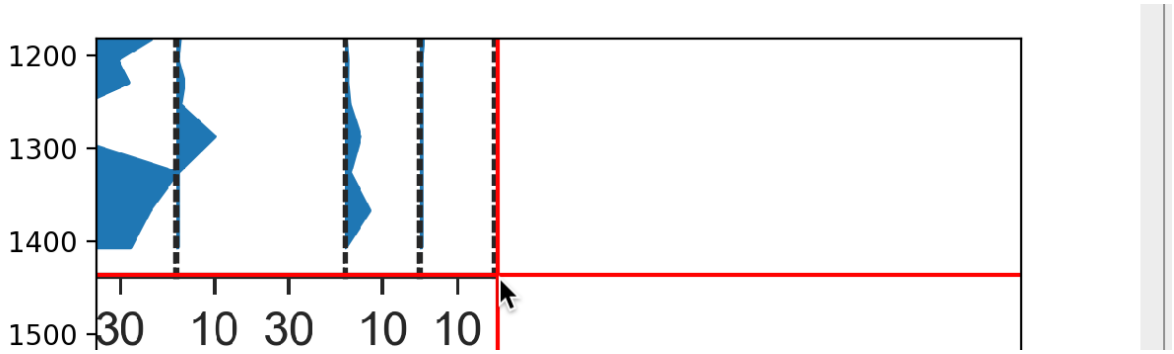
1. `Shift` - leftclick on the data diagram. This will create a mark at your current mouse location.
2. Click on the cross to select it



3. Hold the left mouse button and drag the cross to the upper left corner of the diagram. Make sure you don't include the y-axis ticks (i.e. the numbers on the y-axis)



4. Create another mark by another `Shift` - leftclick
5. Move the cross to the lower right corner of the diagram. Make sure you don't include the x-axis ticks (i.e. the numbers on the x-axis)



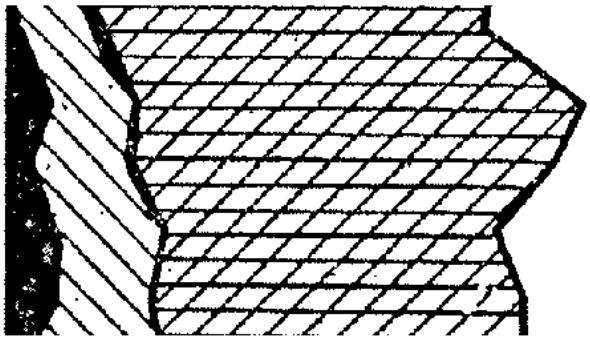
6. Click the *Apply* button at the bottom

If you want to change the appearance of the marks, see the *Marker control* section in the straditizer control panel.

Selecting the reader

Stratigraphic diagrams and especially pollen diagrams come in different styles and formats.

Straditize supports the following:

area	bars	rounded bars
		
line	**stacked area**	
		

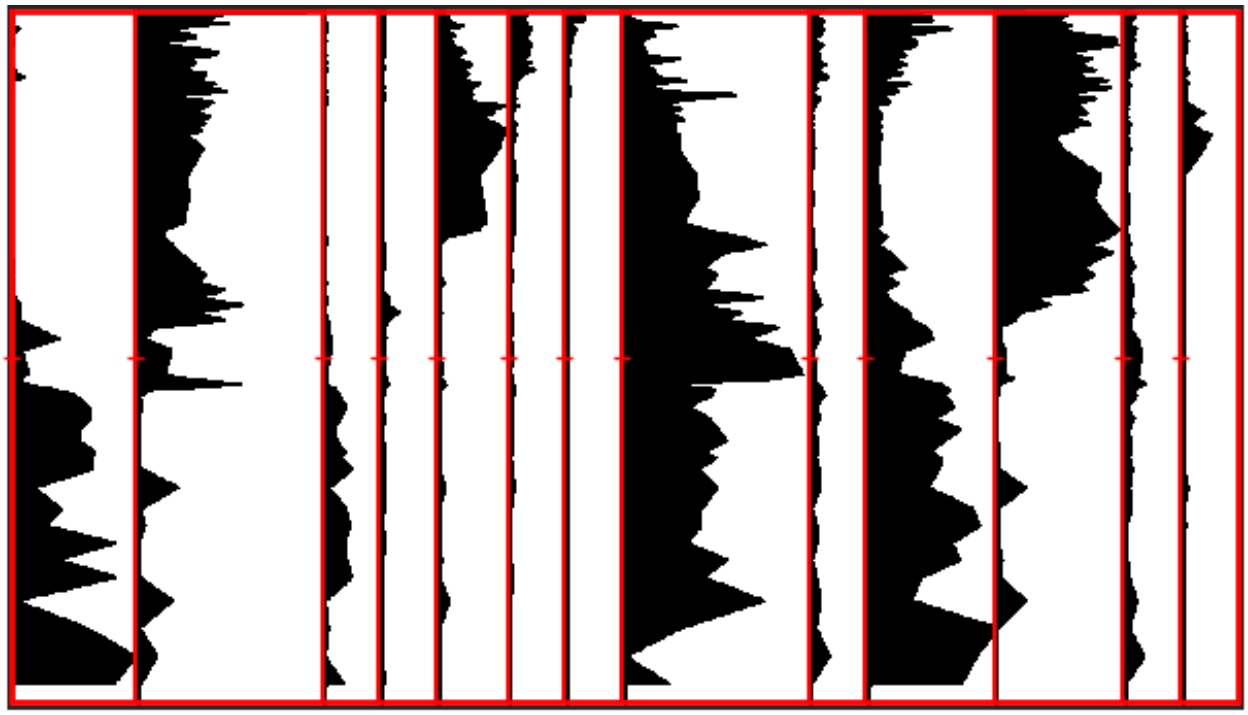
For each type there exists a specific reader that you can select from the drop down menu in the *Reader initialization* section of the straditizer control. If you're diagram contains a mixture of types, just choose one type for now and you can then later *create column specific readers*.

After you chose the *diagram part* and selected the reader type (see the table above), you can now click the *Convert image* button. This initializes the reader and converts the data image to a binary image, i.e. black and white.

Everything that is black in this image will be considered as data by the reader. So the next steps will include, cleaning up the image and removing unimportant parts.

Select the starts for each column

A stratigraphic diagram is defined through different columns, each column represents one variable (or for pollen diagrams, one taxa, species, PFT, etc.). To digitize the data, we have to know, where each of these column starts exactly.



After having clicked the *Column starts* button, the software automatically estimates the starts of the columns. However, you should check whether they are all recognized correctly.

There are several possibilities to edit the automatically estimated columns:

1. Hold the `Shift` button and left-click on the plot to create a new vertical mark.

   Then Left-click the mark and drag it to the start of one column.

2. Delete a column start by right-clicking the mark.

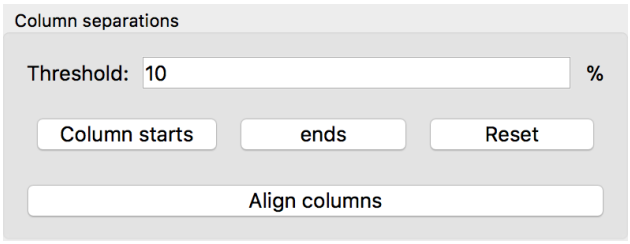
3. Modify the *threshold*. It defines the percentage of a pixel column that has to be covered to assume a valid column start. If you have columns that only contain a very small amount of data, you should lower it

See also:

Select the ends for each column

A pollen diagram is defined through different columns, each column represents one taxa, species, PFT, etc.. To digitize the data, we have to know, where each column starts and ends exactly.

After having clicked the *Column starts* button, the software automatically estimates the starts of the columns and, based on that, the column ends as well. Clicking the *ends* button however lets you modify the automatically estimated column ends.



As for the *column starts*, there are two ways to modify the column ends:

1. Hold the `Shift` button and left-click on the plot to create a new vertical mark.

   Then Left-click the mark and drag it to the start of one column.

2. Delete a column start by right-clicking the mark.

See also:

- *Align the columns vertically*
- *Select the starts for each column*

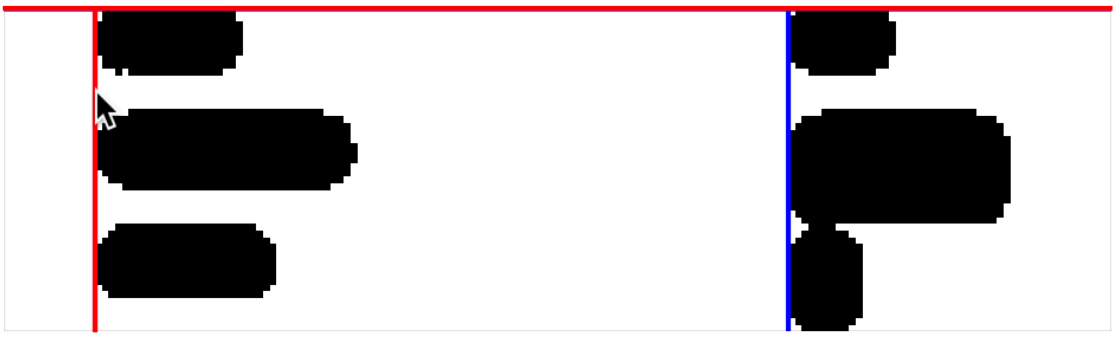
Align the columns vertically

Sometimes the columns are not aligned properly which makes it hard to identify where the exact sample is located and might lead to misinterpretations of samples. To correct this, we implemented a methodology to align the columns.

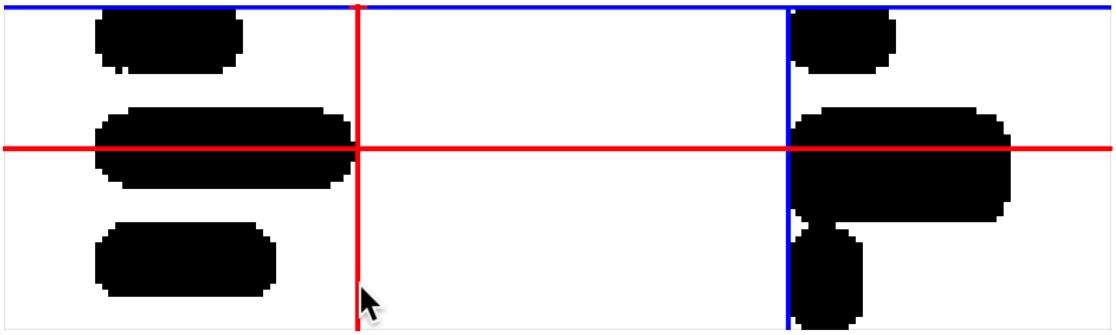
After having clicked the *Align columns* button, multiple marks are created. To correct the vertical displacement, move those marks to the vertical position in each column, that should be on the same vertical level.

You can select the horizontal lines for moving multiple marks and the vertical line for selecting only one mark.

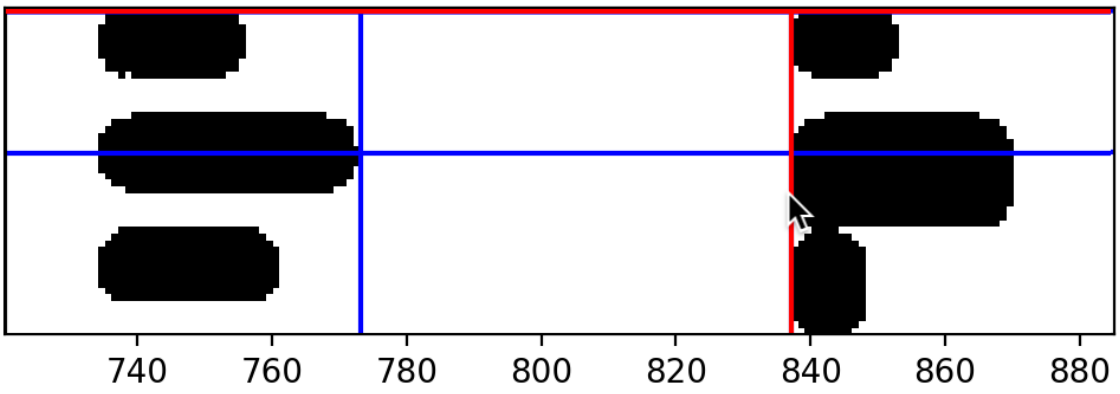
1. Select one mark on one column by clicking on it's vertical line



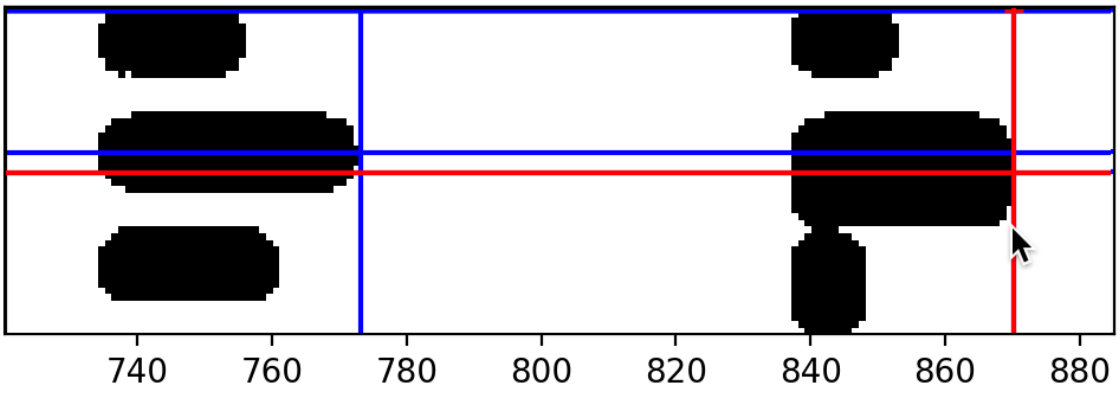
2. Move the mark vertically (the horizontal line does not matter)



3. Select another mark of another column



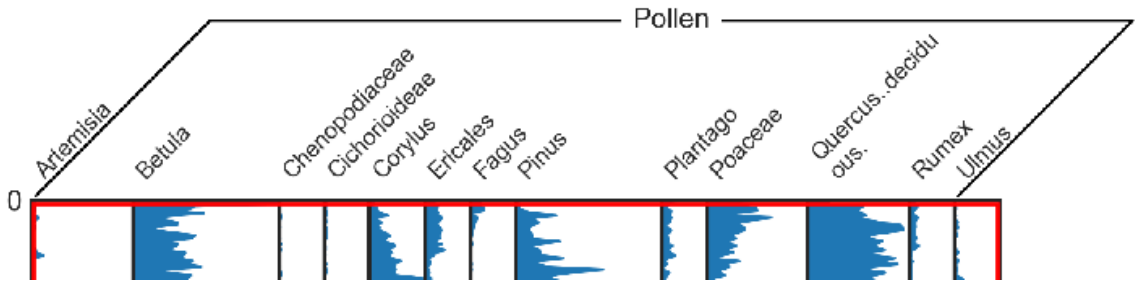
4. Move this mark to a point that should have the same y-value as the first mark



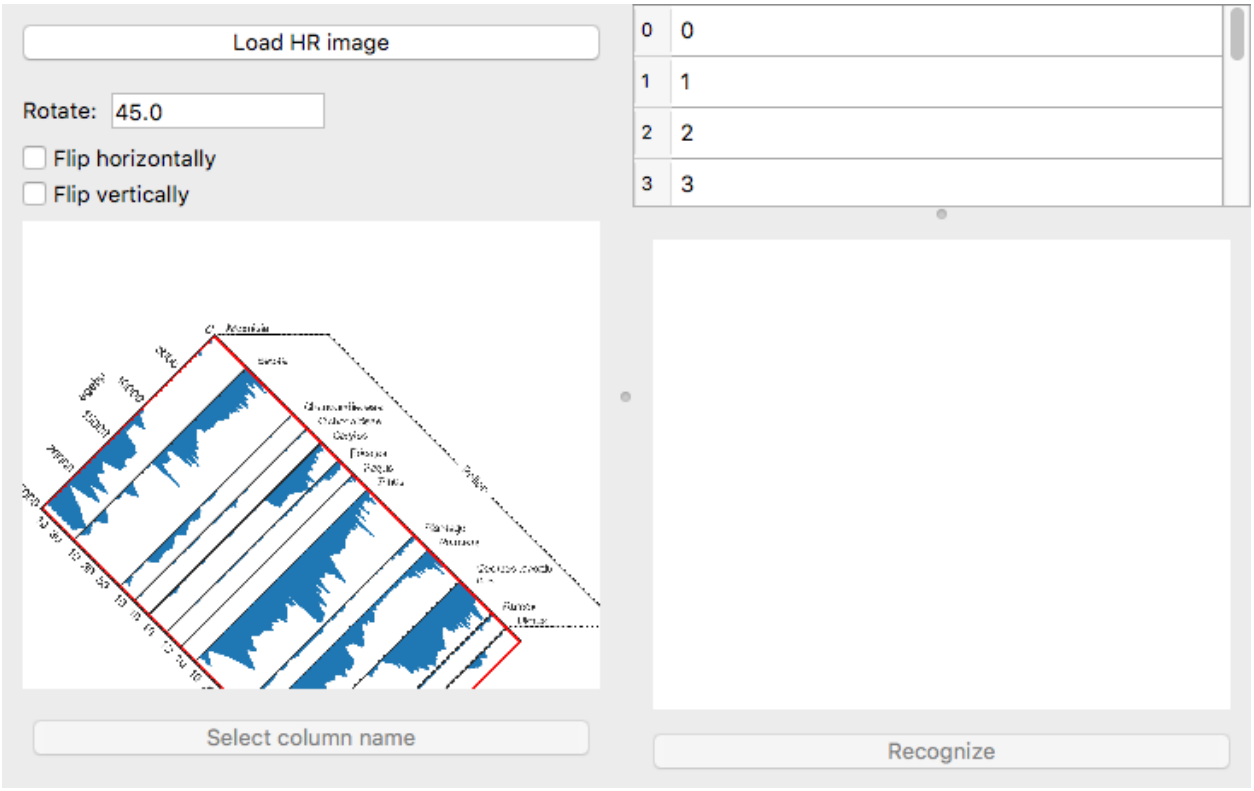
5. Click the *Apply* button

Handling column names

Within straditize, you can assign a name to each of the columns in your diagram, which will then turn up as the header in your final export. Usually these names are the titles of your columns, e.g. such as here



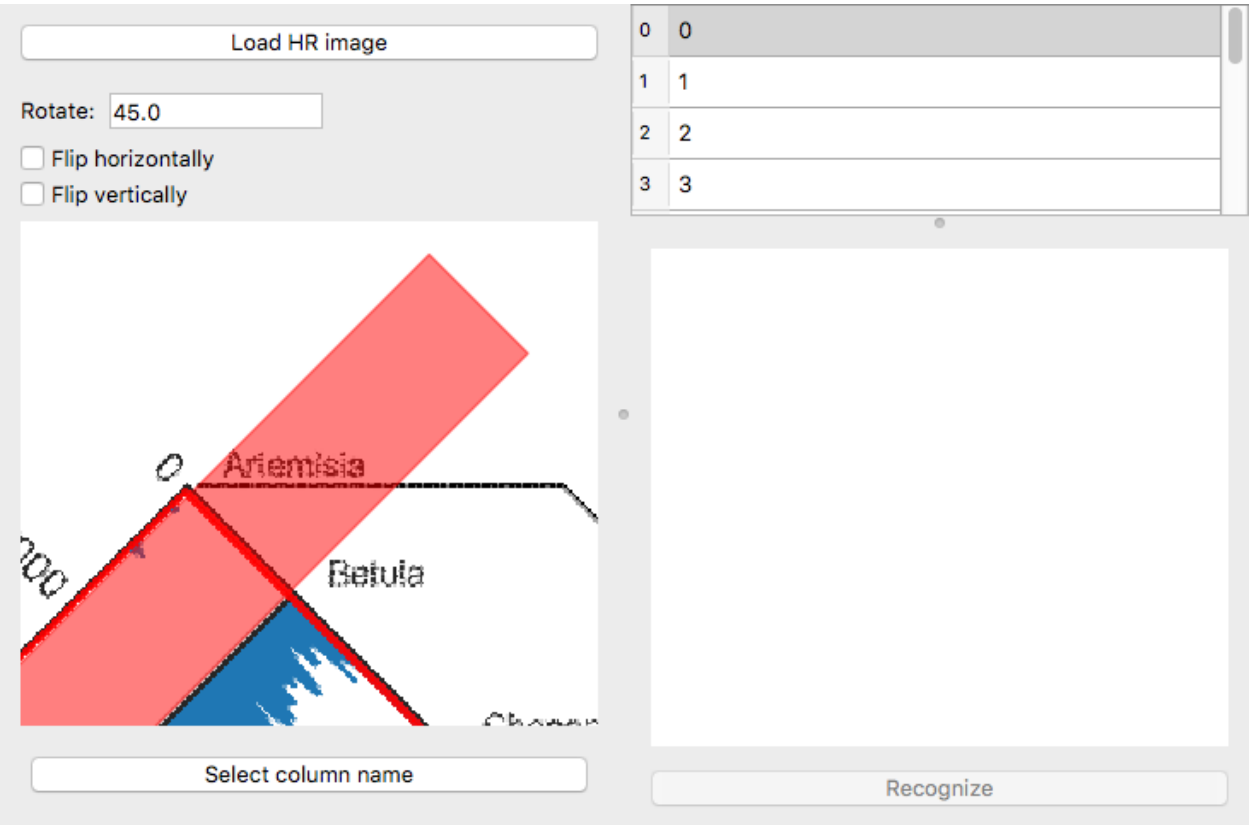
To enter the names of the columns, hit the *Edit column names* button in the *Column names* section of the straditizer control. This will open a dialog to handle the column names:



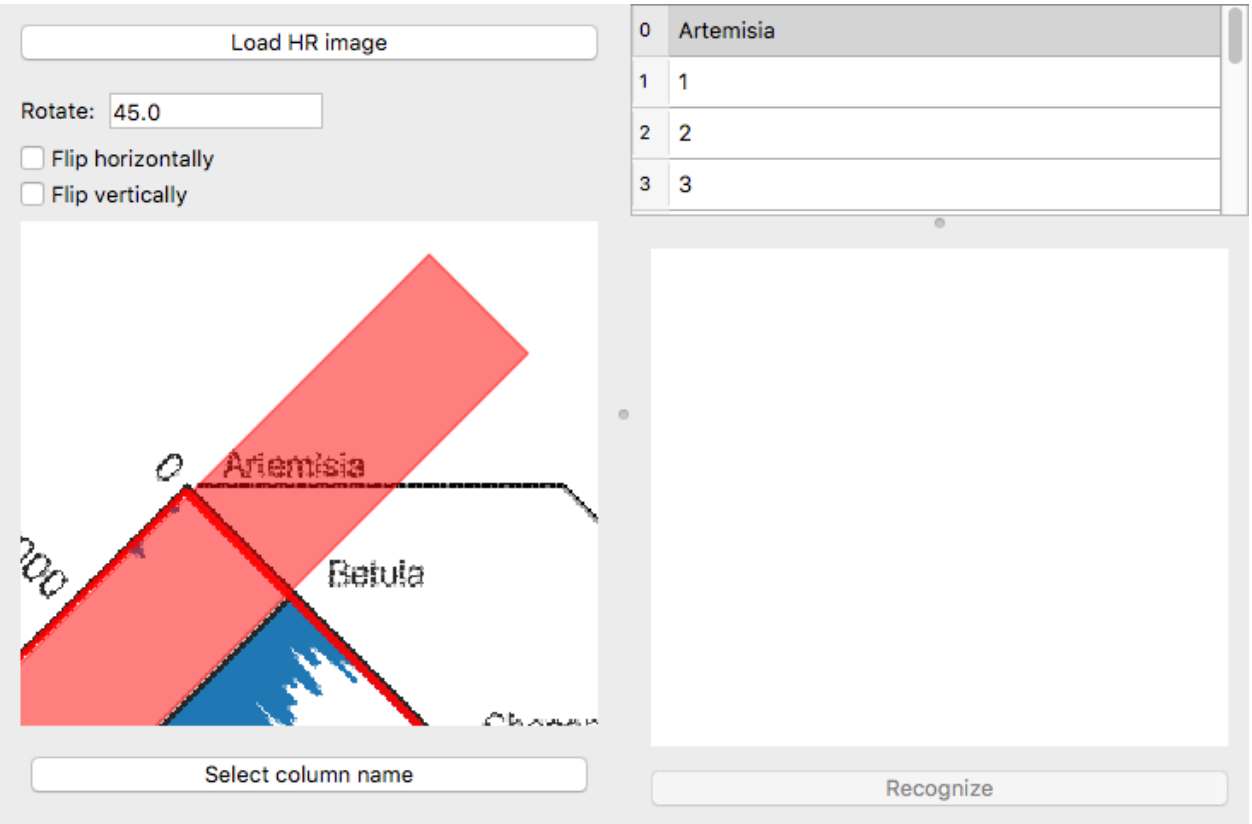
On the left side, you have a rotated version of the image that you can control with the options above it. To navigate in this window, place your mouse over it and then you can move (pan) the image with left-click.

Zooming in and out is done via right-click. To zoom in, hold your right mouse button and move your cursor to the upper right corner of your diagram. To zoom out, move it to the lower left corner.

On the right side you have a table with the column names. Initially, these are just increasing numbers starting from 0 for your first column. If you select one column, it will be highlighted in the left image:



You can now enter the name of the column, in this case Artemisia



Automatic optical character recognition (OCR)

You don't have to type the column name, you can also use the builtin text recognition. For this, we rely on the tesseract software that can be installed (on linux and MacOS) via:

```
conda install -c conda-forge tesseract
```

see the tesseract feedstock on conda-forge. If you want to automatically find the column names, you also need tesserocr to be installed which can be done via:

```
pip install tesserocr
```

or:

```
conda install -c chilipp tesserocr  # on linux/MacOS
```

(see https://pypi.org/project/tesserocr/ for more installation options, in particular for Windows).

1. Use a high resolution version of the image

To improve the result of the text recognition, we recommend to use a sufficient resolution of about 600 dpi. If the image that you are digitizing does not have this resolution, you can optionally load a higher resolution version of it using the *Load HR image* button at the upper left of the dialog.

We also recommend to remove everything in this file but the column names to improve the text recognition.

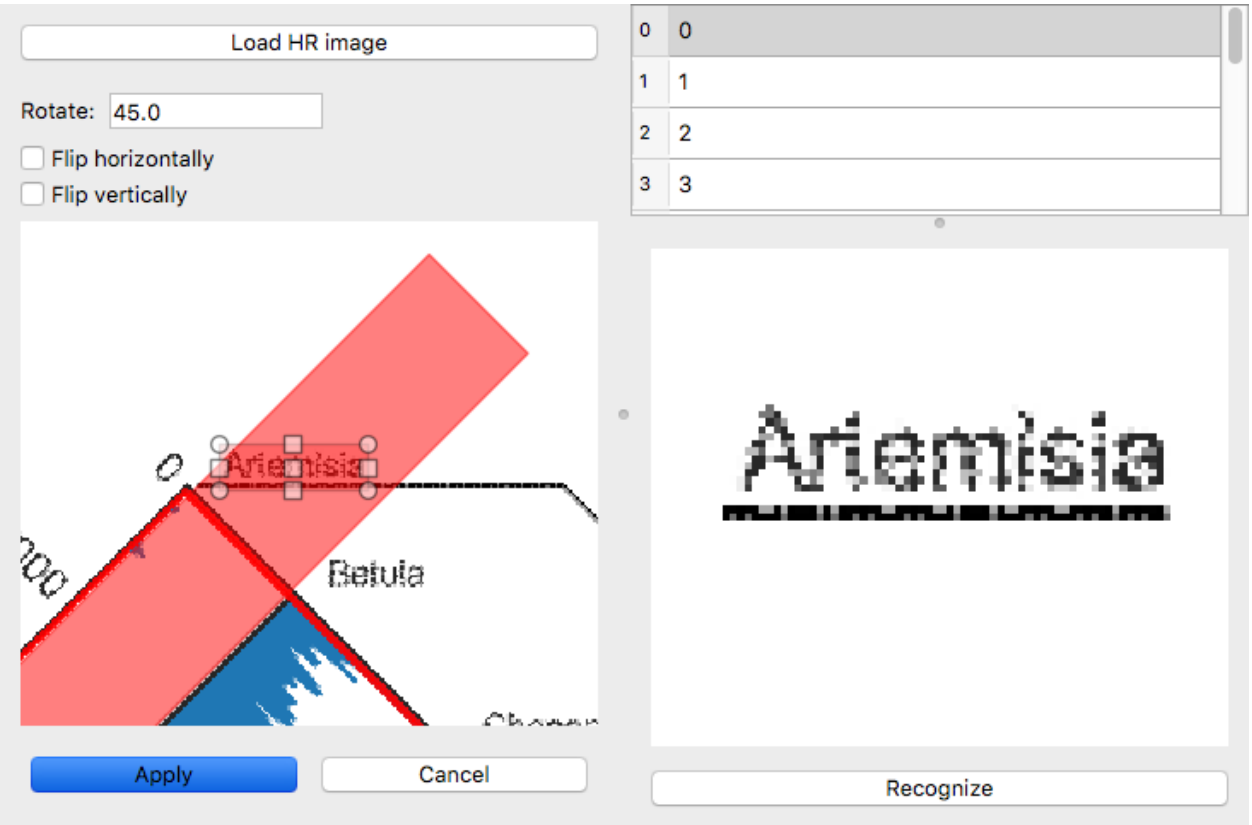
2. Automatically find all column names

Check the *all columns* checkbox and click the *Find column names* button. This then will look for column names in the image that is displayed on the left of the table and insert them into the table.

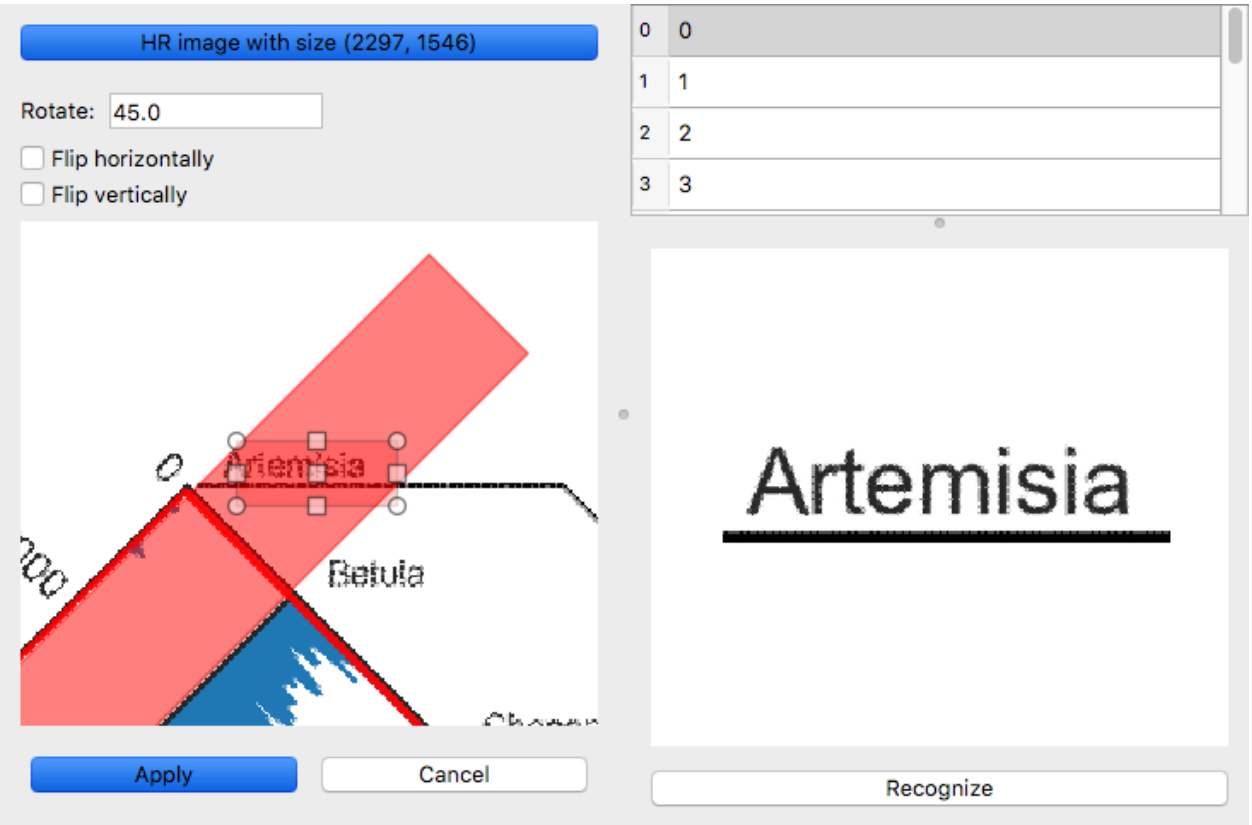
3. Separate treatment of column names

Now, select the first column in the table, zoom in such that you can see the name of the first column and click the *Select column name* button.

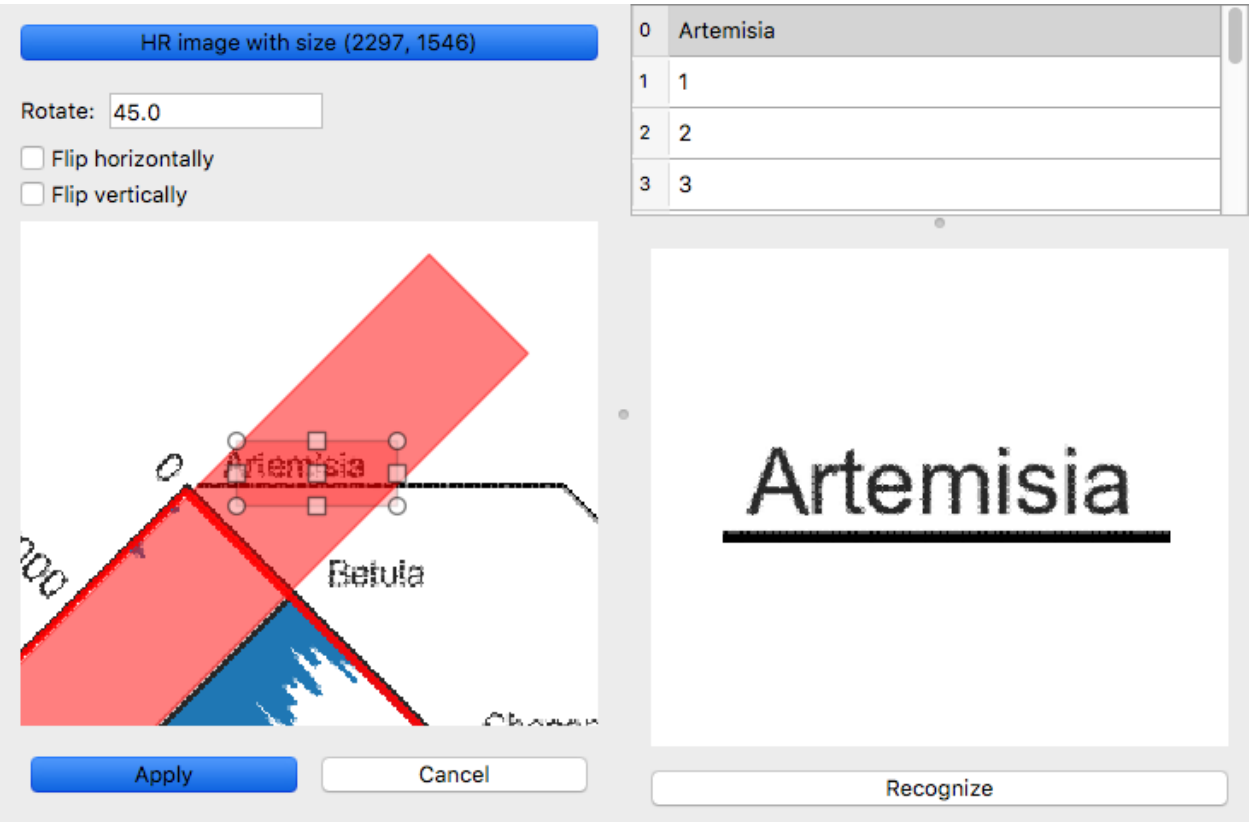
You won't be able to zoom or pan in your plot anymore, but you can now draw a rectangle in the plot around the name of the first column or use click the *Find column names* button (without having the *all columns* checkbox checked) and it will show up in the plot on the lower right of the diagram.



Here you see the result of the low resolution. If we instead specify a higher resolution image (see above), this can look much better



Now click the *Recognize* button at the bottom and it will use tesseract to read the text in the image and fill in the table



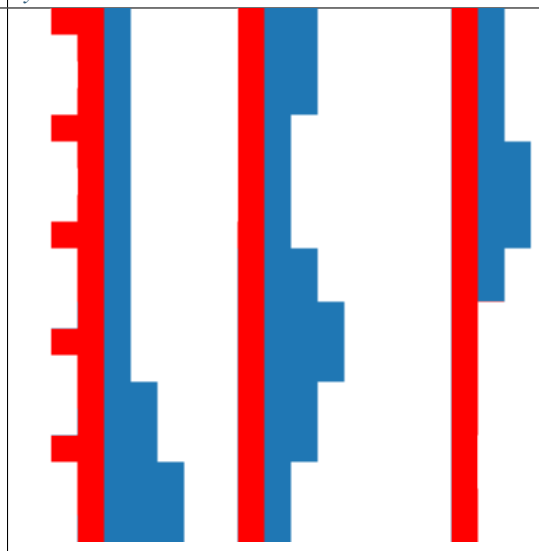
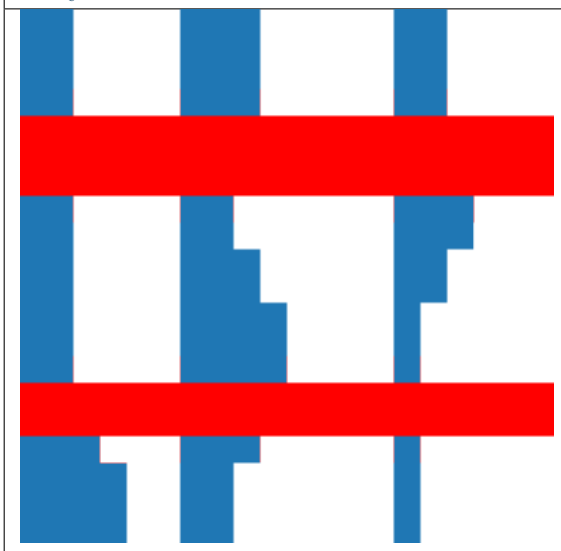
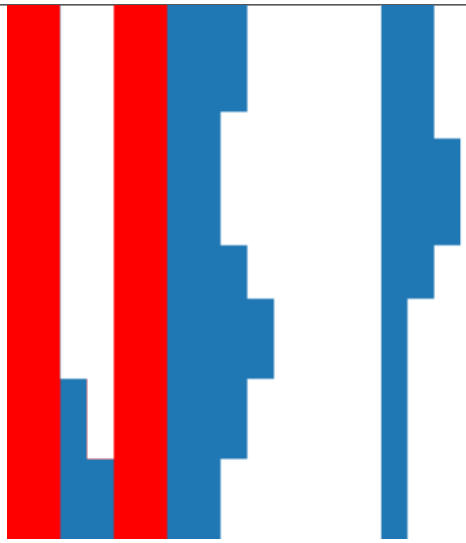
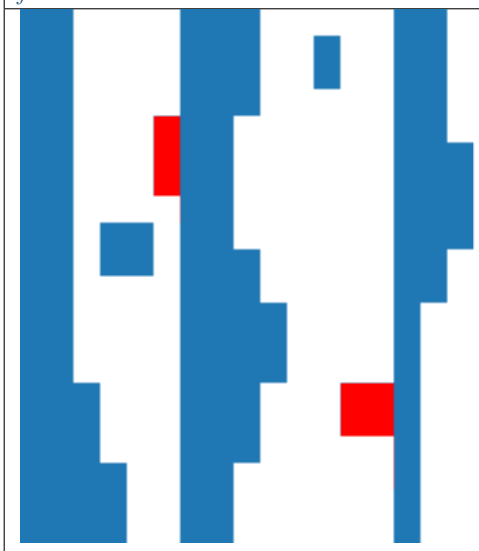
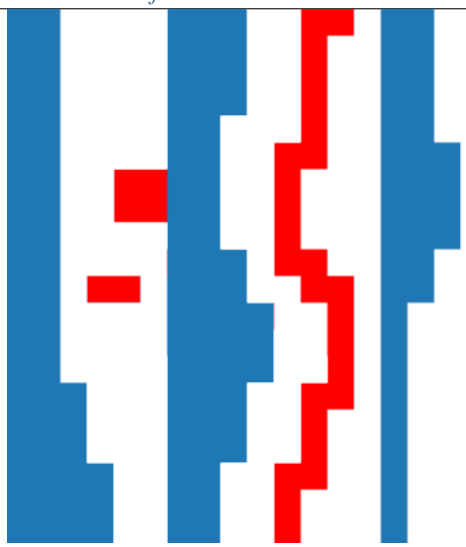
You can now select the next column or click the *Apply* button on the lower left. The latter will reenable the navigation (pan and zoom) in the plot.

Removing features

The diagram part is converted to a binary image where everything black (i.e. 1) represents data (see *Selecting the reader*). So before the data is digitized, features that do not represent data have to be removed.

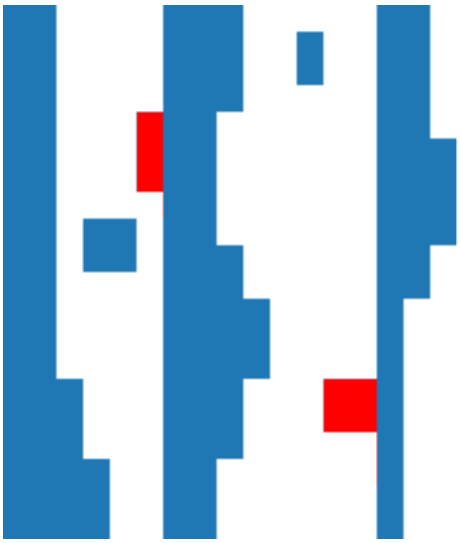
Here, you can either use the *features of the selection toolbar*, or you use one of the automatic removal tools we provide here. Note that you can edit every selection using the *mouse selection tools* or the *automatic tools* from the selection toolbar.

After you selected the features to remove, click the *Remove* button at the bottom of the straditizer control panel.

x-axes	*y-axes*
	
horizontal lines	*vertical lines*
	
features at column ends	*disconnected features*
	
cross column features	*small artifacts*
	

Remove features at column ends

This function automatically finds the features that touch the end selects them for removement.



Remove cross-column features

This function removes features that have a certain amount of pixels in multiple columns. Only those features are selected for removement, that have at least the given *Number of pixels* in more than one column of the stratigraphic diagram.

Note: This feature should be called before intializing column specific readers.

Remove disconnected features

This function automatically recognizes features that are disconnected either from the column start or from the previous feature in the column.

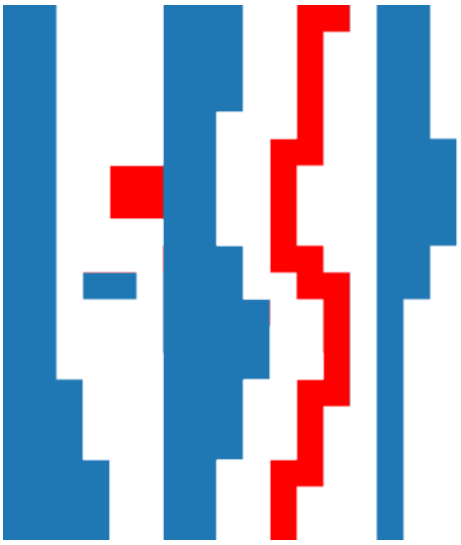
That can be useful, for example, to remove letters or symbols in the diagram.

What features are recognized can be defined in three ways:

1. Features that have a certain distance from the column start (see *Disconnected from the column start*
2. Features that have a certain distance from the previous feature in the column (see *Disconnected from the previous feature*)
3. A combination of 1. and 2. (see *Disconnected from start and previous*)

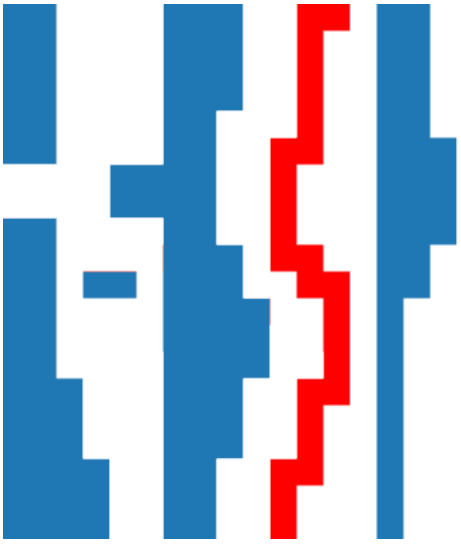
Disconnected from the column start

When only the checkbox *from column start* is checked, all features that have the given distance from the column start are selected.



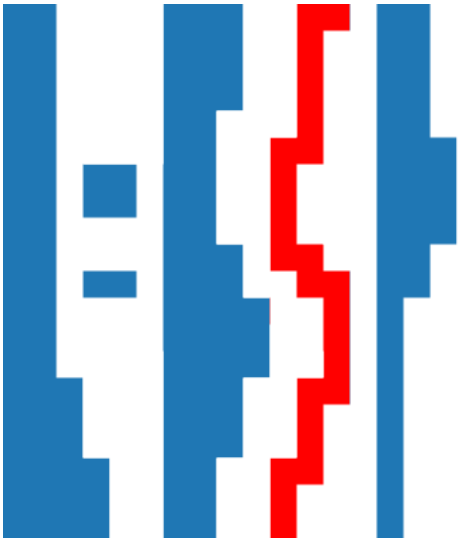
Disconnected from the previous feature

When only the checkbox *from previous feature* is selected, all features that have the given distance from the previous feature in the column (and everything to the right of it) is selected.



Disconnected from start and previous

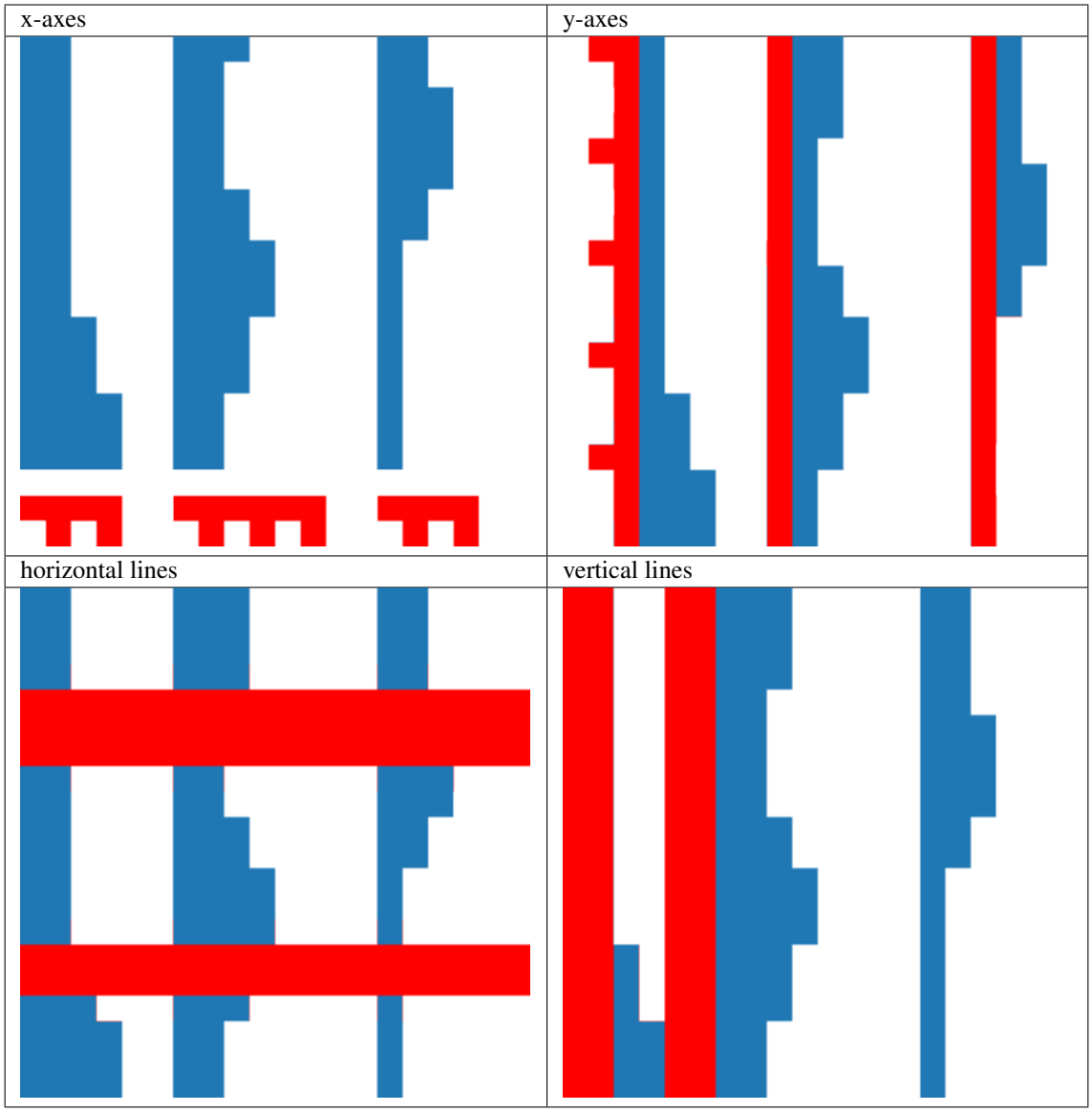
If both are selected, the selected feature must have the given distance from the column start and from the previous feature in the column to be selected.



Remove horizontal or vertical lines

The features described here automatically remove the y- and x-axes, or in general vertical or horizontal lines that span a certain fraction of the diagram part.

The methodology is simple: If a certain pixel column or row in the binary data image is covered, it is considered as a vertical or horizontal line, respectively.



You can modify the recognition using three options

Minimum fraction: This is the minimum fraction of a pixel column (for or row that must be covered

Minimum line width: Lines are only selected, if their line width is greater than or equal to the given minimum axis width.

Maximum line width: Lines are only selected up to the given maximum axis width

Remove small artifacts

This method recognizes small artifacts using the `skimage.morphology.remove_small_objects()` method.

The pixel size of the artifacts can be determined through the *Smaller than* field and you can always modify the selection through the tools in the selection toolbar.

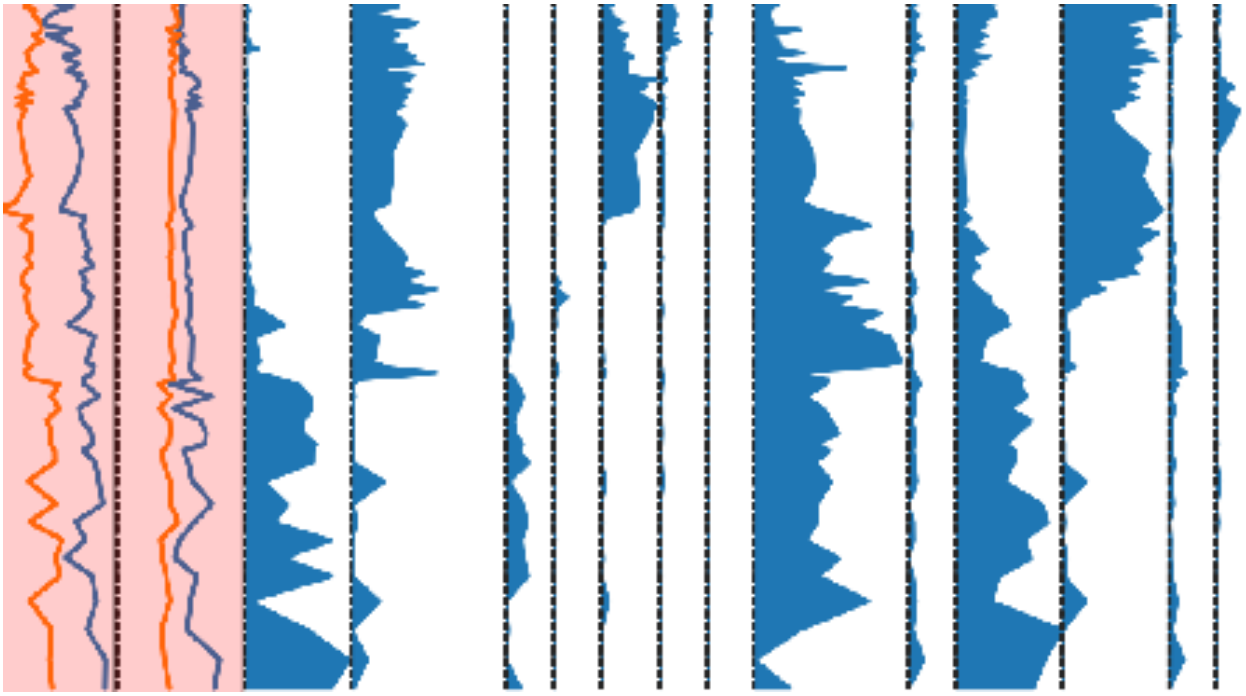
If you want a better visualization of the small artifacts that will be removed, use the *Highlight selection smaller than* button.

Column specific readers

Note: If the entire diagram is just based on one diagram type (see *Selecting the reader*), you can skip this step.

As noted in the *reader selection section*, you can choose different readers for different columns in order to account for multiple subdiagram styles within the stratigraphic diagram. The current reader, i.e. the one that is selected in the *Current reader* tab in the *Digitization control* is the one that can be accessed via the `stradi.data_reader` attribute in the console. It is also sometimes referred to as the *parent reader*.

Clicking the + button in the *Current reader* tab let's you select columns just by clicking on the data image.

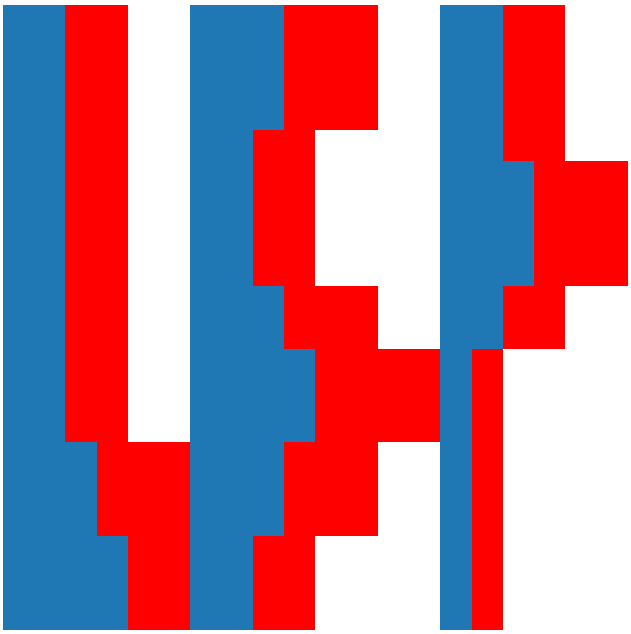


After you selected the necessary columns, hit the *Apply* button. In the appearing window, select the reader type you want (see *Selecting the reader*). You now initialized a new child reader for the selected columns that can be set as the parent reader using the dropdown menu in the *Current reader* tab.

Interpreting exaggerations

Visualizing an exaggerated version of the data is a common methodology in pollen diagrams, since even small pollen counts can say a lot about the environment.

Therefore it is common to not only plot the data, but also to exaggerate it by a certain factor, e.g. 10 or two like the red areas in the image below.



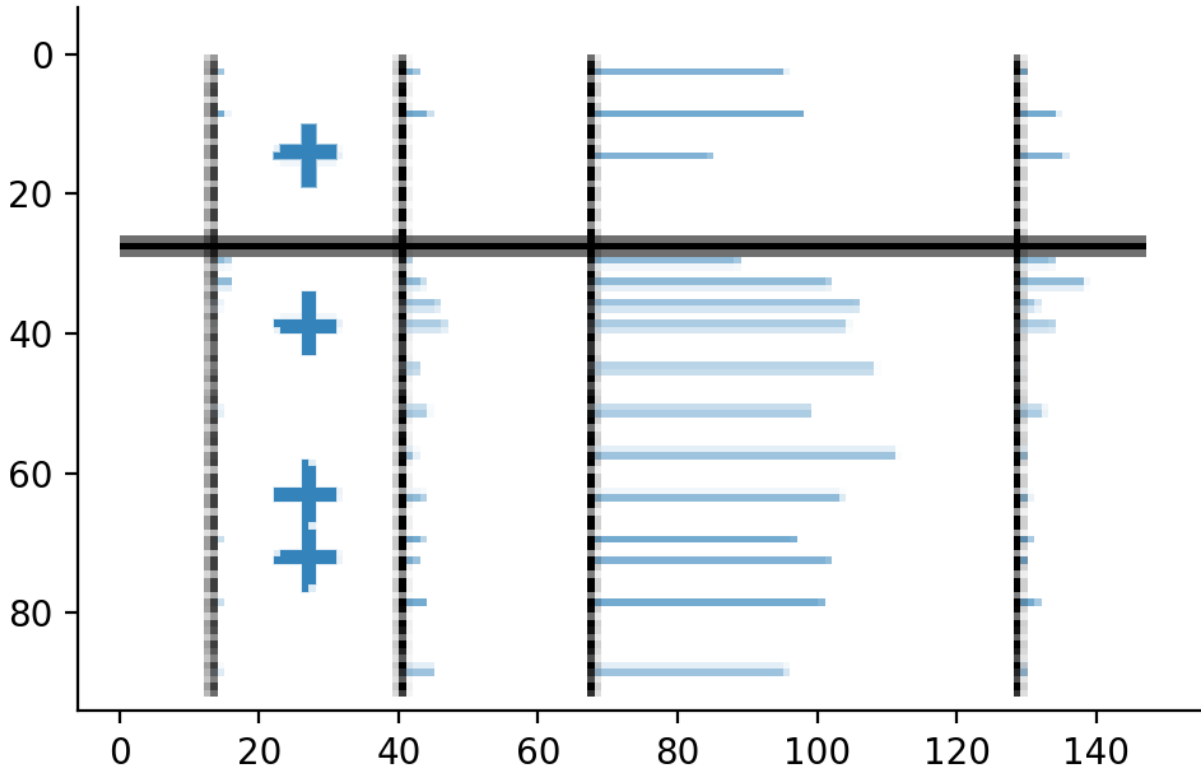
You have two choices: Either you remove these exaggerated areas (see *Removing features*), or you merge them into your results to improve the digitization result.

To include and interprete the exaggerations, straditize can create a new reader dedicated for the exaggerations which you then have to select and specify when they should be used. This is all done in the *Exaggerations* tab of your digitization control.

1. Specify the *exaggeration factor*. This number is usually described in the caption of your diagram and must be greater than 1.
2. Select the data reader type for the exaggerations (see *Selecting the reader*). Most common this is one of *area* or *line*.
3. Initialize the reader for the exaggerations by clicking +. You will now find a new reader dedicated to the exaggerations in the dropdown menu of the *Current reader* tab. But for now, we stick with the original one.
4. Now, you then have to select the exaggerations. Click the *Select exaggerations* button and use the tools in the *selection toolbar* for selecting the features in the diagram that represent the exaggerations. When you're done with this, click the *Select* button. You can also repeat this step to select more and more features.
5. Specify when the exaggerations should be used. This can be either where the unexaggerated data is below a certain percentage of the image width and/or below a certain number of pixels.
6. When you now click the *Digitize exaggerations* button (but after you clicked the *Digitize* button for the original reader), the exaggerations will be digitized and merged into the digitization result of the non-exaggerated reader.

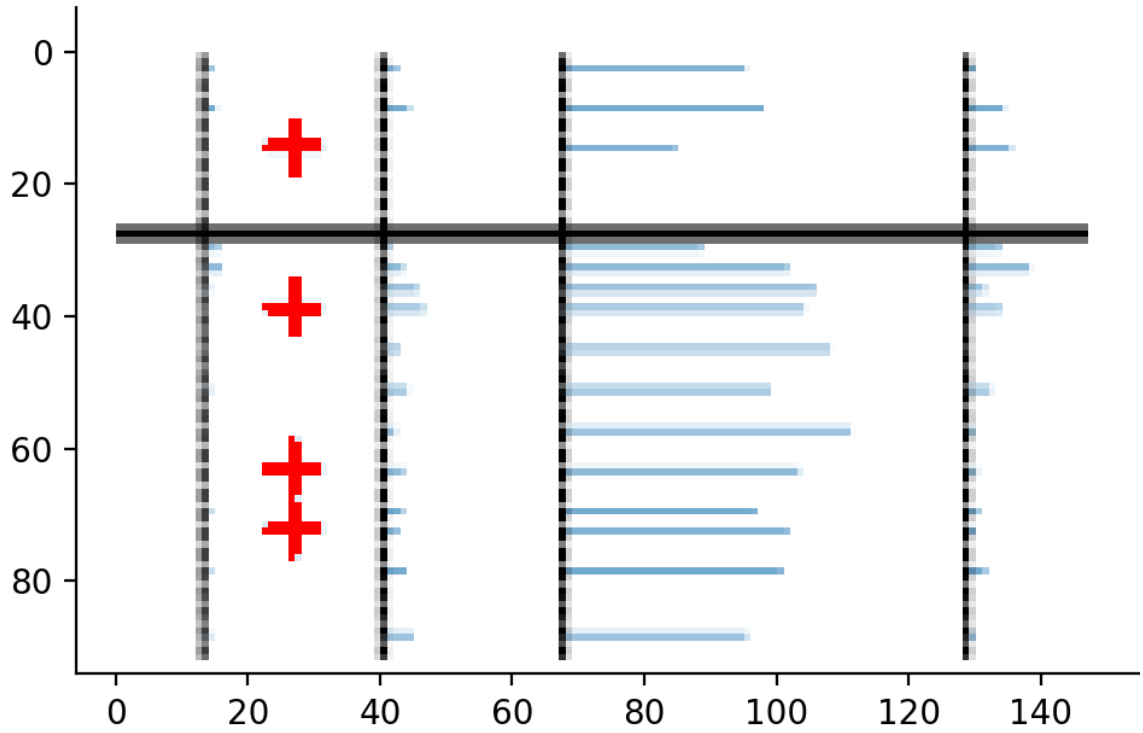
Select occurences

Pollen diagrams often mark low taxon percentages to highlight the occurence of a taxon, such as the + in the image below

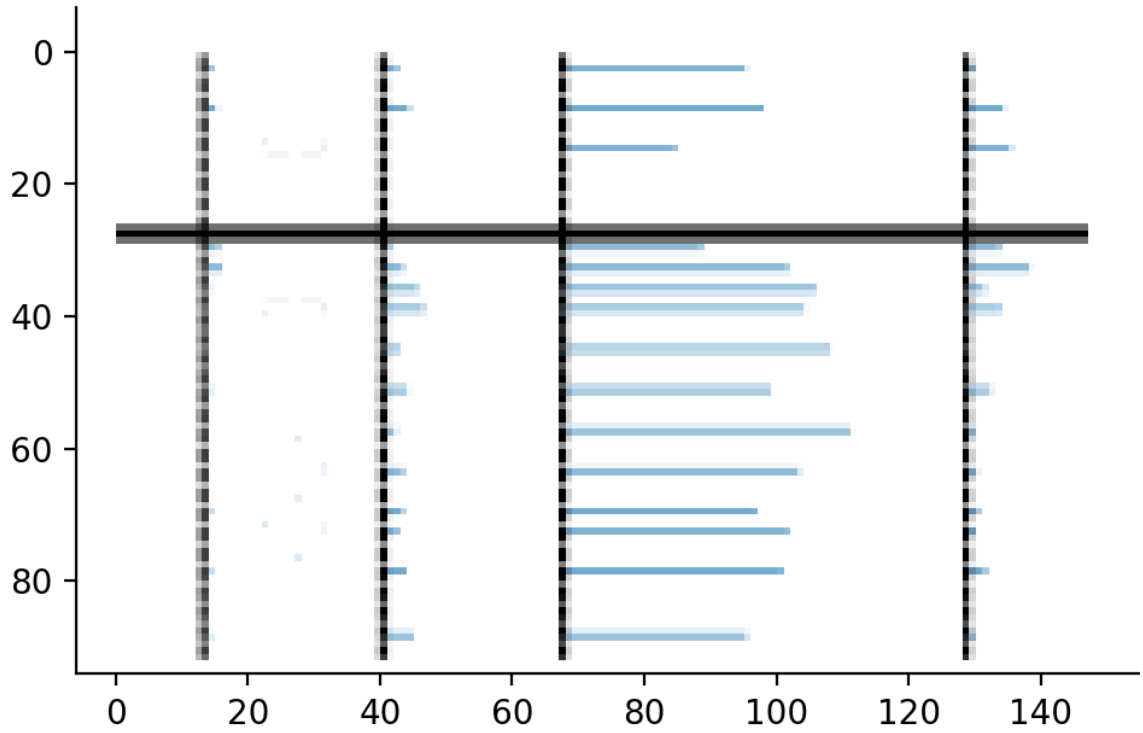


This is very useful information which can be considered by straditize.

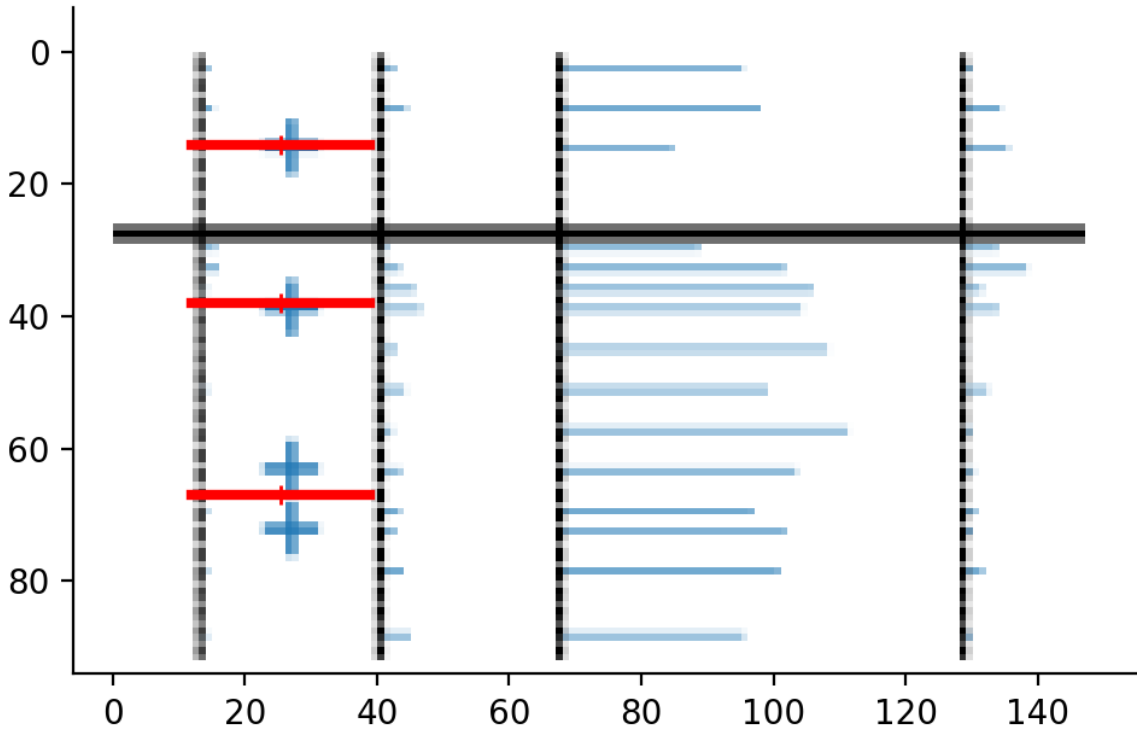
1. Expand the *Occurences* tab in the *Digitization control* section
2. Click the *Select occurences* button and select the occurences using the tools from the *selection toolbar*, especially the tool is useful



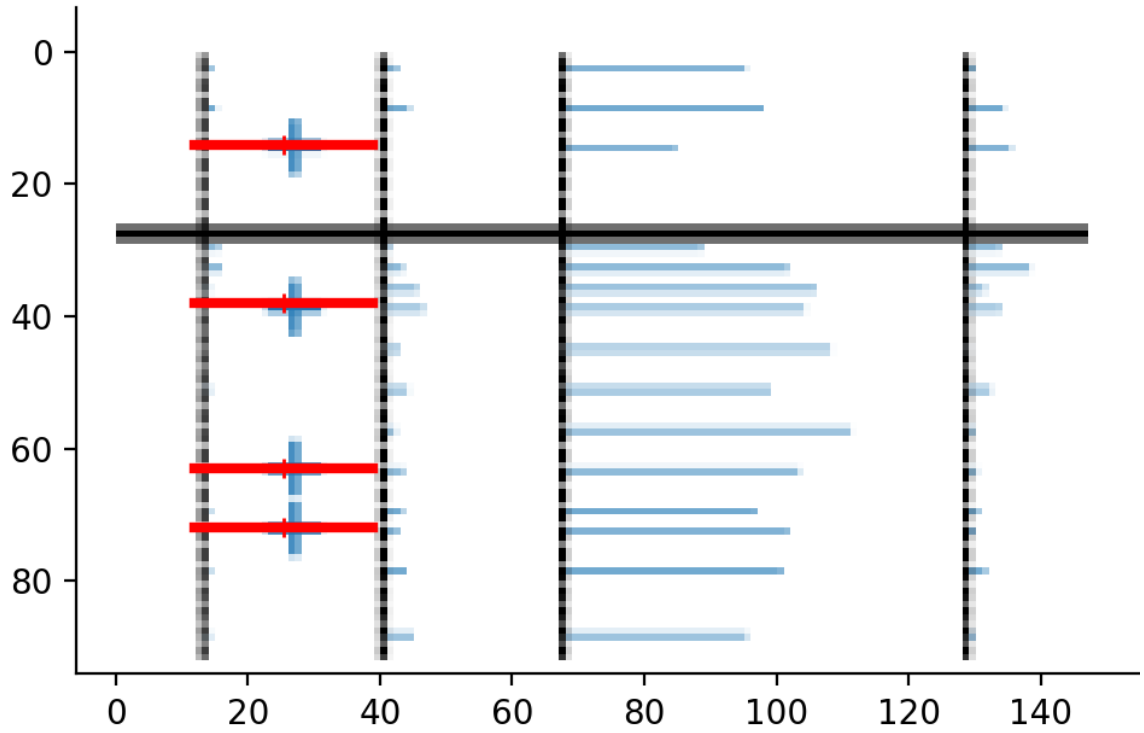
3. Click the *Apply button* and the selected features in the diagram will be registered as an *occurence*. If you also checked the *Remove on apply* checkbox next the to *Select occurences* button (which you normally should), the selected features in the diagram will be removed, too and you will end up with a plot like this



4. You can see and edit the occurences using the *Edit occurences* button. This will create horizontal markers at the positions of the original occurences



In our demonstration, straditize could not distinguish two of the occurence markers because they were too close to each other. But you can just move the mark with Left-click and Shift+Leftclick on the plot to create a new marker



Click the *Apply* button when you are done.

5. The marked occurences will appear later in the final data frame with the specified *Occurence value*.

Digitizing the diagram

The digitization transforms your picture into a table. The outcome is a data frame with one value for each row and column in the data part of your image. After the digitization, you can access this raw data from the command line via:

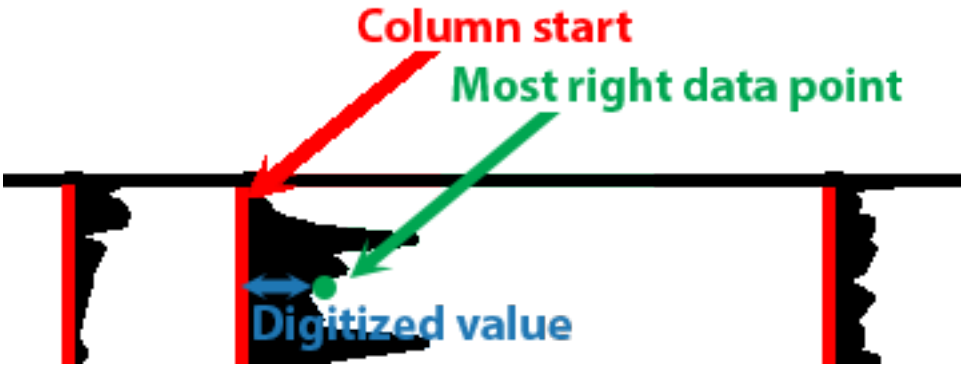
```
stradi.data_reader.full_df
```

and you can plot it using the *Full digitized data* item in the *plot control*.

The digitization procedure depends on the reader type that you are using (see *Selecting the reader*).

Contents

Digitizing area and line readers



This class of stratigraphic diagrams is the most popular within the pollen diagrams and it's digitization is fully automatic.

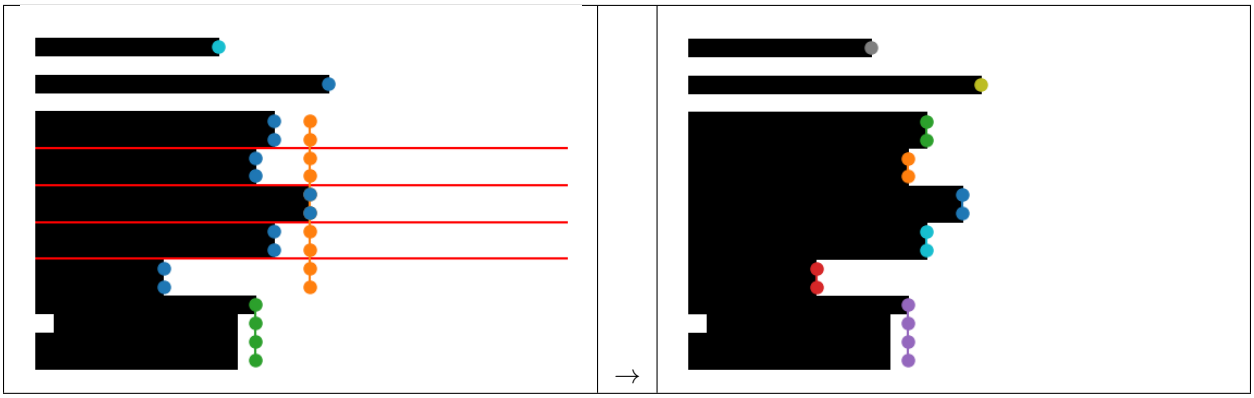
The algorithm simply uses the distance of the most right data (black) pixel to the column start.

Digitizing bars

For digitizing bar diagrams, it is crucial that the software finds and distinguishes adjacent bars. This is mainly done through the *tolerance*. The algorithm is: if the pixel row, compared to the previous row, is higher or smaller by the given *tolerance* , the software assumes that those two rows belong to two distinct bars.

The image above illustrates this. The reader distinguishes the bars that are clearly separated or were one bar is significantly lower or higher than the other, but it cannot distinguish bars were there is only a small difference in height.

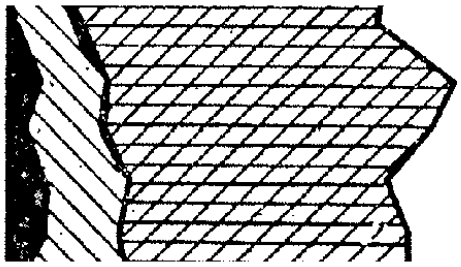
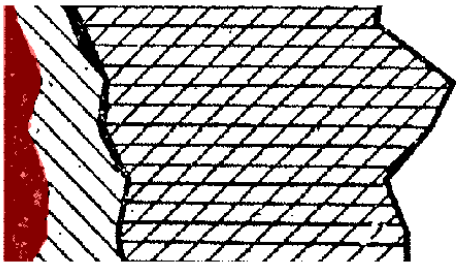
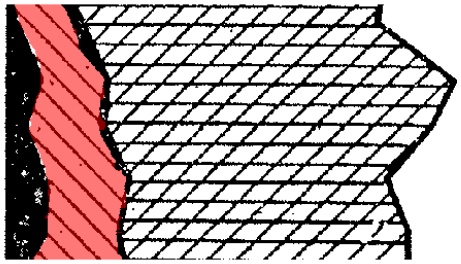
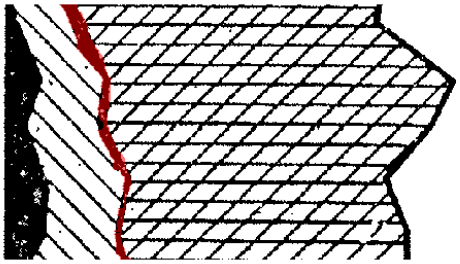
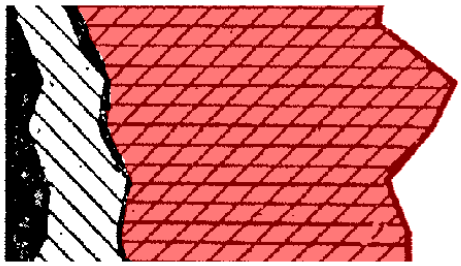
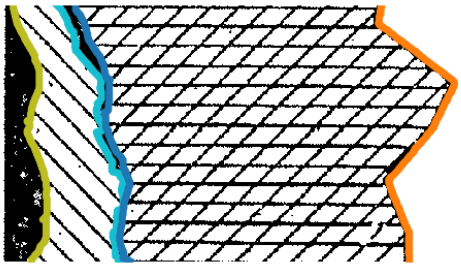
Splitting bars



To overcome this problem, straditize automatically finds the bars that are wider than the others and marks them for splitting. If the straditizer finds bars that are too long, they are shown in the *Split too long bars* tab of the digitization control. To split one bar:

1. Double-click an item in the table (the plot will zoom to the bar)
2. Left-click on the bar to indicate the vertical location for a split (right-click again on the same line to revert this). A red line will appear (left-click, see image above) or disappear (right-click)
3. Click the symbol to continue with the next bar that is too long or double-click an item in the table. Again split the bar as described in 2.
4. When you are done, hit the *Apply* button and the bars will be splitted right were you marked it

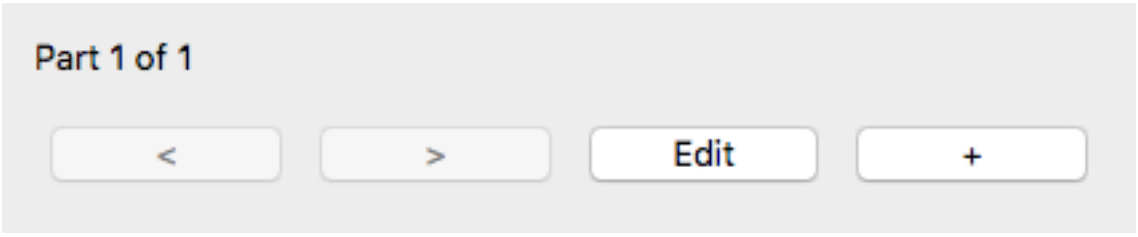
Digitizing stacked areas

	initial		column 1
		→	
	column 2		column 3
→		→	
	column 4		column 5
→		→	

Stacked diagrams are more difficult to digitize because of the variety of combinations. The columns might be distinguished by colors or patterns (hatches). Therefore, the user has to select the features corresponding to one column, one after the other.

The digitization here works as follows:

1. click the *Digitize* button, you will see a control for navigating within the columns



2. click the + button to setup a new column. Everything in the column will be selected and now you have to select only those features, that are part of the first column using the selection tools (see the *column 1* above).
3. When you did this, click the *Apply* button.
4. click the *Edit* button if you want to edit the column that you just selected

5. Click the > button to go to the right column
6. click the + to add a new column and select the second column (see above)
7. repeat the steps 3-6 until you digitized all columns

If you want to display your digitization result, use the *Full digitized data* in the *plot control*.

Hint: you can change the transparency of the selection with the transparency slider in the *selection toolbar*. That helps to still see the selected features

---

Samples

samples are the source data that has been used to create the diagram (see *Basics and Terminology*). The crucial part after the digitization of a diagram is to find out, where these samples exactly are.

Straditize assists you with this through automatic sample finding algorithms, or you can load the sample locations from an external files or you just add and edit the samples manually.

Automatic samples identification

straditize has reader specific sample identification algorithms implemented. The overall algorithm is based on two steps:

1. For each column: Identify the intervals that contain exactly one samples (the rough locations)
2. Align the overlapping intervals between the columns to estimate the exact location

Where step 2 is the same for all diagram types, step one is diagram specific. For bar diagrams, we use the bars that have been identified during the digitization (see *Digitizing bars*).

For area and line diagrams on the other hand, we identify the local extrema (minima and maxima) for each column. This strategy works well for pollen diagrams since they are supposed to sum up to 100%. However, this does not generally work for all stratigraphic diagrams.

Load samples locations from external file

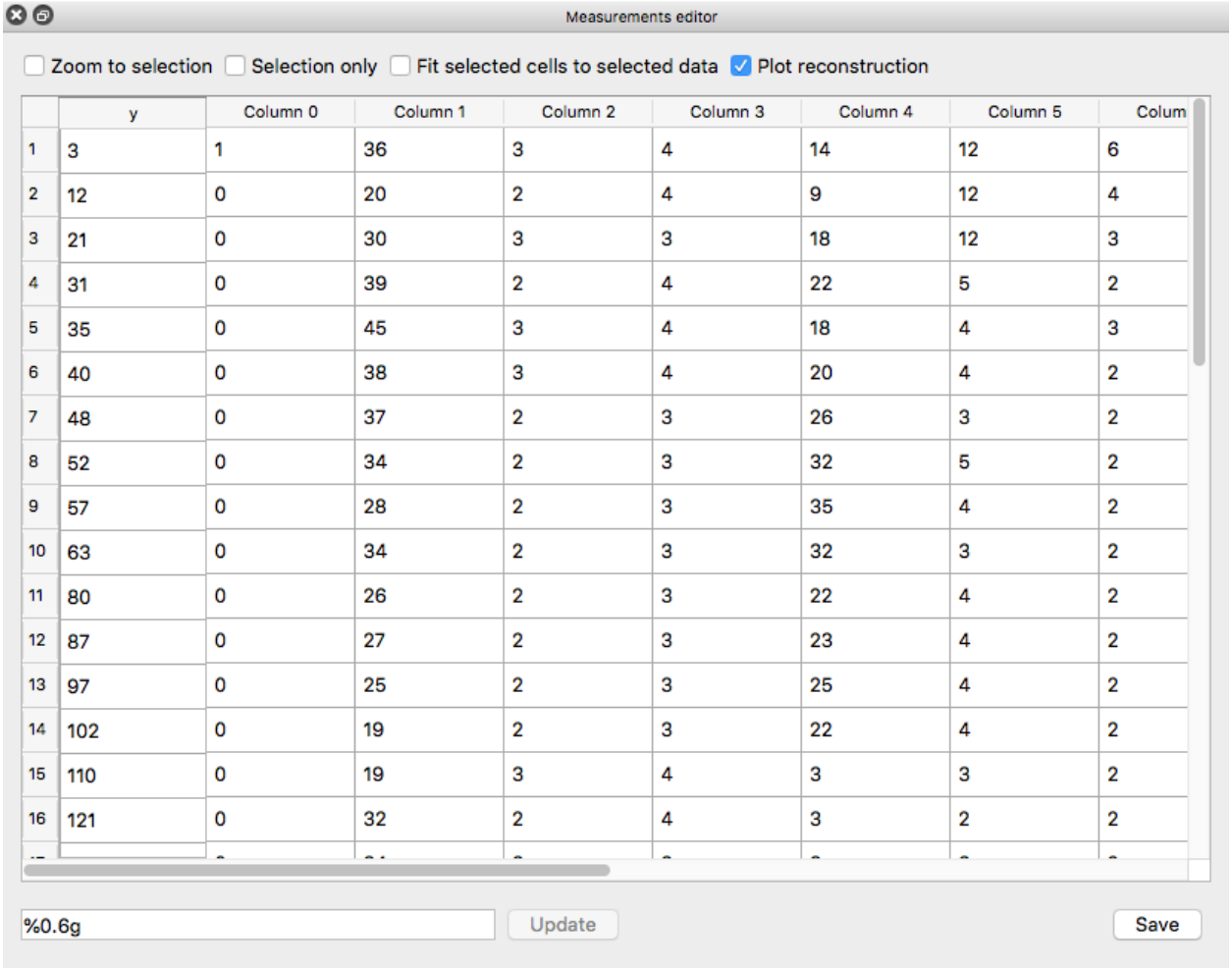
samples can be loaded from an external CSV (comma-separated) file. The first column in this CSV file is expected to represent the y-locations of the samples. If the y-axis scale has already been specified (see *Translating the shared vertical axis*), this data should be on the same scale. Otherwise, we assume that the data is in pixel coordinates.

Warning: Make sure, that you first entered the correct *y-axis scale*. Otherwise the samples might not be interpreted correctly.

Editing samples

To edit samples, click the *Edit samples* button and a new figure will be created with one mark for each sample and a table is shown to you with the samples data. To edit the samples, you can edit the table or the marks in the plots

The sample table



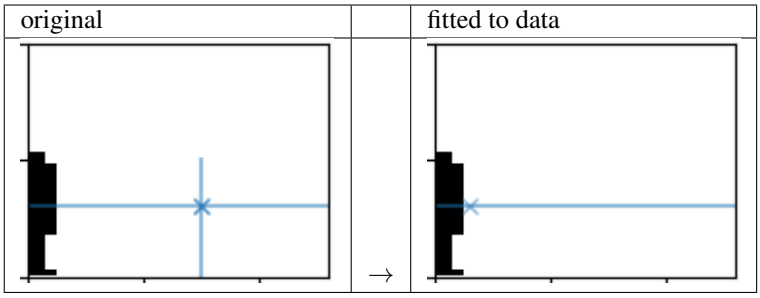
	y	Column 0	Column 1	Column 2	Column 3	Column 4	Column 5	Colum
1	3	1	36	3	4	14	12	6
2	12	0	20	2	4	9	12	4
3	21	0	30	3	3	18	12	3
4	31	0	39	2	4	22	5	2
5	35	0	45	3	4	18	4	3
6	40	0	38	3	4	20	4	2
7	48	0	37	2	3	26	3	2
8	52	0	34	2	3	32	5	2
9	57	0	28	2	3	35	4	2
10	63	0	34	2	3	32	3	2
11	80	0	26	2	3	22	4	2
12	87	0	27	2	3	23	4	2
13	97	0	25	2	3	25	4	2
14	102	0	19	2	3	22	4	2
15	110	0	19	3	4	3	3	2
16	121	0	32	2	4	3	2	2

The sample table shows you the vertical coordinates of the samples and the values for each column obtained through the digitization.

Each row in this table represents one sample in the diagram.

The first column is the vertical location (y-axis) of the sample, the remaining columns are the x-values of the samples. The numbers shown in the table are in pixel coordinates of the diagram part. To interact with the samples, you can

1. Right click the table to add new or delete old samples/rows below or above the selected row
2. Edit the numbers in the cells to change the values for the samples.
3. Fit the x-values of the samples to the data.
    i. Either by right-click and *Fit selected cells to the data*, which will adjust the column values of the mark such, that it fits with the digitized value

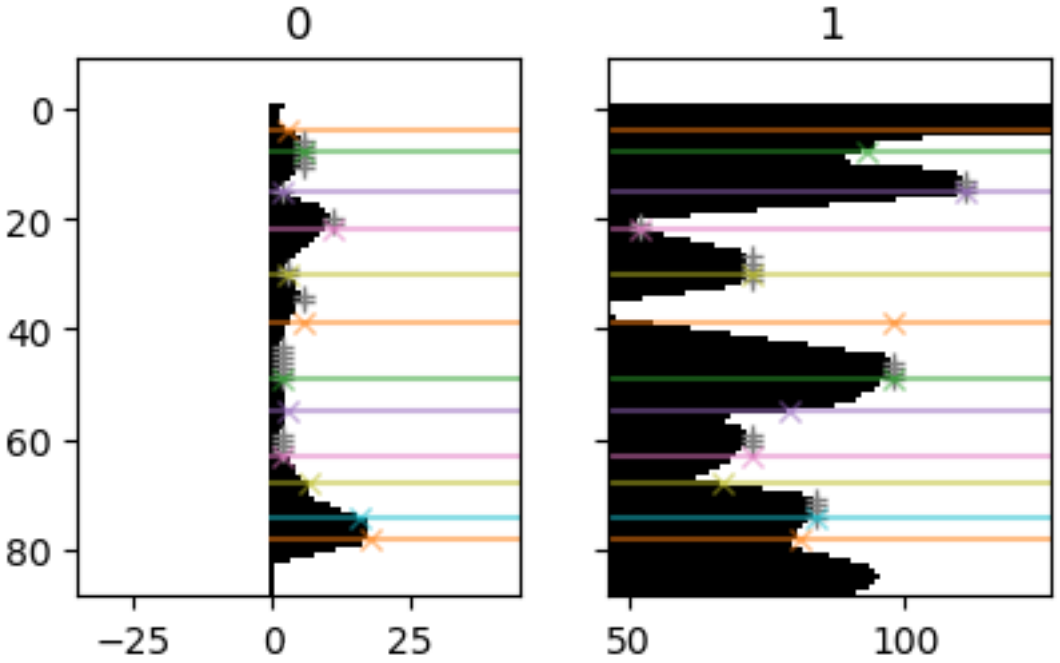


ii. Or by using the *Fit selected cells to selected data* checkbox. If this is checked:

1. select cells in the table
2. click on the plot
3. the cells will be updated with the x-values from the digitization at the click-position

You can also zoom the selection or hide everything else but the selection.

Editing the marks



Additional to the table, you have a visual representation of the samples in the figure. Here, you have one plot per column each mark represents the vertical location of a sample. The marker `x` in the line shows you the value of the sample (i.e. the location on the x-axes).

1. move a sample by
   i. left-click a mark and hold the mouse button (the mark will change it's color)
   ii. while still holing the mouse button, drag the mark to a different location
2. delete a sample by right-clicking the mark

3. add a new sample by holding down the `Shift` button and left-click on one of the plots.

Translating from pixel to data coordinates

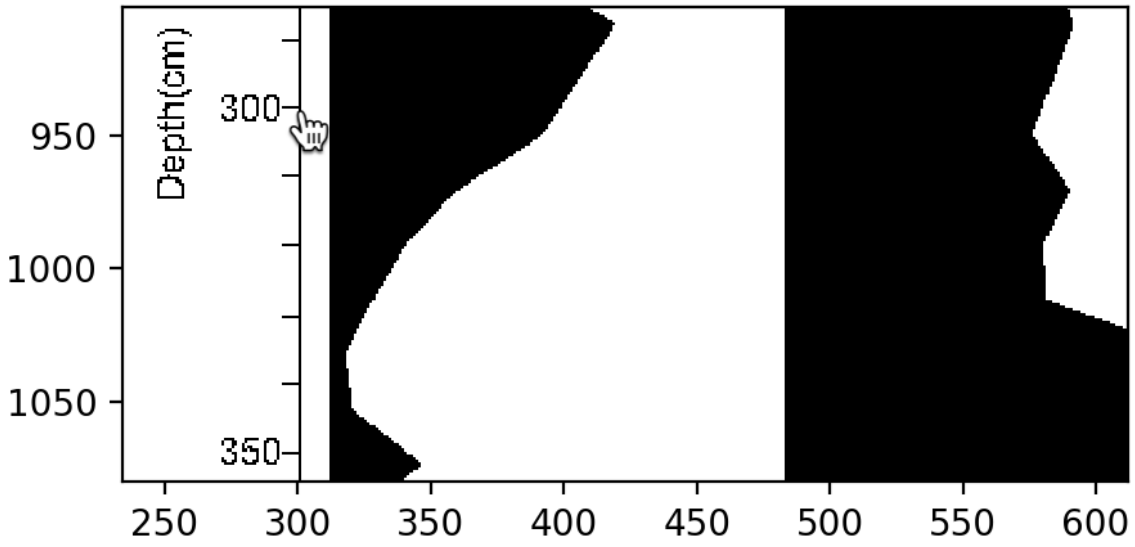
All the previous steps in the digitization where in pixel coordinates. However, for the final output, we need to translate this into the real units of the diagram.

Translating the shared vertical axis

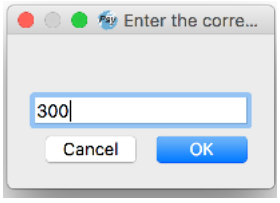
For your final product, straditize needs to know about how the data in the diagram is scaled, such that we can transform the final sample information from the pixel coordinates into the units of the diagram (e.g. years or meters in case of time or depth).

To translate the y-axis information,

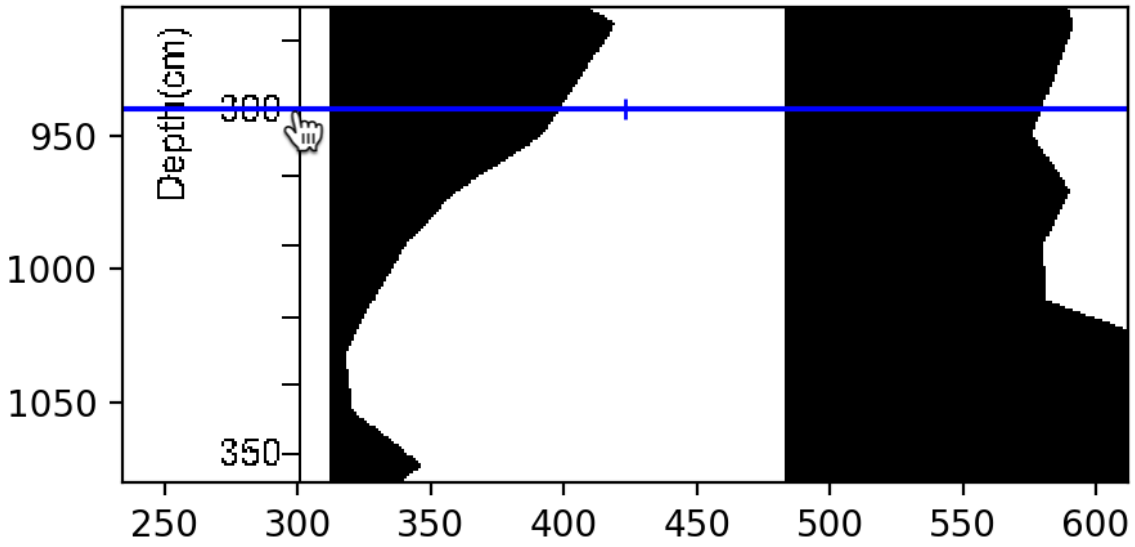
1. Expand the *Axes translations* tab in the digitization control
2. Click the *Insert Y-axis values* button in the *Axes translations* section of the straditizer control (if not already done)
3. Shift-leftclick on the plot to enter the corresponding y-value.



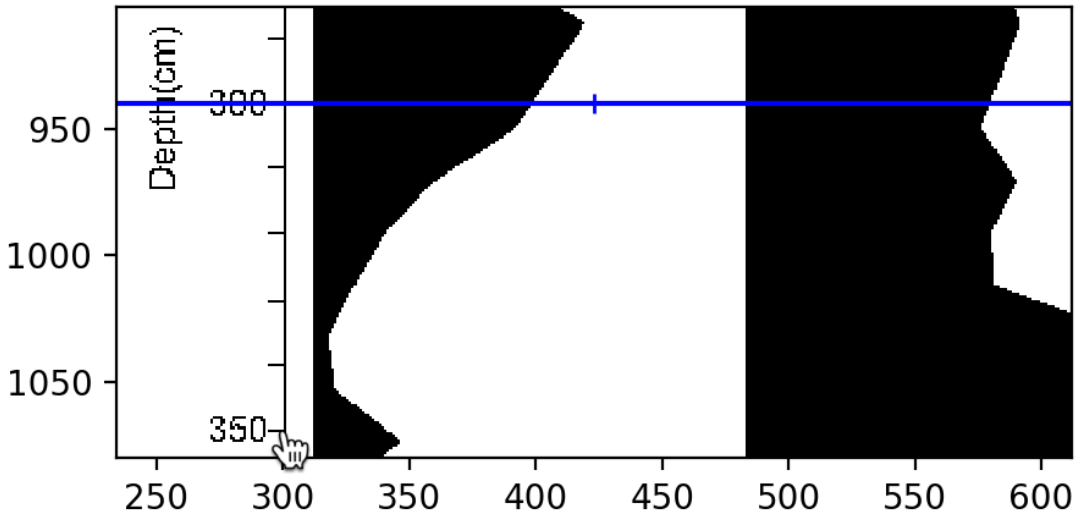
4. A small dialog will appear where you should enter the y-value to use (in this case, `300`)



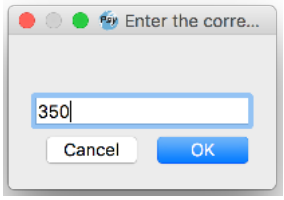
5. After hitting the *Ok* button, you will see a mark on the plot (blue line). You can select the mark via leftclick and drag it to a different location or you can delete it via rightclick.



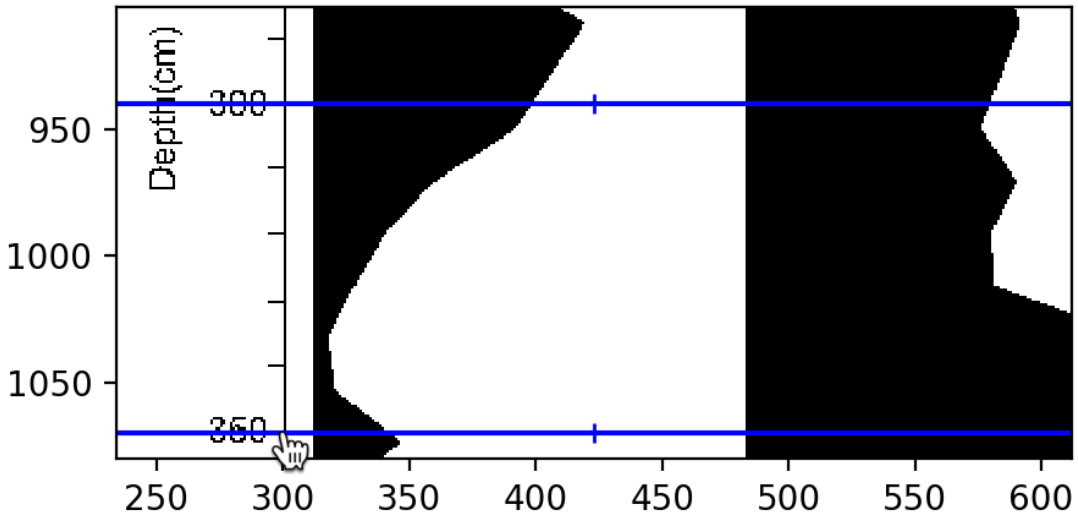
6. now repeat steps 2-4 on a second point on the y-axis
    - Select another point



- Enter the corresponding value (here `350`)



- A new mark is created that you can modify



7. Click the *Apply* button at the bottom of the straditizer control when you are done.

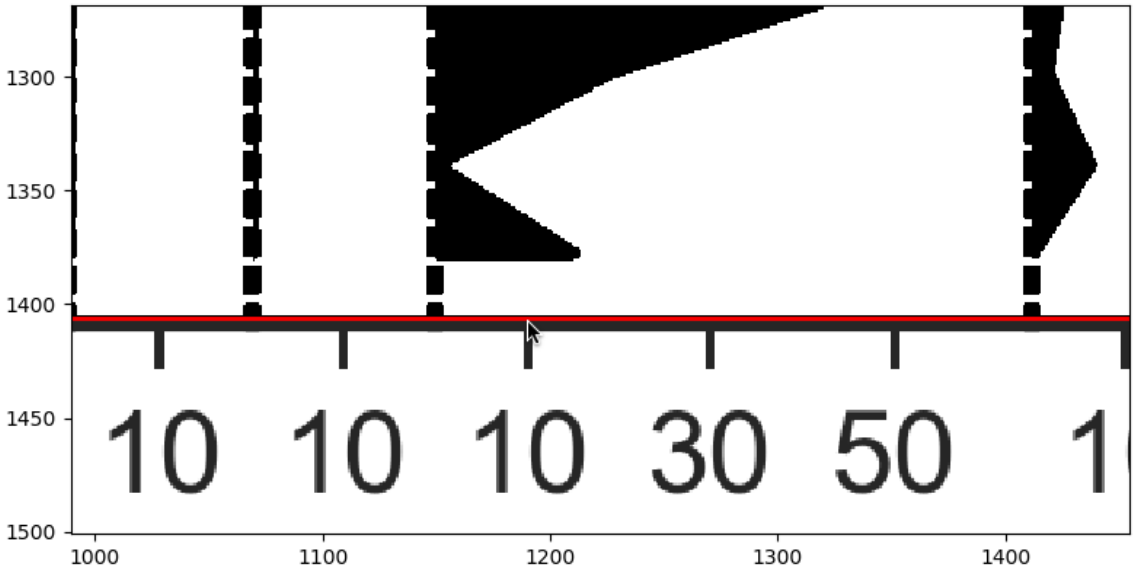
Note: If you drag a mark and hold the `Shift` button while releasing the mouse button, the dialog in point 3 from above will not pop up.

Translating the horizontal axes

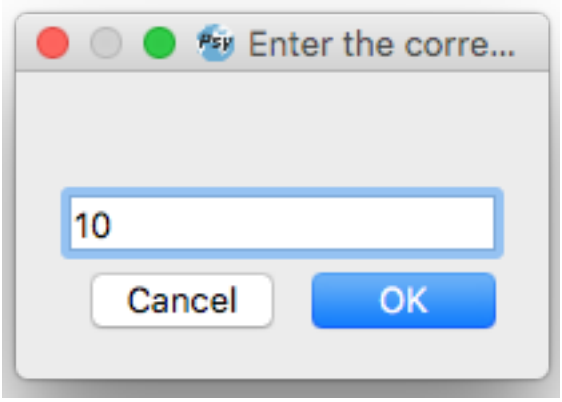
For your final product, straditize needs to know about how the data in the diagram is scaled, such that we can transform the final sample information from the pixel coordinates into the units of the diagram (e.g. percent for pollen data or Kelvin for temperature, etc.).

To translate the x-axis information,

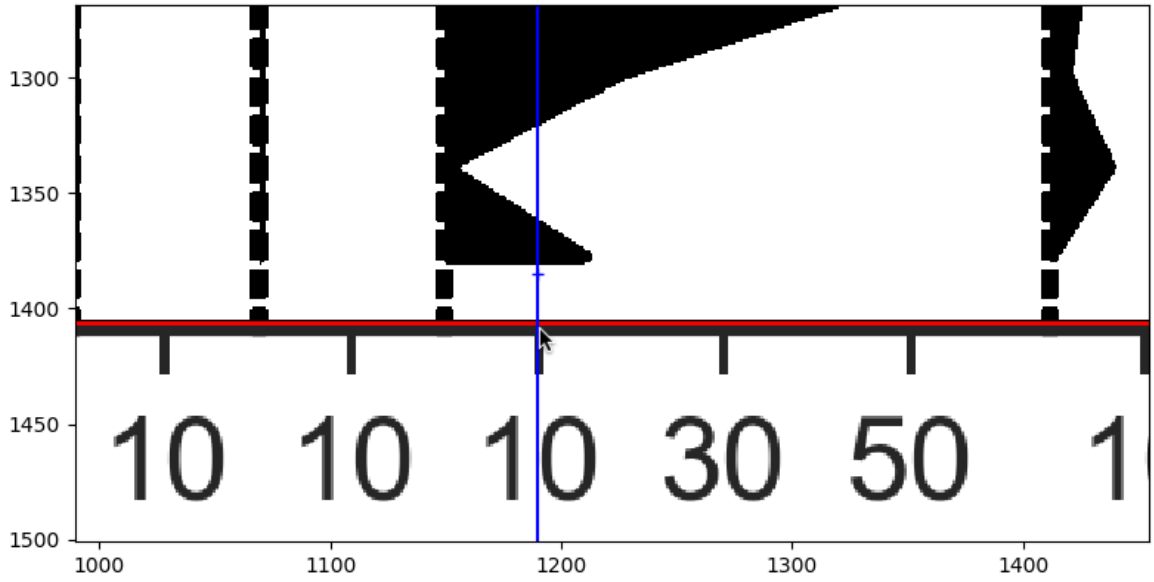
1. Expand the *Axes translations* tab in the digitization control
2. Click the *Insert X-axis values* button in the *Axes translations* section of the straditizer control (if not already done)
3. Shift-leftclick on the plot in one of the columns to enter the corresponding x-value.



4. A small dialog will appear where you should enter the x-value to use (in this case, `10`)

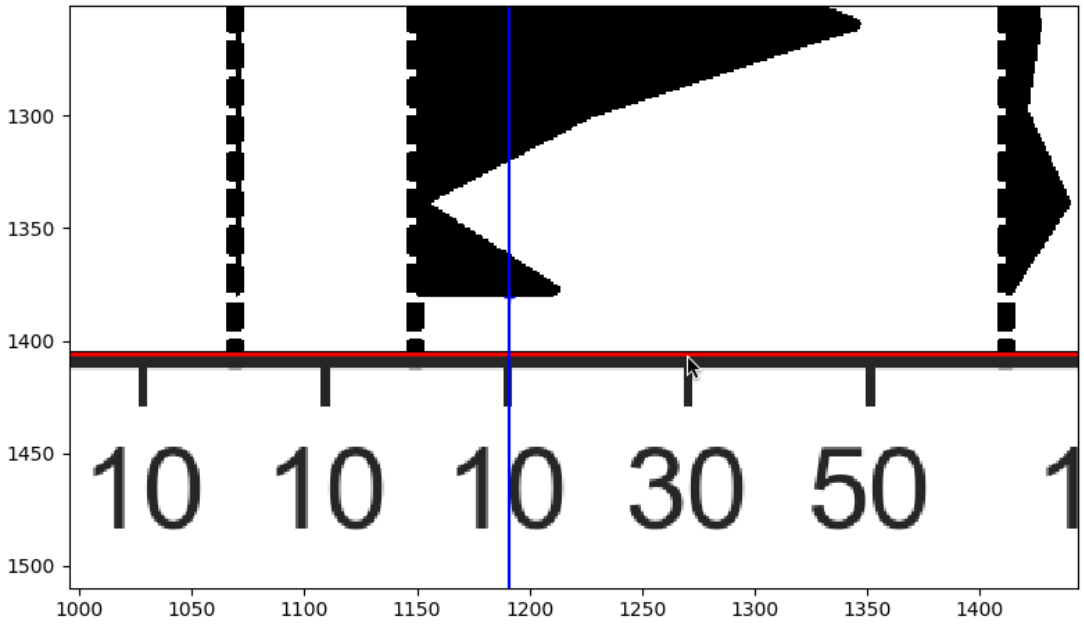


5. After hitting the *Ok* button, you will see a mark on the plot (blue line). You can select the mark via leftclick and drag it to a different location or you can delete it via rightclick.

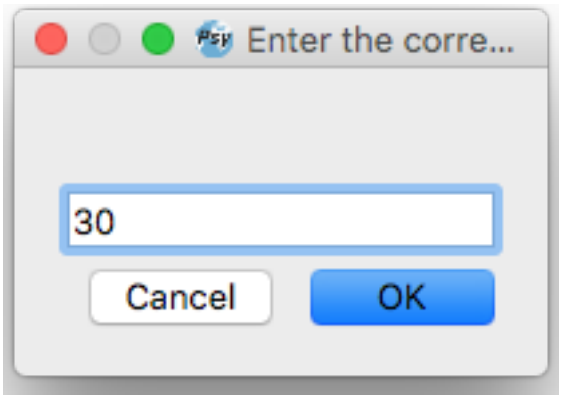


6. now repeat steps 3-5 on a second point in the same column

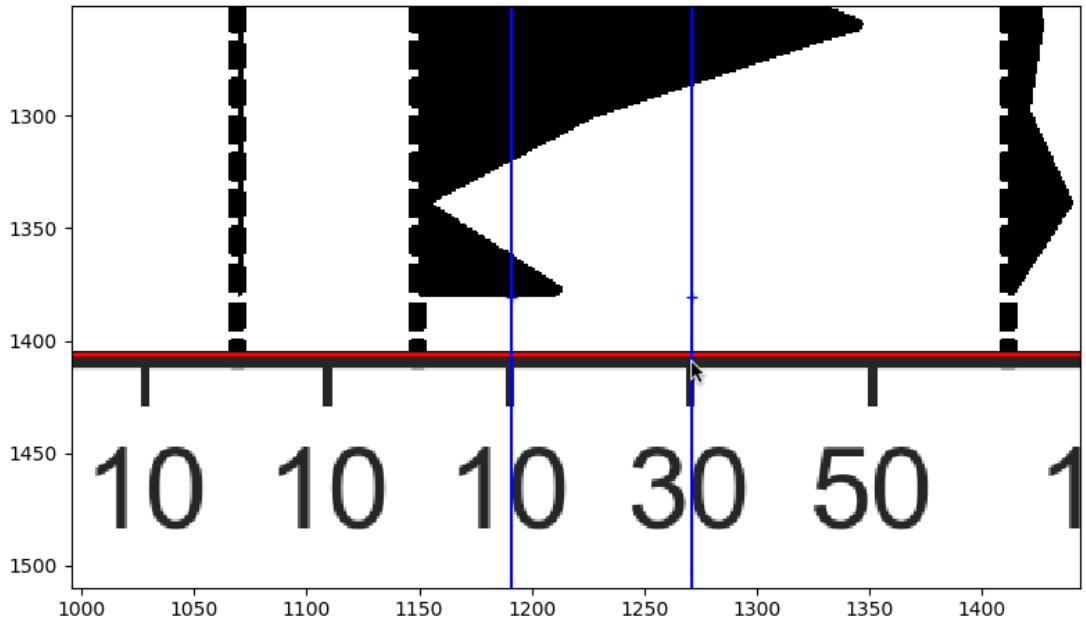
   - Select another point



   - Enter the corresponding value (here `30`)



- A new mark is created that you can modify



7. Click the *Apply* button at the bottom of the straditizer control when you are done.

Note: If you have different units or different scalings in the diagram, create *column specific readers* and translate the x-axis separately for each reader/column.

Note: If you drag a mark and hold the `Shift` button while releasing the mouse button, the dialog in point 3 from above will not pop up.

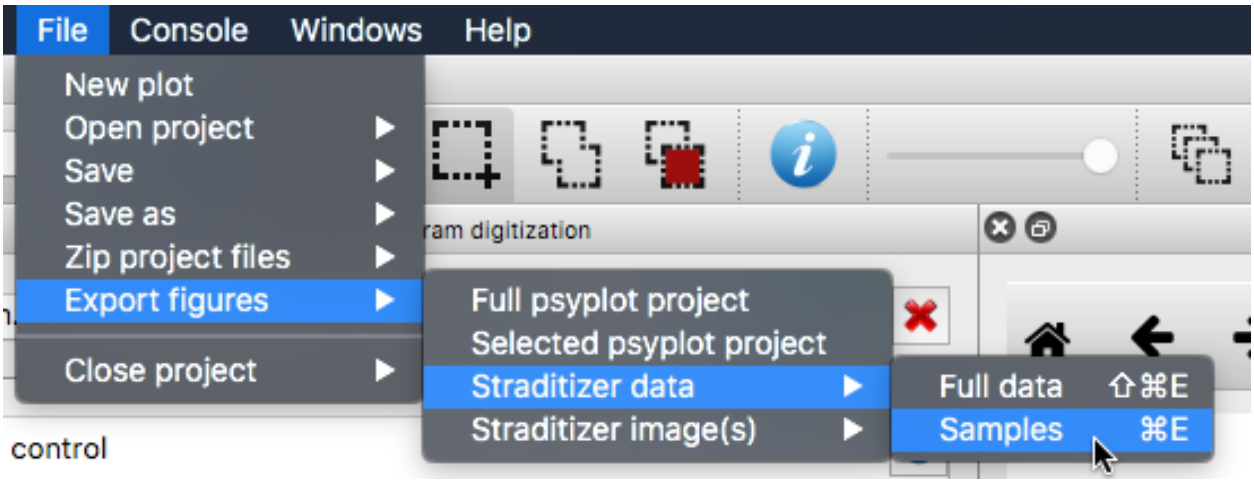
Export your results

Once you found the samples and added the y-axis and x-axis transformations, you can export them to a comma-separated (CSV) or an Excel file.

Just use the export menu via

File → *Export figures* → *Straditizer data* → *Samples*

or hit `Control-E` (`Command-E` on MacOS)



If you want to export the full digitized data, i.e. not only the samples, use

File → *Export figures* → *Straditizer data* → *Full data*

or hit `Shift-Control-E` (`Shift-Command-E` on MacOS)

1.6.3 Straditize helpers

Straditize has several generic tools to help you with various aspects.

The selection toolbar

The tools in this toolbar provides you the possibility to select features in the diagram. By default, the selected features will then be removed when you click the *Apply* or : guilabel:*Remove* button at the bottom of the straditizer window. The processing of the selection however varies depending on what task you are doing at the moment (for exaggerations, the selected features are for example marked as exaggerations).

The selection toolbar can *select from different sources*, provides *differenti mouse tools*, can be used in *different modes* provides *additional automatic tools for the selection*

Choosing the selection source

Using the combo box, you can also choose, where the features should be selected.

Straditizer If you choose the *Straditizer*, features will only be selected in the full image and the diagram part is untouched.

Reader If you choose the *Reader*, features will only be selected in the diagram image and the original image will be kept untouched

Reader - Greyscale The same as *Reader*, but whether too parts are connected or not is based on their color, not only on the binary image (see the tool below).

Selection tools

Too facilitate the selection for you, we implemented several tools to select features in the image:

select a rectangle or point  With this tool you can select everything within a rectangle or a single point

select a polygon  This tool can be accessed through the context menu of the  tool (click and hold this button to open the menu). If activated and you click on the image, hold the mouse button and drag it around the features you want to select. Everything that is in the shape you draw while pressing the mouse button will be selected.

select based on connectivity or functionality  This tool selects entire features in the image. The exact behaviour depends on what you are doing at the moment. If you hold and drag the mouse, you can select all features within a rectangle

In most cases, the tool just selects a feature based on the connectivity. If the selection source is the *Straditizer* or *Reader - Greyscale*, this connectivity is based on the greyscale image. For the *Reader* selection source, this is based on the binary image.

select based on color  This tool can be accessed through the context menu of the  tool (click and hold this button to open the menu). If activated and you click on the image, all connected cells that have the same color are selected. You can also relax this a bit, such that all colors that are close to the selected color will be selected. Furthermore, you can choose the select the colors in the *whole plot*, i.e. all pixels that have the same (or similar) color as the selected cell will be selected.

select the entire pixel row  This tool also can be accessed through the context menu of the  tool (click and hold this button to open the menu). If you click on the image, all selectable features on this horizontal level (i.e. on this pixel row in the image) are selected.

select the entire pixel column  The same as the  tool but for the vertical columns.

Selection mode

What exatly happens when you use the *selection tools desribed above* depends on the selection mode that you are in. This can be one of

new selection  When you select something, it will create a new selection

add to selection  Any new selection will be added the current selection

remove from selection  Any new selection will be removed from the current selection

Automatic tools

The selection toolbar also provides you some automatic tools to facilitate the selection for you

select all  Everything that can be selected will be selected

expand selection  Every selected pixel will be expanded based on it's connectivity (see the  tool above)

invert selection  Everything that is selected will be unselected and everything that was not previously selected will be selected

clear selection  Everything will be unselected

select everything to the right  For each selected pixel in a column of the diagram part, we also select everything that is to the right of this pixel

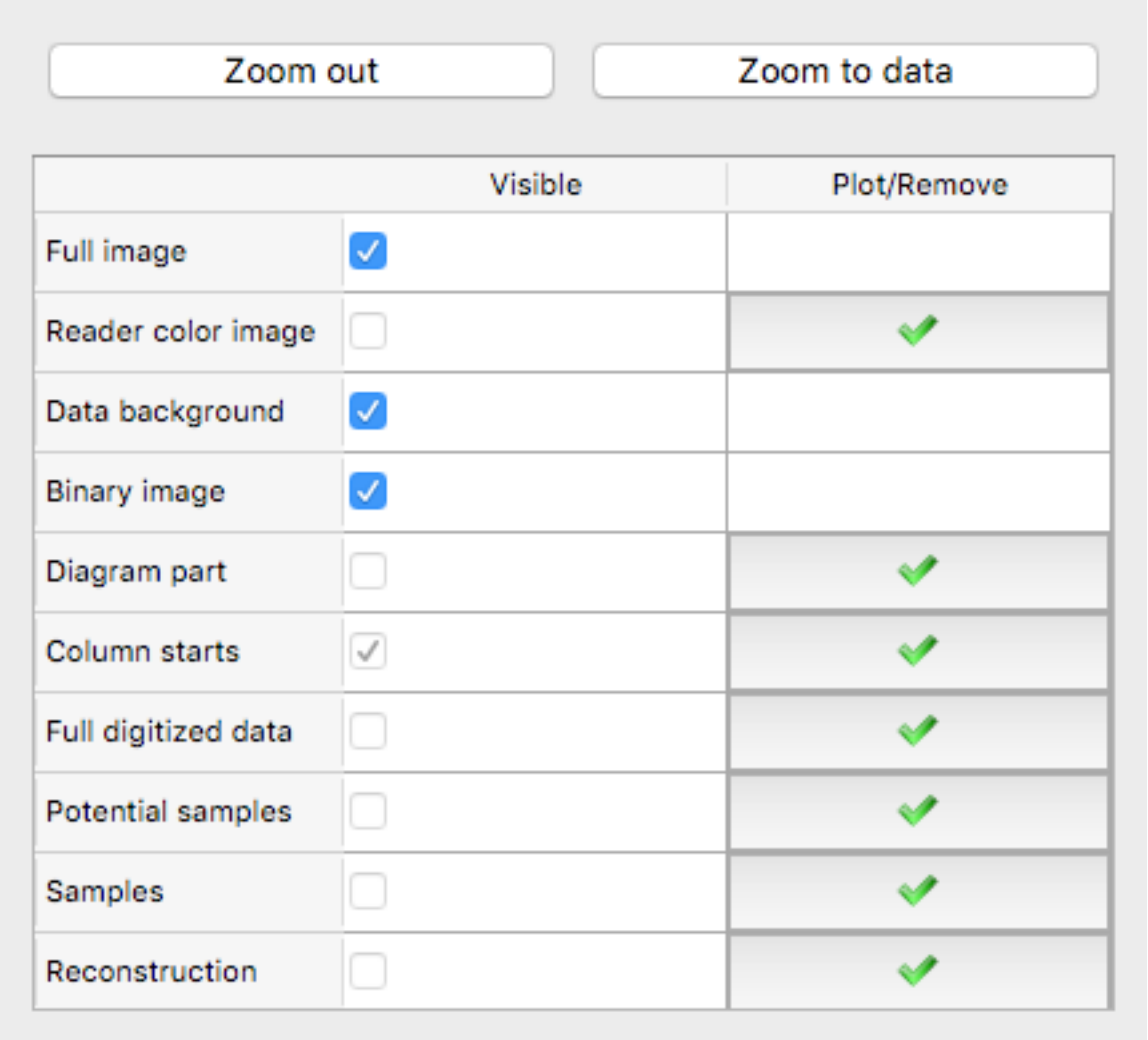
select based on a template  This method uses the `skimage.feature.match_template()` function to find a template within the current selection. This will then be selected or removed from the current selection based on the *current mode*.

Visualize your progress with the plot control

To explore and validate your digitization, straditize implements multiple ways to visualize the results. You can show and remove these static visualizations using the items in the plot control.

Here you can hide the several plot objects using the *Visible* checkbox or you can plot the objects by clicking the  button or remove it by clicking the  button.

Note: Please note, that these are static plots. So if you change your data, e.g. through modifying the column starts, you have to remove () and plot () the *Column starts* item to visualize the changes.



Some of the plotting features only visualize the currently selected columns. So if you have *column-specific readers* or *exaggerations*, these options hide and show the plots for the currently selected columns only.

Full image: The full image is the diagram, that you are digitizing

Reader color image: This is the diagram part of the currently selected columns in color.

Data background: This is just a white layer that lies over the *Full image* in the diagram part. If you want to see the diagram part in the *Full image*, you have to hide this layer

Binary image: This is the image that is used to digitize the diagram part. As with the *Reader color image*, this option shows/hides the binary image for the currently selected columns.

Diagram part: This draws a red rectangle around the *diagram part* (see the *Basics and Terminology*)

Column starts: Red lines showing you, where each column starts of out of the currently selected columns starts

Full digitized data: After having hit the *Digitize* button, you can visualize the full digitized data for the currently selected columns.

Potential samples: For each column, we identify rough locations where samples may lie (see *Automatic samples identification*). This plot option lets you visualize these.

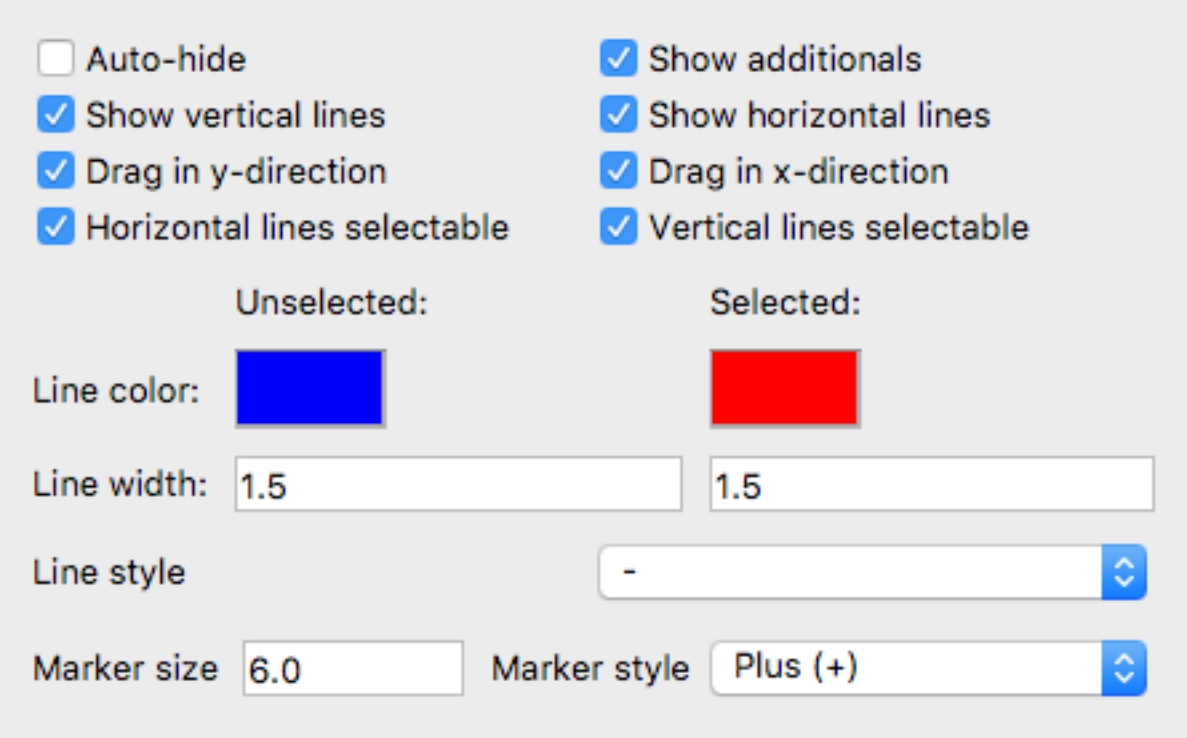
Samples: This option draws horizontal lines to show, the sample locations

Reconstruction: This plot option draws one line for each column, as the *Full digitized data* does, but it uses only the x- and y-values at the sample locations.

Configuring the appearance of markers

In the digitization, you will work with markers to select parts of your image. These can be the marks to *select the diagram part*, to *separate the columns* or to *edit your samples*.

You can modify the appearance of the marks using the *Marker control* section in the straditization control. This helps you to better visualize and select different parts in the diagram.



The functionalities are

Auto-hide Hide the lines of the marks, when they are not selected

Show additionals When *editing the samples*, for example, you will also see the rough locations of the samples for each column (little grey crosses). With the *Show additionals* setting, you can hide or show them

Show vertical lines* and *Show horizontal lines If your marks are crosses, e.g. when you are *selecting the diagram part*, you can hide and show the vertical or horizontal lines with these settings

Drag in y-direction* and *Drag in x-direction If unchecked, a mark cannot be moved along the corresponding axes. You can use this option, if you want to change the y-value of a mark without changing the x-value (or the other way around)

Horizontal lines selectable* and *Vertical lines selectable If you uncheck one of these, you can prevent selecting a mark from the horizontal or vertical position. By default, you can select and move/drag a mark, by clicking on either of the lines (horizontal or vertical). But, for example, when editing the samples, many marks may lie on the same vertical position. Therefore it can be hard to move the mark of interest if you're selecting the vertical line of the mark.

Line color This changes the color of all marks.

Line width This changes the line width of the marks. Sometimes it is better, if the linewidth increases when you are selecting a line

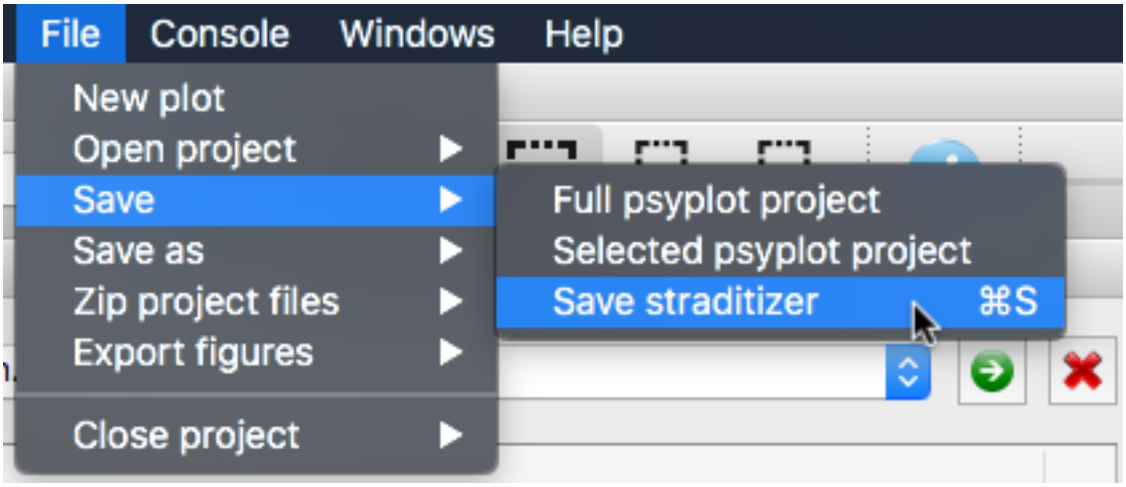
Line style You can change the style of the lines, to a dotted line, dashed line, etc.

Marker size* and *Marker style The location of the marks, i.e. where vertical and horizontal lines intersect, e.g. on corner of the diagram when *selecting the diagram part*, has a marker associated with it. It shows you, where the corner of your diagram is, where the sample value is, etc.. To make this more visible, you can change the size and the appearance of the mark.

Saving and loading your project

Saving a project

You can always save your current state to your hard-disc and continue working at it at a later point. To save a project, click *File* → *Save* → *Save straditizer*



or hit `Control-S` (`Command-S` on MacOS).

This will save your project, or, if you never saved it before or used the *Save As* option, it will open a file dialog to let you select the target location.

The default and recommended way is to save it is a netCDF file (ending with `.nc` or `.nc4`). This is a common self-describing, machine-independent data format that supports the creation, access, and sharing of array-oriented scientific

data. We recommend this format, because it works independent of the python libraries that you are using and let's you easily share the project with others or copy it to another machine.

The other possibility is to save it as a pickle (`.pkl`) file which uses the built-in `pickle` module to save the straditizer to a file. The reason, why we generally do not recomment this, is because the saved file might not work anymore if you upgrade your libraries (e.g. `numpy`, `pandas` or `pillow`).

Loading a project

Reloading the project works the same way as you open a new image file (see *Load a diagram*). Just go to *File* → *Open project* → *Open straditizer* → *Project or image* and choose the project file to load (or use the  button at the top of the straditizer control). If the chosen file ends with `.nc`, `.nc4` or `.pkl`, straditize will recreate the given project.

1.7 API Reference

Digitizing stratigraphic diagrams with straditize

The core of straditize is the *Straditizer* class and the reader for the diagram part, the *DataReader*. Both classes work completely independent of the graphical user interace.

The graphical user interface is implemented as a plugin into the Graphical User Interface for the psyplot package package and is implemented in the *straditize.widgets* subpackage.

Authors

The code and GUI of straditize was developed by Philipp S. Sommer at the Institute of Earth System Dynamics (IDYST) at the University of Lausanne as part of the SNF funded HORNET Project (200021_169598).

The other contributors are Basil A. S. Davis, Manuel Chevalier and Dilan Rech who made significant contributions to the layout, workflow, beta tests and and reviewing of the software.

Disclaimer

1.7.1 Subpackages

straditize.widgets package

The straditizer widgets

This module contains widgets to digitize the straditizer diagrams through a GUI

Disclaimer

This program is free software: you can redistribute it and/or modify it under the terms of the GNU General Public License as published by the Free Software Foundation, either version 3 of the License, or (at your option) any later version.

This program is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU General Public License for more details.

You should have received a copy of the GNU General Public License along with this program. If not, see <https://www.gnu.org/licenses/>.

Classes

EnableButton	A *QPushButton* that emits a signal when enabled
InfoButton(parent[, fname, rst, name])	A button to display help informations in the help explorer
StraditizerControlBase	A base class for the straditizer widget
StraditizerWidgets(*args, **kwargs)	A widget that contains widgets to control the straditization in a GUI

Functions

get_doc_file(fname)	Return the path to a documentation file
get_icon(fname)	Return the path of an icon
get_straditizer_widgets([mainwindow])	Get the *StraditizerWidgets* from the psyplot GUI mainwindow
read_doc_file(fname)	Return the content of a rst documentation file

class straditize.widgets.**EnableButton**

Bases: PyQt5.QtWidgets.QPushButton

A *QPushButton* that emits a signal when enabled **Attributes**

enabled(*args, **kwargs)	A signal that is emitted with a boolean whether if the button is

Methods

setEnabled(b)	Reimplemented to emit the `enabled` signal

enabled(**args*, ***kwargs*)

A signal that is emitted with a boolean whether if the button is enabled or disabled

setEnabled(*b*)

Reimplemented to emit the *enabled* signal

class straditize.widgets.**InfoButton**(*parent*, *fname=None*, *rst=None*, *name=None*)

Bases: PyQt5.QtWidgets.QToolButton

A button to display help informations in the help explorer

Parameters

- **parent** (*QWidget*) – The parent widget
- **fname** (*str*) – The name of the rst file. If None, specify the *rst* directly

- **rst** (*str*) – The restructured text to render when this button is clicked. If None, the *fname* has to be provided
- **name** (*str*) – The name to use for the document in the help_explorer

Methods

show_docs()	Show the docs

show_docs()
Show the docs

Shows the docs in the in the help_explorer

class straditize.widgets.**StraditizerControlBase**
Bases: object

A base class for the straditizer widget **Methods**

add_info_button(child[, fname, rst, name, ...])	Add an info button to the given QTreeWidgetItem
connect2apply(*funcs)	Connect the functions to the apply_button
connect2cancel(*funcs)	Connect the functions to the cancel_button
enable_or_disable_widgets(b)	Enable or disable the widgets in this control
init_straditizercontrol(straditizer_widgets)	Initialize the straditizer control widget
refresh()	Refresh from the straditizer
setup_children(item)	Setup the children for this control
should_be_enabled(w)	Check if a widget should be enabled

Attributes

apply_button	The apply button of the straditizer_widgets
cancel_button	The cancel button of the straditizer_widgets
help_explorer	The psyplot_gui.help_explorer.HelpExplorer of the
straditizer	The current straditizer from the straditizer_widgets
straditizer_widgets	The StraditizierWidgets control
widgets2disable	A list of widgets to disable or enable if the apply_button is enabled

add_info_button(*child*, *fname=None*, *rst=None*, *name=None*, *connections=[]*)
Add an info button to the given QTreeWidgetItem

Parameters **child** (*QTreeWidgetItem*) – The item to which to add the infobutton

property apply_button
The apply button of the *straditizer_widgets*

property cancel_button
The cancel button of the *straditizer_widgets*

connect2apply(**funcs*)
Connect the functions to the *apply_button*

Parameters ***funcs** – The callables that should be connected to the *apply_button*

connect2cancel(**funcs*)
Connect the functions to the *cancel_button*

Parameters ***funcs** – The callables that should be connected to the *cancel_button*

enable_or_disable_widgets(*b*)
Enable or disable the widgets in this control

This method enables or disables the *widgets2disable* if the *should_be_enabled()* method evaluates to True

Parameters **b** (*bool*) – If True, enable the widgets, if False, disable them

property help_explorer
The psyplot_gui.help_explorer.HelpExplorer of the psyplot_gui.main.mainwindow

init_straditizercontrol(*straditizer_widgets*, *item=None*)
Initialize the straditizer control widget

This method should be called by every subclass when initializing. It sets the *straditizer_widgets*, connects the *enable_or_disable_widgets()* method and adds a new item to the *StraditizerWidgets.tree*

Parameters

- **straditizer_widgets** (StraditizerWidgets) – The main widget for the straditizer GUI
- **item** (*QTreeWidgetItem*) – The parent item in the *StraditizerWidgets.tree*. If given, the *setup_children()* is called with this item

refresh()
Refresh from the straditizer

setup_children(*item*)
Setup the children for this control

This method is called to setup the children in the *StraditizerWidgets.tree*. By default, it just creates a child QTreeWidgetItem and sets this control as it's widget

Parameters **item** (*QTreeWidgetItem*) – The top level item in the *StraditizerWidgets.tree*

should_be_enabled(*w*)
Check if a widget should be enabled

This function checks if a given widget *w* from the *widgets2disable* attribute should be enabled or not

Parameters **w** (*QWidget*) – The widget to check

Returns True, if the widget should be enabled

Return type bool

property straditizer
The current straditizer from the *straditizer_widgets*

straditizer_widgets = None
The StraditizierWidgets control

widgets2disable = []
A list of widgets to disable or enable if the apply_button is enabled

class straditize.widgets.**StraditizerWidgets**(**args*, ***kwargs*)

Bases: PyQt5.QtWidgets.QWidget, psyplot_gui.common.DockMixin

A widget that contains widgets to control the straditization in a GUI

This widget is the basis of the straditize GUI and implemented as a plugin into the psyplot gui. The open straditizers are handled in the `_straditizer` attribute.

The central parts of this widget are

- The combobox to manage the open straditizers
- The QTreeWidget in the *tree* attribute that contains all the controls to interface the straditizer
- the tutorial area
- the *Apply* and *Cancel* button

Methods

add_info_button(child[, fname, rst, name, ...])	Add an infobutton to the `tree` widget
add_straditizer(stradi)	Add a straditizer to the list of open straditizers
autosave()	Autosave the current straditizer
close_all_straditizers()	Close all straditizers
close_straditizer()	Close the current straditizer
create_straditizer_from_args(fnames[, ...])	Create a straditizer from the given file name
disable_apply_button()	Method that is called when the `cancel_button` is clicked
edit_attrs()	Edit the attributes of the current straditizer
get_attr(stradi, attr)	
raise_figures()	Raise the figures of the current straditizer in the GUI
refresh()	Refresh from the straditizer
reload_autosaved()	Reload the autosaved straditizer and close the old one
reset_control()	Reset the GUI of straditize
set_current_stradi(i)	Set the i-th straditizer to the current one
show_or_hide_toolbar()	Show or hide the toolbar depending on the visibility of this widget
start_tutorial(state[, tutorial_cls])	Start or stop the tutorial
switch_to_straditizer_layout()	Switch to the straditizer layout
to_dock(main, *args, **kwargs)	

Attributes

always_yes	Boolean that is True if all dialogs should be answered with *Yes*
apply_button	The apply button
attrs_button	The button to edit the straditizer attributes
autosaved	Auto-saved straditizers
axes_translations	The *straditize.widgets.axes_translations.AxesTranslations* to
btn_close_stradi	A button to close the current straditizer

Continued on next page

Table 9 – continued from previous page

btn_open_stradi	A button to open a new straditizer
btn_reload_autosaved	A button to reload the last autosaved state
cancel_button	The cancel button
colnames_manager	The *straditize.widgets.colnames.ColumnNamesManager* to interface
digitizer	The *straditize.widgets.data.DigitizingControl* to interface
hidden	bool(x) -> bool
image_rescaler	The *straditize.widgets.image_correction.ImageRescaler* class to
image_rotator	The *straditize.widgets.image_correction.ImageRotator* class to
info_button	An *InfoButton* to display the docs
marker_control	The *straditize.widgets.marker_control.MarkerControl* to modify
open_external(*args, **kwargs)	
plot_control	The *straditize.widgets.plots.PlotControl* to display additional
progress_widget	The *straditize.widgets.progress_widget.ProgressWidget* to
selection_toolbar	The *straditize.widgets.selection_toolbar.SelectionToolbar* to
stradi_combo	A QComboBox to select the current straditizer
straditizer	The *straditize.straditizer.Straditizer* instance
title	str(object='') -> str
tree	The QTreeWidget that contains the different widgets for the digitization
tutorial	The `straditize.widgets.tutorial.Tutorial` class
tutorial_button	The button to start a tutorial

add_info_button(*child*, *fname=None*, *rst=None*, *name=None*, *connections=[]*)
Add an infobutton to the *tree* widget

Parameters

- **child** (*QTreeWidgetItem*) – The item to which to add the infobutton
- **connections** (*list of QPushButtons*) – Buttons that should be clicked when the info button is clicked

add_straditizer(*stradi*)
Add a straditizer to the list of open straditizers

always_yes = False
Boolean that is True if all dialogs should be answered with *Yes*

apply_button = None
The apply button

attrs_button = None
The button to edit the straditizer attributes

autosave()
Autosave the current straditizer

autosaved = []
Auto-saved straditizers

axes_translations = None
The *straditize.widgets.axes_translations.AxesTranslations* to handle the y- and x-axis conversions

btn_close_stradi = None
A button to close the current straditizer

btn_open_stradi = None
A button to open a new straditizer

btn_reload_autosaved = None
A button to reload the last autosaved state

cancel_button = None
The cancel button

close_all_straditizers()
Close all straditizers

close_straditizer()
Close the current straditizer

colnames_manager = None
The *straditize.widgets.colnames.ColumnNamesManager* to interface the :straditize.straditizer.Straditizer.colnames_reader‘

create_straditizer_from_args(*fnames*, *project=None*, *xlim=None*, *ylim=None*, *full=False*, *reader_type='area'*)
Create a straditizer from the given file name

This method is called when the `psyplot_gui.main.mainwindow` receives a ‘straditize’ callback

digitizer = None
The *straditize.widgets.data.DigitizingControl* to interface the :straditize.straditizer.Straditizer.data_reader‘

disable_apply_button()
Method that is called when the *cancel_button* is clicked

dock_position = 1

edit_attrs()
Edit the attributes of the current straditizer

This creates a new dataframe editor to edit the *straditize.straditizer.Straditizer.attrs* meta informations

get_attr(*stradi*, *attr*)

hidden = True

image_rescaler = None
The *straditize.widgets.image_correction.ImageRescaler* class to rescale the image

image_rotator = None
The *straditize.widgets.image_correction.ImageRotator* class to rotate the image

info_button = None
An *InfoButton* to display the docs

marker_control = None
The *straditize.widgets.marker_control.MarkerControl* to modify the appearance of the `marks` of the current straditizer

open_external(**args*, ***kwargs*)

plot_control = None
The *straditize.widgets.plots.PlotControl* to display additional information on the diagram

progress_widget = None
The *straditize.widgets.progress_widget.ProgressWidget* to display the progress of the straditization

raise_figures()
Raise the figures of the current straditizer in the GUI

refresh()
Refresh from the straditizer

reload_autosaved()
Reload the autosaved straditizer and close the old one

reset_control()
Reset the GUI of straditize

selection_toolbar = None
The *straditize.widgets.selection_toolbar.SelectionToolbar* to select features in the stratigraphic diagram

set_current_stradi(*i*)
Set the i-th straditizer to the current one

show_or_hide_toolbar()
Show or hide the toolbar depending on the visibility of this widget

start_tutorial(*state*, *tutorial_cls=None*)
Start or stop the tutorial

Parameters

- **state** (*bool*) – If False, the tutorial is stopped. Otherwise it is started
- **tutorial_cls** (straditize.widgets.tutorial.beginner.Tutorial) – The tutorial class to use. If None, it will be asked in a QInputDialog

stradi_combo = None
A QComboBox to select the current straditizer

straditizer = None
The *straditize.straditizer.Straditizer* instance

switch_to_straditizer_layout()
Switch to the straditizer layout

This method makes this widget visible and stacks it with the psyplot content widget

title = 'Stratigraphic diagram digitization'

to_dock(*main*, **args*, ***kwargs*)

tree = None
The QTreeWidget that contains the different widgets for the digitization

tutorial = None
The `straditize.widgets.tutorial.Tutorial` class

tutorial_button = None
The button to start a tutorial

window_layout_action = None

`straditize.widgets.`**get_doc_file**(*fname*)
Return the path to a documentation file

`straditize.widgets.`**get_icon**(*fname*)
Return the path of an icon

`straditize.widgets.`**get_straditizer_widgets**(*mainwindow=None*)
Get the *StraditizerWidgets* from the psyplot GUI mainwindow

Parameters **psyplot_gui.main.MainWindow** – The mainwindow to use. If None, the `psyplot_gui.main.mainwindow` is used.

Returns The straditizer widgets of the given *mainwindow*

Return type *StraditizerWidgets*

`straditize.widgets.`**read_doc_file**(*fname*)
Return the content of a rst documentation file

Subpackages

straditize.widgets.tutorial package

Tutorials for straditize

The *beginner* tutorial serves as a first look into the software. The *hoya_del_castillo* tutorial on the other hand is a true pollen diagram with more than 20 taxa. Both tutorials can be started from the GUI through the *Tutorial* button.

Disclaimer

Submodules

straditize.widgets.tutorial.beginner module

The tutorial of straditize

This module contains a guided tour to get started with straditize

Disclaimer

Classes

CleanImagePage(filename, tutorial)	Tutorial page to clean the diagram part
ColumnNames(filename, tutorial)	The page for recognizing column names
ControlIntro(filename, tutorial)	Tutorial page for the control
CreateReader(filename, tutorial)	The page for creating the reader
DigitizePage(filename, tutorial)	The page for digitizing the diagram
FinishPage(filename, tutorial)	The last page of the tutorial
LoadImage(filename, tutorial)	TutorialPage for loading the straditizer image
SamplesPage(filename, tutorial)	The page for finding and editing the samples
SelectDataPart(filename, tutorial)	TutorialPage for selecting the data part
SeparateColumns(filename, tutorial)	The page for separating the columns
TranslateXAxis(filename, tutorial)	The tutorial page for translating the y-axis
TranslateYAxis(filename, tutorial)	The tutorial page for translating the y-axis
Tutorial(straditizer_widgets)	A tutorial for digitizing an area diagram
TutorialDocs(*args, **kwargs)	A documentation viewer for the tutorial docs
TutorialNavigation(nsteps, validate, *args, ...)	A widget for navigating through the tutorial.
TutorialPage(filename, tutorial)	A base class for the tutorial pages

class straditize.widgets.tutorial.beginner.**CleanImagePage**(*filename*, *tutorial*)

Bases: *straditize.widgets.tutorial.beginner.TutorialPage*

Tutorial page to clean the diagram part

Parameters

- **filename** (*str*) – The basename (without ending) of the RST file corresponding to this tutorial page
- **tutorial** (Tutorial) – The tutorial instance

Methods

activate()	Method that is called, when the page is activated
clicked_btn_remove_xaxes()	
clicked_btn_remove_yaxes()	

Continued on next page

Table 11 – continued from previous page

deactivate()	Method that is called, when the page is deactivated
hint()	A method that should display a hint to the user
icon_to_bytes(icon)	
skip()	Skip the steps in this page

Attributes

btn_remove_xaxes_clicked	bool(x) -> bool
btn_remove_yaxes_clicked	bool(x) -> bool
is_finished	Boolean that is True, if the steps are all finished

activate()
Method that is called, when the page is activated

btn_remove_xaxes_clicked = False

btn_remove_yaxes_clicked = False

clicked_btn_remove_xaxes()

clicked_btn_remove_yaxes()

deactivate()
Method that is called, when the page is deactivated

hint()
A method that should display a hint to the user

icon_to_bytes(*icon*)

property is_finished
Boolean that is True, if the steps are all finished

skip()
Skip the steps in this page

class straditize.widgets.tutorial.beginner.**ColumnNames**(*filename*, *tutorial*)
Bases: *straditize.widgets.tutorial.beginner.TutorialPage*

The page for recognizing column names

Parameters

- **filename** (*str*) – The basename (without ending) of the RST file corresponding to this tutorial page
- **tutorial** (Tutorial) – The tutorial instance

Methods

activate()	Method that is called, when the page is activated
clicked_select_names_button()	
deactivate()	Method that is called, when the page is deactivated
hint()	A method that should display a hint to the user
hint_for_start_editing()	
hint_for_wrong_name(col, curr, ref)	Display a hint if a name is not correctly set
skip()	Skip the steps in this page

Attributes

column_names	The column names in the diagram
is_finished	Boolean that is True, if the steps are all finished
select_names_button_clicked	bool(x) -> bool

activate()
Method that is called, when the page is activated

clicked_select_names_button()

column_names = ['Pinus', 'Juniperus', 'Quercus ilex-type', 'Chenopodiaceae']
The column names in the diagram

deactivate()
Method that is called, when the page is deactivated

hint()
A method that should display a hint to the user

hint_for_start_editing()

hint_for_wrong_name(*col*, *curr*, *ref*)
Display a hint if a name is not correctly set

property is_finished
Boolean that is True, if the steps are all finished

select_names_button_clicked = False

skip()
Skip the steps in this page

class straditize.widgets.tutorial.beginner.**ControlIntro**(*filename*, *tutorial*)
Bases: *straditize.widgets.tutorial.beginner.TutorialPage*

Tutorial page for the control

Parameters

- **filename** (*str*) – The basename (without ending) of the RST file corresponding to this tutorial page
- **tutorial** (Tutorial) – The tutorial instance

Methods

activate()	Method that is called, when the page is activated

activate()
Method that is called, when the page is activated

class straditize.widgets.tutorial.beginner.**CreateReader**(*filename*, *tutorial*)
Bases: *straditize.widgets.tutorial.beginner.TutorialPage*

The page for creating the reader

Parameters

- **filename** (*str*) – The basename (without ending) of the RST file corresponding to this tutorial page
- **tutorial** (Tutorial) – The tutorial instance

Methods

hint()	A method that should display a hint to the user
skip()	Skip the steps in this page

Attributes

is_finished	Boolean that is True, if the steps are all finished

hint()
A method that should display a hint to the user

property is_finished
Boolean that is True, if the steps are all finished

skip()
Skip the steps in this page

class straditize.widgets.tutorial.beginner.**DigitizePage**(*filename*, *tutorial*)
Bases: *straditize.widgets.tutorial.beginner.TutorialPage*

The page for digitizing the diagram

Parameters

- **filename** (*str*) – The basename (without ending) of the RST file corresponding to this tutorial page
- **tutorial** (Tutorial) – The tutorial instance

Methods

hint()	A method that should display a hint to the user
skip()	Skip the steps in this page

Attributes

is_finished	Boolean that is True, if the steps are all finished

hint()
A method that should display a hint to the user

property is_finished
Boolean that is True, if the steps are all finished

skip()
Skip the steps in this page

class straditize.widgets.tutorial.beginner.**FinishPage**(*filename*, *tutorial*)
Bases: *straditize.widgets.tutorial.beginner.TutorialPage*

The last page of the tutorial

Parameters

- **filename** (*str*) – The basename (without ending) of the RST file corresponding to this tutorial page
- **tutorial** (Tutorial) – The tutorial instance

Methods

show()	Reimplemented to release the help explorer lock

show()
Reimplemented to release the help explorer lock

class straditize.widgets.tutorial.beginner.**LoadImage**(*filename*, *tutorial*)
Bases: *straditize.widgets.tutorial.beginner.TutorialPage*

TutorialPage for loading the straditizer image

Parameters

- **filename** (*str*) – The basename (without ending) of the RST file corresponding to this tutorial page
- **tutorial** (Tutorial) – The tutorial instance

Methods

activate()	Method that is called, when the page is activated
deactivate()	Method that is called, when the page is deactivated
hint()	A method that should display a hint to the user
skip()	Skip the steps in this page

Attributes

is_finished	Boolean that is True, if the steps are all finished

activate()
Method that is called, when the page is activated

deactivate()
Method that is called, when the page is deactivated

hint()
A method that should display a hint to the user

property is_finished
Boolean that is True, if the steps are all finished

skip()
Skip the steps in this page

class straditize.widgets.tutorial.beginner.**SamplesPage**(*filename*, *tutorial*)
Bases: *straditize.widgets.tutorial.beginner.TutorialPage*

The page for finding and editing the samples

Parameters

- **filename** (*str*) – The basename (without ending) of the RST file corresponding to this tutorial page
- **tutorial** (Tutorial) – The tutorial instance

Methods

activate()	Method that is called, when the page is activated

Continued on next page

Table 23 – continued from previous page

clicked_correct_button()	
deactivate()	Method that is called, when the page is deactivated
hint()	A method that should display a hint to the user
skip()	Skip the steps in this page

Attributes

correct_button_clicked	bool(x) -> bool
is_finished	Boolean that is True, if the steps are all finished

activate()
Method that is called, when the page is activated

clicked_correct_button()

correct_button_clicked = False

deactivate()
Method that is called, when the page is deactivated

hint()
A method that should display a hint to the user

property is_finished
Boolean that is True, if the steps are all finished

skip()
Skip the steps in this page

class straditize.widgets.tutorial.beginner.**SelectDataPart**(*filename*, *tutorial*)
Bases: *straditize.widgets.tutorial.beginner.TutorialPage*

TutorialPage for selecting the data part

Parameters

- **filename** (*str*) – The basename (without ending) of the RST file corresponding to this tutorial page
- **tutorial** (Tutorial) – The tutorial instance

Methods

activate()	Method that is called, when the page is activated
check_mark(mark)	
clicked_correct_button()	
deactivate()	Method that is called, when the page is deactivated
display_reference_marks()	
hint()	A method that should display a hint to the user
is_valid_x(x)	
is_valid_y(y)	
skip()	Skip the steps in this page
validate_corners()	

Attributes

correct_button_clicked	bool(x) -> bool
is_finished	Boolean that is True, if the steps are all finished
marks	Built-in mutable sequence.
ref_lims	The reference x- and y- limits
remove_data_box	true if the data box should be removed at the end
valid_xlims	Valid ranges for xmin and xmax
valid_ylims	Valid ranges for ymin and ymax

activate()
Method that is called, when the page is activated

check_mark(*mark*)

clicked_correct_button()

correct_button_clicked = False

deactivate()
Method that is called, when the page is deactivated

display_reference_marks()

hint()
A method that should display a hint to the user

property is_finished
Boolean that is True, if the steps are all finished

is_valid_x(*x*)

is_valid_y(*y*)

marks = []

ref_lims = array([[258, 1803], [375, 1666]])
The reference x- and y- limits

remove_data_box = True
true if the data box should be removed at the end

skip()
Skip the steps in this page

valid_xlims = array([[221, 263], [1730, 1922]])
Valid ranges for xmin and xmax

valid_ylims = array([[346, 403], [1648, 1701]])
Valid ranges for ymin and ymax

validate_corners()

class straditize.widgets.tutorial.beginner.**SeparateColumns**(*filename*, *tutorial*)
Bases: *straditize.widgets.tutorial.beginner.TutorialPage*

The page for separating the columns

Parameters

- **filename** (*str*) – The basename (without ending) of the RST file corresponding to this tutorial page
- **tutorial** (Tutorial) – The tutorial instance

Methods

activate()	Method that is called, when the page is activated
clicked_correct_button()	
deactivate()	Method that is called, when the page is deactivated
hint()	A method that should display a hint to the user
skip()	Skip the steps in this page

Attributes

correct_button_clicked	bool(x) -> bool
is_finished	Boolean that is True, if the steps are all finished
ncols	int([x]) -> integer

activate()
Method that is called, when the page is activated

clicked_correct_button()

correct_button_clicked = False

deactivate()
Method that is called, when the page is deactivated

hint()
A method that should display a hint to the user

property is_finished
Boolean that is True, if the steps are all finished

ncols = 4

skip()
Skip the steps in this page

class straditize.widgets.tutorial.beginner.**TranslateXAxis**(*filename*, *tutorial*)
Bases: *straditize.widgets.tutorial.beginner.TutorialPage*

The tutorial page for translating the y-axis

Parameters

- **filename** (*str*) – The basename (without ending) of the RST file corresponding to this tutorial page
- **tutorial** (Tutorial) – The tutorial instance

Methods

activate()	Method that is called, when the page is activated
clicked_correct_button()	
deactivate()	Method that is called, when the page is deactivated
hint()	A method that should display a hint to the user
skip()	Skip the steps in this page

Attributes

correct_button_clicked	bool(x) -> bool
is_finished	Boolean that is True, if the steps are all finished

activate()
Method that is called, when the page is activated

clicked_correct_button()

correct_button_clicked = False

deactivate()
Method that is called, when the page is deactivated

hint()
A method that should display a hint to the user

property is_finished
Boolean that is True, if the steps are all finished

skip()
Skip the steps in this page

class straditize.widgets.tutorial.beginner.**TranslateYAxis**(*filename*, *tutorial*)
Bases: *straditize.widgets.tutorial.beginner.TutorialPage*

The tutorial page for translating the y-axis

Parameters

- **filename** (*str*) – The basename (without ending) of the RST file corresponding to this tutorial page
- **tutorial** (Tutorial) – The tutorial instance

Methods

activate()	Method that is called, when the page is activated
clicked_correct_button()	
deactivate()	Method that is called, when the page is deactivated
hint()	A method that should display a hint to the user
skip()	Skip the steps in this page

Attributes

correct_button_clicked	bool(x) -> bool
is_finished	Boolean that is True, if the steps are all finished

activate()
Method that is called, when the page is activated

clicked_correct_button()

correct_button_clicked = False

deactivate()
Method that is called, when the page is deactivated

hint()
A method that should display a hint to the user

property is_finished
Boolean that is True, if the steps are all finished

skip()
Skip the steps in this page

class straditize.widgets.tutorial.beginner.**Tutorial**(*straditizer_widgets*)
Bases: *straditize.widgets.StraditizerControlBase*, *straditize.widgets.tutorial.beginner.TutorialPage*

A tutorial for digitizing an area diagram **Methods**

close()	Close the tutorial and remove the widgets
display_hint(i)	Display the hint for a tutorial page
get_doc_files()	Get the rst files for the tutorial
goto_page(old, new)	Go to another page
refresh()	Refresh from the straditizer
setup_tutorial_pages()	Setup the `pages` attribute and initialize the tutorial pages
show()	Show the documentation of the tutorial
skip_page(i)	Skip a tutorial page
validate_page(i[, silent])	Validate a tutorial page

Attributes

current_page	The current page of the tutorial (corresponding to the
load_image_step	The number of the page that loads the diagram image (i.e.
navigation	A *TutorialNavigation* to navigate through the tutorial
pages	A list of the `TutorialPages` for this tutorial
tutorial_docs	A *TutorialDocs* to display the RST-files of the tutorial

close()
Close the tutorial and remove the widgets

property current_page
The current page of the tutorial (corresponding to the *TutorialNavigation.current_step*)

display_hint(*i*)
Display the hint for a tutorial page

Parameters **i** (*int*) – The index of the page in the *pages* attribute

See also:

TutorialPage.hint()

get_doc_files()
Get the rst files for the tutorial

Returns

- *str* – The path to the tutorial introduction file
- *list of str* – The paths of the remaining tutorial files

goto_page(*old*, *new*)
Go to another page

Parameters

- **old** (*int*) – The index of the old page in the *pages* attribute that is subject to be deactivated (see *TutorialPage.deactivate()*)
- **new** (*int*) – The index of the new page in the *pages* attribute that is subject to be activated (see *TutorialPage.activate()*)

See also:

TutorialPage.activate(), *TutorialPage.deactivate()*

property load_image_step
The number of the page that loads the diagram image (i.e. the index of the *LoadImage* instance in the *pages* attribute

navigation = None
A *TutorialNavigation* to navigate through the tutorial

pages = []
A list of the TutorialPages for this tutorial

refresh()
Refresh from the straditizer

setup_tutorial_pages()
Setup the *pages* attribute and initialize the tutorial pages

show()
Show the documentation of the tutorial

skip_page(*i*)
Skip a tutorial page

Parameters **i** (*int*) – The index of the page in the *pages* attribute

See also:

TutorialPage.skip()

tutorial_docs = None
A *TutorialDocs* to display the RST-files of the tutorial

validate_page(*i*, *silent=False*)
Validate a tutorial page

Parameters

- **i** (*int*) – The index of the page in the *pages* attribute
- **silent** (*bool*) – If True, and the page is not yet finished (see *TutorialPage.is_finished*), the hint is displayed

Returns True, if the page *is_finished*

Return type bool

class straditize.widgets.tutorial.beginner.**TutorialDocs**(**args*, ***kwargs*)
Bases: psyplot_gui.help_explorer.UrlHelp, psyplot_gui.common.DockMixin

A documentation viewer for the tutorial docs

This viewer is accessible through the *Tutorial.tutorial_docs* attribute and shows the *src_file* for the tutorial *pages* **Attributes**

title	str(object='') -> str

dock_position = 2

title = 'Straditize tutorial'

class straditize.widgets.tutorial.beginner.**TutorialNavigation**(*nsteps*, *validate*, **args*, ***kwargs*)

Bases: PyQt5.QtWidgets.QWidget

A widget for navigating through the tutorial. It has a button to go to the previous step and to go to the next step. Furthermore it has a progressbar implemented a *hint* button

Parameters

- **nsteps** (*int*) – The total number of steps in the *Tutorial*
- **validate** (*callable*) – A callable that takes the *current_step* as an argument and returns a bool whether the current step is valid and finished, or not

Attributes

current_step	The current step in the tutorial
hint_requested(*args, **kwargs)	Signal that is emitted if the hint of the current step is requested
skipped(*args, **kwargs)	Signal that is emitted, if the step is skipped
step_changed(*args, **kwargs)	Signal that is emitted, when the step changes.

Methods

display_hint()	Trigger the hint_requested signal with the current step
goto_next_step()	Increase the current_step by one
goto_prev_step()	Decrease the current_step by one
maybe_enable_widgets()	Enable the buttons based on the current_step
setEnabled(enable)	Enable or disable the navigation buttons
set_current_step(i)	Change the current_step
show_info()	Trigger the step_changed signal with the current step
skip()	Skip the current_step and emit the skipped signal

current_step = 0
The current step in the tutorial

display_hint()
Trigger the *hint_requested* signal with the current step

goto_next_step()
Increase the *current_step* by one

goto_prev_step()
Decrease the *current_step* by one

hint_requested(**args*, ***kwargs*)

Signal that is emitted if the hint of the current step is requested

maybe_enable_widgets()
Enable the buttons based on the *current_step*

setEnabled(*enable*)
Enable or disable the navigation buttons

Parameters **enable** (*bool*) – Whether to enable or disable the buttons

set_current_step(*i*)
Change the *current_step*

Parameters **i** (*int*) – The *current_step* to switch to

show_info()
Trigger the *step_changed* signal with the current step

skip()
Skip the *current_step* and emit the *skipped* signal

skipped(**args*, ***kwargs*)
Signal that is emitted, if the step is skipped

step_changed(**args*, ***kwargs*)
Signal that is emitted, when the step changes. The first integer is the old step, the second one the current step

class straditize.widgets.tutorial.beginner.**TutorialPage**(*filename*, *tutorial*)
Bases: object

A base class for the tutorial pages

Subclasses show implement the *show_hint()* method and the *is_finished()* property

Parameters

- **filename** (*str*) – The basename (without ending) of the RST file corresponding to this tutorial page
- **tutorial** (Tutorial) – The tutorial instance

Methods

activate()	Method that is called, when the page is activated
deactivate()	Method that is called, when the page is deactivated
hint()	A method that should display a hint to the user
lock_viewer(lock)	Set or unset the url lock of the HTML viewer
refresh()	
show()	Show the page and browse the filename in the tutorial docs
show_hint()	Show a hint to the user
show_tooltip_at_widget(tooltip, widget[, ...])	Show a tooltip close to a widget
show_tooltip_in_plot(tooltip, x, y[, ...])	Show a tooltip in the matplotlib figure at the given coordinates
skip()	Skip the steps in this page

Attributes

is_finished	Boolean that is True, if the steps are all finished
is_selecting	True if the user clicked the btn_select_data button
src_base	The basename of the stratigraphic diagram image for this tutorial
src_dir	The source directory for the docs
src_file	The complete path to the of the stratigraphic diagram image for this

activate()
Method that is called, when the page is activated

deactivate()
Method that is called, when the page is deactivated

hint()
A method that should display a hint to the user

property is_finished
Boolean that is True, if the steps are all finished

property is_selecting
True if the user clicked the btn_select_data button

lock_viewer(*lock*)
Set or unset the url lock of the HTML viewer

refresh()

show()
Show the page and browse the `filename` in the tutorial docs

show_hint()
Show a hint to the user

show_tooltip_at_widget(*tooltip*, *widget*, *timeout=20000*)
Show a tooltip close to a widget

Parameters

- **tooltip** (*str*) – The tooltip to display
- **widget** (*QWidget*) – The widget that should be close to the tooltip
- **timeout** (*int*) – The time that the tool tip shall be displayed, in milliseconds

show_tooltip_in_plot(*tooltip*, *x*, *y*, *timeout=20000*, *transform=None*)
Show a tooltip in the matplotlib figure at the given coordinates

Parameters

- **tooltip** (*str*) – The tooltip to display
- **x** (*float*) – The x-coordinate of where to display the tooltip
- **y** (*float*) – The y-coordinate of where to display the tooltip
- **timeout** (*int*) – The time that the tool tip shall be displayed, in milliseconds
- **transform** (*matplotlib transformation*) – The matplotlib transformation to use. If None, the `self.stradi.ax.transData` transformation is used and *x* and *y* are expected to be in data coordinates

skip()
Skip the steps in this page

src_base = 'beginner-tutorial.png'
The basename of the stratigraphic diagram image for this tutorial

src_dir = '/home/docs/checkouts/readthedocs.org/user_builds/straditize/checkouts/lates
The source directory for the docs

src_file = '/home/docs/checkouts/readthedocs.org/user_builds/straditize/checkouts/late
The complete path to the of the stratigraphic diagram image for this tutorial

straditize.widgets.tutorial.hoya_del_castillo module

The tutorial of straditize

This module contains an advanced guided tour through straditize

Disclaimer

Classes

ColumnNames(filename, tutorial)	The page for recognizing column names
ColumnNamesOCR(filename, tutorial)	Tutorial page for column names with tesserocr support
EditMeta(filename, tutorial)	Tutorial page for editing the meta attributes
HoyaDelCastilloTutorial(straditizer_widgets)	A tutorial for digitizing an area diagram
LoadImage(filename, tutorial)	**param filename** The basename (without ending) of the RST file corresponding to this
RemoveLines(filename, tutorial)	Tutorial page for removing horizontal and vertical lines
SelectDataPart(filename, tutorial)	TutorialPage for selecting the data part
SeparateColumns(filename, tutorial)	The page for separating the columns
TranslateXAxis(filename, tutorial)	The tutorial page for translating the x-axes
TranslateYAxis(filename, tutorial)	The tutorial page for translating the y-axis
TutorialPage(filename, tutorial)	**param filename** The basename (without ending) of the RST file corresponding to this

class straditize.widgets.tutorial.hoya_del_castillo.**ColumnNames**(*filename*, *tutorial*)

Bases: *straditize.widgets.tutorial.beginner.ColumnNames*

The page for recognizing column names

Parameters

- **filename** (*str*) – The basename (without ending) of the RST file corresponding to this tutorial page
- **tutorial** (Tutorial) – The tutorial instance

Attributes

column_names	Built-in mutable sequence.
select_names_button_clicked	bool(x) -> bool

column_names = ['Charcoal', 'Pinus', 'Juniperus', 'Quercus ilex-type', 'Quercus suber-

select_names_button_clicked = False

class straditize.widgets.tutorial.hoya_del_castillo.**ColumnNamesOCR**(*filename*, *tutorial*)

Bases: *straditize.widgets.tutorial.hoya_del_castillo.ColumnNames*

Tutorial page for column names with tesserocr support

Parameters

- **filename** (*str*) – The basename (without ending) of the RST file corresponding to this tutorial page
- **tutorial** (Tutorial) – The tutorial instance

Attributes

colpic_extents	Built-in mutable sequence.
colpic_sizes	Built-in mutable sequence.

Methods

hint_for_start_editing()	
hint_for_wrong_name(col, curr, ref)	Display a hint if a name is not correctly set

colpic_extents = [(1051, 1127, 573, 588), (1176, 1225, 704, 719), (1338, 1422, 866, 88

colpic_sizes = [(270, 88), (189, 88), (307, 109), (494, 109), (550, 109), (159, 85), (

hint_for_start_editing()

hint_for_wrong_name(*col*, *curr*, *ref*)

Display a hint if a name is not correctly set

class straditize.widgets.tutorial.hoya_del_castillo.**EditMeta**(*filename*, *tutorial*)

Bases: *straditize.widgets.tutorial.hoya_del_castillo.TutorialPage*

Tutorial page for editing the meta attributes

Parameters

- **filename** (*str*) – The basename (without ending) of the RST file corresponding to this tutorial page

- **tutorial** (Tutorial) – The tutorial instance

Methods

hint()	A method that should display a hint to the user
skip()	Skip the steps in this page

Attributes

is_finished	Boolean that is True, if the steps are all finished

hint()
A method that should display a hint to the user

property is_finished
Boolean that is True, if the steps are all finished

skip()
Skip the steps in this page

class straditize.widgets.tutorial.hoya_del_castillo.**HoyaDelCastilloTutorial**(*straditizer_widgets*)
Bases: *straditize.widgets.tutorial.beginner.Tutorial*

A tutorial for digitizing an area diagram **Methods**

setup_tutorial_pages()	Setup the pages attribute and initialize the tutorial pages
show()	Show the documentation of the tutorial

Attributes

src_base	str(object='') -> str
src_dir	str(object='') -> str
src_file	str(object='') -> str

setup_tutorial_pages()
Setup the pages attribute and initialize the tutorial pages

show()
Show the documentation of the tutorial

src_base = 'hoya-del-castillo.png'

src_dir = '/home/docs/checkouts/readthedocs.org/user_builds/straditize/checkouts/lates

src_file = '/home/docs/checkouts/readthedocs.org/user_builds/straditize/checkouts/late

class straditize.widgets.tutorial.hoya_del_castillo.**LoadImage**(*filename*, *tutorial*)
Bases: *straditize.widgets.tutorial.beginner.LoadImage*, *straditize.widgets.tutorial.hoya_del_castillo.TutorialPage*

Parameters

- **filename** (*str*) – The basename (without ending) of the RST file corresponding to this tutorial page
- **tutorial** (Tutorial) – The tutorial instance

class straditize.widgets.tutorial.hoya_del_castillo.**RemoveLines**(*filename*, *tutorial*)

Bases: *straditize.widgets.tutorial.hoya_del_castillo.TutorialPage*

Tutorial page for removing horizontal and vertical lines

Parameters

- **filename** (*str*) – The basename (without ending) of the RST file corresponding to this tutorial page
- **tutorial** (Tutorial) – The tutorial instance

Methods

activate()	Method that is called, when the page is activated
clicked_hlines_button()	
clicked_vlines_button()	
deactivate()	Method that is called, when the page is deactivated
hint()	A method that should display a hint to the user
skip()	Skip the steps in this page

Attributes

hlines_button_clicked	bool(x) -> bool
is_finished	Boolean that is True, if the steps are all finished
vlines_button_clicked	bool(x) -> bool

activate()
Method that is called, when the page is activated

clicked_hlines_button()

clicked_vlines_button()

deactivate()
Method that is called, when the page is deactivated

hint()
A method that should display a hint to the user

hlines_button_clicked = False

property is_finished
Boolean that is True, if the steps are all finished

skip()
Skip the steps in this page

vlines_button_clicked = False

class straditize.widgets.tutorial.hoya_del_castillo.**SelectDataPart**(*filename*, *tutorial*)

Bases: *straditize.widgets.tutorial.beginner.SelectDataPart*

TutorialPage for selecting the data part

Parameters

- **filename** (*str*) – The basename (without ending) of the RST file corresponding to this tutorial page

- **tutorial** (Tutorial) – The tutorial instance

Attributes

ref_lims	The reference x- and y- limits
remove_data_box	bool(x) -> bool
valid_xlims	Valid ranges for xmin and xmax
valid_ylims	Valid ranges for ymin and ymax

ref_lims = array([[315, 1946], [511, 1311]])
The reference x- and y- limits

remove_data_box = False

valid_xlims = array([[310, 319], [1928, 1960]])
Valid ranges for xmin and xmax

valid_ylims = array([[508, 513], [1307, 1312]])
Valid ranges for ymin and ymax

class straditize.widgets.tutorial.hoya_del_castillo.**SeparateColumns**(*filename*, *tutorial*)

Bases: *straditize.widgets.tutorial.beginner.SeparateColumns*

The page for separating the columns

Parameters

- **filename** (*str*) – The basename (without ending) of the RST file corresponding to this tutorial page
- **tutorial** (Tutorial) – The tutorial instance

Attributes

ncols	int([x]) -> integer

ncols = 28

class straditize.widgets.tutorial.hoya_del_castillo.**TranslateXAxis**(*filename*, *tutorial*)

Bases: *straditize.widgets.tutorial.hoya_del_castillo.TutorialPage*

The tutorial page for translating the x-axes

Parameters

- **filename** (*str*) – The basename (without ending) of the RST file corresponding to this tutorial page
- **tutorial** (Tutorial) – The tutorial instance

Methods

activate()	Method that is called, when the page is activated
clicked_add_reader_button()	
clicked_translations_button()	
deactivate()	Method that is called, when the page is deactivated
hint()	A method that should display a hint to the user
hint_for_col(col)	

Continued on next page

Table 52 – continued from previous page

refresh()	
skip()	Skip the steps in this page

Attributes

add_reader_button_clicked	Built-in mutable sequence.
is_finished	Boolean that is True, if the steps are all finished
xaxis_translations_button_clicked	bool(x) -> bool

activate()
Method that is called, when the page is activated

add_reader_button_clicked = []

clicked_add_reader_button()

clicked_translations_button()

deactivate()
Method that is called, when the page is deactivated

hint()
A method that should display a hint to the user

hint_for_col(*col*)

property is_finished
Boolean that is True, if the steps are all finished

refresh()

skip()
Skip the steps in this page

xaxis_translations_button_clicked = False

class straditize.widgets.tutorial.hoya_del_castillo.**TranslateYAxis**(*filename*, *tutorial*)

Bases: *straditize.widgets.tutorial.hoya_del_castillo.TutorialPage*

The tutorial page for translating the y-axis

Parameters

- **filename** (*str*) – The basename (without ending) of the RST file corresponding to this tutorial page
- **tutorial** (Tutorial) – The tutorial instance

Methods

activate()	Method that is called, when the page is activated
clicked_correct_button()	
deactivate()	Method that is called, when the page is deactivated
hint()	A method that should display a hint to the user
skip()	Skip the steps in this page

Attributes

correct_button_clicked	bool(x) -> bool
is_finished	Boolean that is True, if the steps are all finished

activate()
Method that is called, when the page is activated

clicked_correct_button()

correct_button_clicked = False

deactivate()
Method that is called, when the page is deactivated

hint()
A method that should display a hint to the user

property is_finished
Boolean that is True, if the steps are all finished

skip()
Skip the steps in this page

class straditize.widgets.tutorial.hoya_del_castillo.**TutorialPage**(*filename*, *tutorial*)

Bases: *straditize.widgets.tutorial.beginner.TutorialPage*

Parameters

- **filename** (*str*) – The basename (without ending) of the RST file corresponding to this tutorial page
- **tutorial** (Tutorial) – The tutorial instance

Attributes

src_base	str(object=”) -> str
src_dir	str(object=”) -> str
src_file	str(object=”) -> str

src_base = 'hoya-del-castillo.png'

src_dir = '/home/docs/checkouts/readthedocs.org/user_builds/straditize/checkouts/lates

src_file = '/home/docs/checkouts/readthedocs.org/user_builds/straditize/checkouts/late

Submodules

straditize.widgets.axes_translations module

Module for translating the x- and y-axes from pixel into data coordinates

Disclaimer

This program is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU General Public License for more details.

You should have received a copy of the GNU General Public License along with this program. If not, see <https://www.gnu.org/licenses/>.

Classes

AxesTranslations(straditizer_widgets, item)	The control for translating x- and y-axes

class straditize.widgets.axes_translations.**AxesTranslations**(*straditizer_widgets*, *item*)

Bases: *straditize.widgets.StraditizerControlBase*

The control for translating x- and y-axes

This object creates two buttons for translating x- and y-axes from pixel to data coordinates

Parameters

- **straditizer_widgets** (*StraditizerWidgets*) – The main widget for the straditizer GUI
- **item** (*QTreeWidgetItem*) – The parent item in the `StraditizerWidgets.tree`. If given, the *setup_children()* is called with this item

Methods

marks_for_x([at_col_start])	Create (or enable) the marks for the x-axis translation
marks_for_y()	Create (or enable) the marks for the y-axis translation
setup_children(item)	Setup the children for this control
should_be_enabled(w)	Check if a widget should be enabled

Attributes

tree

marks_for_x(*at_col_start=True*)

Create (or enable) the marks for the x-axis translation

See also:

straditize.straditizer.Straditizer.marks_for_x_values(), *straditize.straditizer.Straditizer.update_xvalues()*

marks_for_y()

Create (or enable) the marks for the y-axis translation

See also:

straditize.straditizer.Straditizer.marks_for_y_values(), *straditize.straditizer.Straditizer.update_yvalues()*

setup_children(*item*)

Setup the children for this control

This method is called to setup the children in the `StraditizerWidgets.tree`. By default, it just creates a child QTreeWidgetItem and sets this control as it's widget

Parameters **item** (*QTreeWidgetItem*) – The top level item in the StraditizerWidgets.tree

should_be_enabled(*w*)

Check if a widget should be enabled

This function checks if a given widget *w* from the widgets2disable attribute should be enabled or not

Parameters **w** (*QWidget*) – The widget to check

Returns True, if the widget should be enabled

Return type bool

property tree

straditize.widgets.colnames module

Widget for handling column names

Disclaimer

Copyright (C) 2018-2019 Philipp S. Sommer

This program is free software: you can redistribute it and/or modify it under the terms of the GNU General Public License as published by the Free Software Foundation, either version 3 of the License, or (at your option) any later version.

This program is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU General Public License for more details.

You should have received a copy of the GNU General Public License along with this program. If not, see <https://www.gnu.org/licenses/>.

Classes

ColumnNamesManager(straditizer_widgets[, item])	Manage the column names of the reader
DummyNavigationToolbar2(canvas)	Reimplemented NavigationToolbar2 just to add an _init_toolbar method

class straditize.widgets.colnames.**ColumnNamesManager**(*straditizer_widgets*, *item=None*, **args*, ***kwargs*)

Bases: *straditize.widgets.StraditizerControlBase*, psyplot_gui.common.DockMixin, PyQt5.QtWidgets.QSplitter

Manage the column names of the reader

Parameters

- **straditizer_widgets** (StraditizerWidgets) – The main widget for the straditizer GUI
- **item** (*QTreeWidgetItem*) – The parent item in the StraditizerWidgets.tree. If given, the *setup_children()* is called with this item

Attributes

NAVIGATION_LABEL	str(object='') -> str
SELECT_LABEL	str(object='') -> str
btn_find	A QPushButton to find column names in the visible part of the
btn_load_image	A QPushButton to load the highres image
btn_recognize	A QPushButton to recognize text in the `colpic`
btn_select_colpic	A checkable QPushButton to initialize a `selector` to select the
btn_select_names	The QPushButton in the *straditize.widgets.StraditizerWidgets*
cb_find_all_cols	A QCheckBox to find the column names (see `btn_find`) for all
cb_fliph	A QCheckBox to set the `straditize.colnames.ColNamesReader.mirror`
cb_flipv	A QCheckBox to set the `straditize.colnames.ColNamesReader.flip`
cb_ignore_data_part	A QCheckBox to ignore the part within the
colnames_reader	The *straditize.straditizer.Straditizer.colnames_reader*
colnames_table	A QTableWidget to display the column names
colpic	The `PIL.Image.Image` of the column name (see also
colpic_ax	The `matplotlib.axes.Axes` to display the `colpic_im`
colpic_canvas	The canvas to display the `colpic_im`
colpic_extents	The extents of the `colpic` in the `im_rotated`
colpic_im	The matplotlib image of the `colpic`
current_col	The currently selected column
fig_h	The original height of the `main_canvas`
fig_w	The original width of the `main_canvas`
im_rotated	The matplotlib image of the
main_ax	The `matplotlib.axes.Axes` to display the `im_rotated`
main_canvas	The canvas to display the `im_rotated`
rect	The rectangle to highlight a column (see `highlight_selected_col()`)
refreshing	True if the widget is refreshing
selector	The `matplotlib.widgets.RectangleSelector` to select the
txt_rotate	A QLineEdit to set the `straditize.colnames.ColNamesReader.rotate`

Methods

adjust_lims()	Adjust the limits of the `main_ax` to fill the entire figure
adjust_lims_after_resize(event)	Adjust the limits of the `main_ax` after resize of the figure
cancel_colpic_selection()	Stop the colpic selection in the `im_rotated`
change_ignore_data_part(checked)	Change *straditize.colnames.ColNamesReader.ignore_data_part*

Continued on next page

Table 62 – continued from previous page

colname_changed(row, column)	Update the column name in the colnames_reader
create_selector()	Create the selector to enable colpic selection
enable_or_disable_btn_find(*args, **kwargs)	
find_colnames([warn, full_image, all_cols])	Find the column names automatically
flip(checked)	TFlip the image
highlight_selected_col()	Highlight the column selected in the colnames_tables
load_image()	Load a high resolution image
maybe_tabify()	
mirror(checked)	Mirror the image
plot_colpic()	Plot the colpic in the colpic_ax
read_colpic()	Recognize the text in the colpic
refresh()	Refresh from the straditizer
remove_images()	Remove the im_rotated and the colpic_im
remove_selector()	Remove and disconnect the selector
replot_figure()	Remove and replot the im_rotated
reset_control()	Reset the dialog
rotate(val)	Rotate the image
set_xc_yc()	Set the x- and y-center before rotating or flipping
setup_children(item)	Setup the children for this control
should_be_enabled(w)	Check if a widget should be enabled
to_dock(main[, title, position])	
toggle_colpic_selection()	Enable or disable the colpic selection
toggle_dialog()	Close the dialog when the btn_select_names button is clicked
update_image(*args, **kwargs)	Update the colpic with the extents of the selector

NAVIGATION_LABEL = 'Use left-click of your mouse to move the image below and right-cli

SELECT_LABEL = 'Left-click and hold on the image to select the column name'

adjust_lims()
Adjust the limits of the *main_ax* to fill the entire figure

adjust_lims_after_resize(*event*)
Adjust the limits of the *main_ax* after resize of the figure

btn_find = None
A QPushButton to find column names in the visible part of the *im_rotated*

btn_load_image = None
A QPushButton to load the highres image

btn_recognize = None
A QPushButton to recognize text in the *colpic*

btn_select_colpic = None
A checkable QPushButton to initialize a *selector* to select the *colpic*

btn_select_names = None
The QPushButton in the *straditize.widgets.StraditizerWidgets* to toggle the column names dialog

cancel_colpic_selection()
Stop the colpic selection in the *im_rotated*

cb_find_all_cols = None
A QCheckBox to find the column names (see *btn_find*) for all columns and not just the one selected in the *colnames_table*

cb_fliph = None
A QCheckBox to set the straditize.colnames.ColNamesReader.mirror

cb_flipv = None
A QCheckBox to set the straditize.colnames.ColNamesReader.flip

cb_ignore_data_part = None
A QCheckBox to ignore the part within the *straditize.colnames.ColNamesReader.data_ylim*

change_ignore_data_part(*checked*)
Change *straditize.colnames.ColNamesReader.ignore_data_part*

colname_changed(*row*, *column*)
Update the column name in the *colnames_reader*

This method is called when a cell in the *colnames_table* has been changed and updates the corresponding name in the *colnames_reader*

Parameters

- **row** (*int*) – The row of the cell in the *colnames_table* that changed
- **column** (*int*) – The column of the cell in the *colnames_table* that changed

property colnames_reader
The *straditize.straditizer.Straditizer.colnames_reader* of the current straditizer

colnames_table = None
A QTableWidget to display the column names

colpic = None
The PIL.Image.Image of the column name (see also *straditize.colnames.ColNamesReader.colpics*)

colpic_ax = None
The matplotlib.axes.Axes to display the *colpic_im*

colpic_canvas = None
The canvas to display the *colpic_im*

colpic_extents = None
The extents of the *colpic* in the *im_rotated*

colpic_im = None
The matplotlib image of the *colpic*

create_selector()
Create the *selector* to enable *colpic* selection

property current_col
The currently selected column

enable_or_disable_btn_find(**args*, ***kwargs*)

fig_h = None
The original height of the *main_canvas*

fig_w = None
The original width of the *main_canvas*

find_colnames(*warn=True*, *full_image=False*, *all_cols=None*)
Find the column names automatically

See also:

straditize.colnames.ColNamesReader.find_colnames()

flip(*checked*)
TFlip the image

highlight_selected_col()
Highlight the column selected in the `colnames_tables`

See also:

straditize.colnames.ColNamesReader.highlight_column()

im_rotated = None
The matplotlib image of the *straditize.colnames.ColNamesReader.rotated_image*

load_image()
Load a high resolution image

main_ax = None
The `matplotlib.axes.Axes` to display the *im_rotated*

main_canvas = None
The canvas to display the *im_rotated*

maybe_tabify()

mirror(*checked*)
Mirror the image

plot_colpic()
Plot the *colpic* in the *colpic_ax*

read_colpic()
Recognize the text in the *colpic*

See also:

straditize.colnames.ColNamesReader.recognize_text()

rect = None
The rectangle to highlight a column (see *highlight_selected_col()*)

refresh()
Refresh from the straditizer

property refreshing
True if the widget is refreshing

remove_images()
Remove the *im_rotated* and the *colpic_im*

remove_selector()
Remove and disconnect the *selector*

replot_figure()
Remove and replot the *im_rotated*

reset_control()
Reset the dialog

rotate(*val*)
Rotate the image

Parameters **float** – The angle for the rotation

selector = None
The matplotlib.widgets.RectangleSelector to select the *colpic*

set_xc_yc()
Set the x- and y-center before rotating or flipping

setup_children(*item*)
Setup the children for this control

This method is called to setup the children in the StraditizerWidgets.tree. By default, it just creates a child QTreeWidgetItem and sets this control as it's widget

Parameters **item** (*QTreeWidgetItem*) – The top level item in the StraditizerWidgets.tree

should_be_enabled(*w*)
Check if a widget should be enabled

This function checks if a given widget *w* from the widgets2disable attribute should be enabled or not

Parameters **w** (*QWidget*) – The widget to check

Returns True, if the widget should be enabled

Return type bool

to_dock(*main*, *title=None*, *position=None*, **args*, ***kwargs*)

toggle_colpic_selection()
Enable or disable the colpic selection

toggle_dialog()
Close the dialog when the *btn_select_names* button is clicked

txt_rotate = None
A QLineEdit to set the straditize.colnames.ColNamesReader.rotate

update_image(**args*, ***kwargs*)
Update the *colpic* with the extents of the *selector*

*args and **kwargs are ignored

class straditize.widgets.colnames.**DummyNavigationToolbar2**(*canvas*)
Bases: matplotlib.backend_bases.NavigationToolbar2

Reimplemented NavigationToolbar2 just to add an _init_toolbar method **Methods**

set_cursor(cursor)	Set the current cursor to one of the Cursors enums values.

set_cursor(*cursor*)
Set the current cursor to one of the Cursors enums values.

If required by the backend, this method should trigger an update in the backend event loop after the cursor is set, as this method may be called e.g. before a long-running task during which the GUI is not updated.

straditize.widgets.data module

The main control widget for handling the data

Disclaimer

Classes

BarSplitter(straditizer_widgets, *args, **kwargs)	A widget for splitting bars
DigitizingControl(straditizer_widgets, item)	An interface to *straditize.straditizer.Straditizer.data_reader*

Functions

get_reader_name(reader)	Get the reader key in the `straditize.binary.readers` dictionary
int_list2str(numbers)	Create a short string representation of an integer list

class straditize.widgets.data.**BarSplitter**(*straditizer_widgets*, **args*, ***kwargs*)

Bases: `PyQt5.QtWidgets.QTreeWidget`, *straditize.widgets.StraditizerControlBase*

A widget for splitting bars **Methods**

add_toolbar_widgets()	Add an action to switch between the bars to the matplotlib toolbar
disconnect()	Disconnect the events to split an item
enable_or_disable_widgets(b)	Enable or disable the widgets in this control
fill_table([source])	Fill the table with the bars that should be splitted
get_overlapping_bars()	Get the bars the overlap with multiple bars in another column
go_to_next_bar()	Go to the `next_item`
go_to_prev_bar()	Go to the `previous_item`
new_split(y, y0[, draw_figure])	Mark the current item to be splitted at *y*
prepare_for_split(event)	Select or deselect a split location
refresh()	Reimplemented to use the `fill_table()` method
remove_actions()	Remove the actions added by `add_toolbar_widgets()`
remove_lines()	Remove the plotted lines
remove_split_children()	Remove all the child items that mark a split
revert_split(y, y0)	Revert the split

Continued on next page

Table 66 – continued from previous page

set_suggestions_fig_titles(ax)	Set the title in the suggestion figure with the displayed column
split_bars()	Split the bars after they have been separated manually
start_splitting(item, *args, **kwargs)	Enable the splitting for the selected item
suggest_splits()	Find overlaps for the current selected bar in other columns

Attributes

next_item	The QTreeWidgetItem for the next bar to split
prev_action	The action in the matplotlib toolbar to go to the previous bar to split
previous_item	The QTreeWidgetItem for the previous bar to split
selected_child	The QTreeWidgetItem that is currently shown in the plot
suggestions_fig	A figure to show the other columns

add_toolbar_widgets()
Add an action to switch between the bars to the matplotlib toolbar

disconnect()
Disconnect the events to split an item

enable_or_disable_widgets(*b*)
Enable or disable the widgets in this control

This method enables or disables the `widgets2disable` if the `should_be_enabled()` method evaluates to True

Parameters **b** (*bool*) – If True, enable the widgets, if False, disable them

fill_table(*source='too-long'*)
Fill the table with the bars that should be splitted

Parameters **source** (*{ 'too-long' | 'overlaps' | 'all' }*) – The source with what to fill the table.

too-long Only display the bars that are considered as *too long*

overlap Only display the bars that overlap with multiple bars in another column (see *get_overlapping_bars()*)

all Display all bars

get_overlapping_bars()
Get the bars the overlap with multiple bars in another column

go_to_next_bar()
Go to the *next_item*

go_to_prev_bar()
Go to the *previous_item*

new_split(*y*, *y0*, *draw_figure=True*)
Mark the current item to be splitted at *y*

This method draws a horizontal line at *y* and adds a new child QTreeWidgetItem to the *selected_child* to mark the split

Parameters

- **y** (*int*) – The vertical position of the split in the data image coordinate system
- **y0** (*int*) – The vertical start of the data image (see *straditize.binary.DataReader.extent*)
- **draw_figure** (*bool*) – If True, draw the figure

property next_item
The QTreeWidgetItem for the next bar to split

prepare_for_split(*event*)
Select or deselect a split location

LeftButton selects the given location for a new split (see *new_split()*), RightButton deselects it (see *revert_split()*)

prev_action = None
The action in the matplotlib toolbar to go to the previous bar to split

property previous_item
The QTreeWidgetItem for the previous bar to split

refresh()
Reimplemented to use the *fill_table()* method

remove_actions()
Remove the actions added by *add_toolbar_widgets()*

remove_lines()
Remove the plotted lines

remove_split_children()
Remove all the child items that mark a split

revert_split(*y*, *y0*)
Revert the split

Parameters

- **y** (*int*) – The vertical position of the split in the data image coordinate system
- **y0** (*int*) – The vertical start of the data image (see *straditize.binary.DataReader.extent*)

selected_child = None
The QTreeWidgetItem that is currently shown in the plot

set_suggestions_fig_titles(*ax*)
Set the title in the suggestion figure with the displayed column

split_bars()
Split the bars after they have been separated manually

start_splitting(*item*, **args*, ***kwargs*)
Enable the splitting for the selected item

suggest_splits()
Find overlaps for the current selected bar in other columns

suggestions_fig = None
A figure to show the other columns

class straditize.widgets.data.**DigitizingControl**(*straditizer_widgets*, *item*)

Bases: *straditize.widgets.StraditizerControlBase*

An interface to *straditize.straditizer.Straditizer.data_reader*

This widgets contains the functionalities to interface with the data readers for the stratigraphic diagram

Parameters

- **straditizer_widgets** (StraditizerWidgets) – The main widget for the straditizer GUI
- **item** (*QTreeWidgetItem*) – The parent item in the StraditizerWidgets.tree. If given, the *setup_children()* is called with this item

Methods

align_vertical()	Create marks for vertical alignment of the columns
change_reader(txt)	Change the current parent reader
digitize()	Digitize the data
digitize_exaggerations()	Digitize the data
edit_occurences()	Enable the editing of occurences
edit_samples()	Enable the sample editing
enable_col_selection_for_new_reader()	Start the selection process to get a new reader for specific cols
enable_occurences_selection()	Enable the selection of occurences
enable_or_disable_btn_highlight_small_selection()	Enable the btn_highlight_small_selection during a selection
enable_or_disable_widgets(*args, **kwargs)	Enable or disable the widgets in this control
fill_cb_readers()	Fill the cb_readers combo based on the current reader
find_samples()	
finish_exaggerated_features()	Save the exaggerations in the exaggerations reader
init_exaggerated_reader()	Initialize the reader for exaggeration features
init_reader()	Initialize the reader
load_samples([fname])	Load the samples of a text file
maybe_show_btn_reset_columns()	Show the btn_reset_columns if the column starts are set
maybe_show_btn_reset_samples()	Show the btn_reset_samples if the samples are set
modify_column_ends()	Modify the column ends
new_reader_for_selection([cls])	Create a new child reader for the selected columns
refresh()	Refresh from the straditizer
remove_hlines()	Remove horizontal lines
remove_vlines()	Remove vertical lines
remove_xaxes()	Remove x-axes in the plot
remove_yaxes()	Remove y-axes in the plot
reset_column_starts()	Reset the column starts
reset_samples()	Reset the samples
select_column_starts()	Estimate the column starts and draw marks
select_data_part([guess_lims])	Enable the selection of the diagram part
select_exaggerated_features()	Enable the selection of exaggerated features
select_occurences()	Save (and potentially remove) the selected occurences

Continued on next page

Table 68 – continued from previous page

set_occurences_value(value)	Set the *occurences_value*
setup_children(item)	Set up the child items for a topLevelItem in the control tree
should_be_enabled(w)	Check if a widget should be enabled
show_cross_column_features()	Remove cross column features
show_disconnected_parts()	Remove disconnected parts
show_parts_at_column_ends()	Remove parts that touch the column ends
show_small_parts()	Remove parts that touch the column ends
toggle_bar_split_source(i)	Fill the `tree_bar_split` based on the `cb_split_source`
toggle_btn_highlight_small_selection()	Enable or disable the `btn_highlight_small_selection`
toggle_sp_max_lw(state)	Toggle `sp_max_lw` based on `cb_max_lw`
toggle_txt_edit_rows(state)	Toggle `txt_edit_rows` based on `cb_edit_separate`
toggle_txt_from0(state)	Toggle `txt_from0` based on `cb_from0`
toggle_txt_fromlast(state)	Toggle `txt_fromlast` based on `cb_fromlast`
toggle_txt_tolerance(s)	Set the visibility of the `txt_tolerance` based on the reader
update_tolerance(s)	Set the readers `tolerance`

Attributes

btn_column_ends	Button for selecting and modifying column ends
btn_column_starts	Button for selecting and modifying column starts, see the *straditize.straditizer.Straditizer.marks_for_column_starts()* method.
btn_digitize	Button for digitizing the diagram
btn_digitize_exag	Button to digitize the exaggerations
btn_edit_occurences	A button to edit the occurences with the
btn_edit_samples	A button to edit the samples (see the
btn_find_samples	A button to find the samples with the
btn_highlight_small_selection	A button to highlight small selections using the
btn_init_reader	Button for initializing the reader
btn_load_samples	A button to load samples from a file
btn_new_child_reader	Button to add a new column-specific child reader
btn_new_exaggeration	Button to add an exaggerations reader
btn_remove_hlines	button for removing horizontal lines
btn_remove_vlines	Button for removing vertical lines
btn_remove_xaxes	button for removing x-axes
btn_remove_yaxes	Button for removing y-axes
btn_reset_columns	Button to reset the column starts and ends
btn_reset_samples	A button to reset the samples
btn_select_data	Button for selecting the data box, see the *straditize.straditizer.Straditizer.marks_for_data_selection()* method.
btn_select_exaggerations	Button to select the exaggerations
btn_select_occurences	A button to select occurences in the data part (see the

Continued on next page

Table 69 – continued from previous page

btn_show_cross_column	A button to
btn_show_disconnected_parts	Button for removing disconnected parts in the plot.
btn_show_parts_at_column_ends	A button to show the parts that touch the column end
btn_show_small_parts	A button to *show_small_parts()*
cb_edit_separate	A QCheckBox to edit the samples in a separate figure and not inside the
cb_exag_reader_type	A QComboBox to select the reader type for exaggerations
cb_from0	A QCheckBox to enable and disable the *from0* keyword in the
cb_fromlast	A QCheckBox to enable and disable the *fromlast* keyword in the
cb_max_lw	QCheckBox to enable and disable the maximum linewidth as a criterion
cb_reader_type	Combobox for selecting the reader type
cb_remove_occurences	A QCheckBox to remove the occurences in the plot after selection
cb_split_source	A QComboBox to select whether to fill the `tree_bar_split` with
reader	The *straditize.straditizer.Straditizer.data_reader*
selection_toolbar	
sp_max_lw	A QSpinBox to select the maximum linewidth
sp_min_lw	A QSpinBox to select the minimum linewidth
sp_pixel_tol	A QSpinBox to set the minimum distance between to samples in the
tree	The *straditize.widgets.StraditizerWidgets.tree*
tree_bar_split	A *BarSplitter* to split too long bars
txt_column_thresh	A QLineEdit to set the threshold for the column starts detection
txt_cross_column_px	A QLineEdit to select the minimum pixels (*min_px*) for a cross column
txt_edit_rows	A QLineEdit to specify the number or rows in a plot for editing the
txt_exag_factor	A QLineEdit for the exageration factor
txt_from0	A QLineEdit to set the *from0* keyword for the
txt_fromlast	A QLineEdit to set the *fromlast* keyword for the
txt_line_fraction	LineEditor for specifying the fraction of vertical and horizontal lines
txt_max_len	A QLinEdit to specify the maximum length of a potential sample to be
txt_max_small_size	A QLineEdit to set the size for small parts
txt_min_highlight	A QLineEdit to set the maximal size for
txt_min_len	A QLinEdit to specify the minimum length of a potential sample to be
txt_occurences_value	A QLineEdit to set the value for occurences in the final data
txt_tolerance	Line edit for setting the tolerance for bars

align_vertical()
Create marks for vertical alignment of the columns

See also:

straditize.straditizer.Straditizer.marks_for_vertical_alignment(), *straditize.straditizer.Straditizer.align_columns()*

btn_column_ends = None
Button for selecting and modifying column ends

btn_column_starts = None
Button for selecting and modifying column starts, see the *straditize.straditizer.Straditizer.marks_for_column_starts()* method.

btn_digitize = None
Button for digitizing the diagram

btn_digitize_exag = None
Button to digitize the exaggerations

btn_edit_occurences = None
A button to edit the occurences with the *straditize.straditizer.Straditizer.marks_for_occurences()* method

btn_edit_samples = None
A button to edit the samples (see the *straditize.straditizer.Straditizer.marks_for_samples()* and *straditize.straditizer.Straditizer.marks_for_samples_sep()*)

btn_find_samples = None
A button to find the samples with the *straditize.binary.DataReader.find_samples()* method

btn_highlight_small_selection = None
A button to highlight small selections using the *straditize.label_selection.LabelSelection.highlight_small_selections()* method

btn_init_reader = None
Button for initializing the reader

btn_load_samples = None
A button to load samples from a file

btn_new_child_reader = None
Button to add a new column-specific child reader

btn_new_exaggeration = None
Button to add an exaggerations reader

btn_remove_hlines = None
button for removing horizontal lines

btn_remove_vlines = None
Button for removing vertical lines

btn_remove_xaxes = None
button for removing x-axes

btn_remove_yaxes = None
Button for removing y-axes

btn_reset_columns = None
Button to reset the column starts and ends

btn_reset_samples = None
A button to reset the samples

btn_select_data = None
Button for selecting the data box, see the *straditize.straditizer.Straditizer.marks_for_data_selection()* method.

btn_select_exaggerations = None
Button to select the exaggerations

btn_select_occurences = None
A button to select occurences in the data part (see the *enable_occurences_selection()* method)

btn_show_cross_column = None
A button to *show_cross_column_features()*

btn_show_disconnected_parts = None
Button for removing disconnected parts in the plot. See the *straditize.binary.DataReader.show_disconnected_parts()* method

btn_show_parts_at_column_ends = None
A button to show the parts that touch the column end

btn_show_small_parts = None
A button to *show_small_parts()*

cb_edit_separate = None
A QCheckBox to edit the samples in a separate figure and not inside the original diagram

cb_exag_reader_type = None
A QComboBox to select the reader type for exaggerations

cb_from0 = None
A QCheckBox to enable and disable the *from0* keyword in the *straditize.binary.DataReader.show_disconnected_parts()* method

cb_fromlast = None
A QCheckBox to enable and disable the *fromlast* keyword in the *straditize.binary.DataReader.show_disconnected_parts()* method

cb_max_lw = None
QCheckBox to enable and disable the maximum linewidth as a criterion

cb_reader_type = None
Combobox for selecting the reader type

cb_remove_occurences = None
A QCheckBox to remove the occurences in the plot after selection

cb_split_source = None
A QComboBox to select whether to fill the *tree_bar_split* with too long, overlapping, or all bars

change_reader(*txt*)
Change the current parent reader

This changes the *straditize.straditizer.Straditizer.data_reader* using the *straditize.binary.DataReader.set_as_parent()* method

Parameters **s** (*str*) – A string matching `'Columns (\d.*)'`, where the numbers are the columns of the reader to use

digitize()
Digitize the data

This method uses the *straditize.binary.DataReader.digitize()* method to digitize the data of the current reader

digitize_exaggerations()
Digitize the data

This method uses the *straditize.binary.DataReader.digitize_exaggerated()* method to digitize the exaggerated data of the current reader and merge it into the data obtained by the *digitize()* method.

digitize_item = None

edit_occurences()
Enable the editing of occurences

This enables the editing of occurences using the *straditize.straditizer.Straditizer.marks_for_occurences()* method for the occurences selected by the *select_occurences()* method

edit_samples()
Enable the sample editing

This method opens a *straditize.widgets.samples_table.MultiCrossMarksEditor* or a *straditize.widgets.samples_table.SingleCrossMarksEditor* to edit the samples in the GUI. Depending on whether the *cb_edit_separate* is checked or not, we use the *straditize.straditizer.Straditizer.marks_for_samples_sep()* or *straditize.straditizer.Straditizer.marks_for_samples()* method.

enable_col_selection_for_new_reader()
Start the selection process to get a new reader for specific cols

enable_occurences_selection()
Enable the selection of occurences

This method starts the selection of features in the data image and connects the *select_occurences()* to the `apply_button`.

enable_or_disable_btn_highlight_small_selection()
Enable the *btn_highlight_small_selection* during a selection

enable_or_disable_widgets(**args*, ***kwargs*)
Enable or disable the widgets in this control

This method enables or disables the `widgets2disable` if the *should_be_enabled()* method evaluates to True

Parameters **b** (*bool*) – If True, enable the widgets, if False, disable them

fill_cb_readers()
Fill the `cb_readers` combo based on the current reader

find_samples()

finish_exaggerated_features()
Save the exaggerations in the exaggerations reader

This method finalizes the operation initialized by the *select_exaggerated_features()* by calling the *straditize.binary.DataReader.mark_as_exaggerations()* method

init_exaggerated_reader()
Initialize the reader for exaggeration features

init_reader()
Initialize the reader

Initialize the data reader with the *straditize.straditizer.Straditizer.init_reader()* method

load_samples(*fname=None*)
Load the samples of a text file

This method askes for a filename to update the samples. The first column in this file is taken as the sample locations. If the y-axis translation is already done, the new data is assumed to be in this transformed unit.

Parameters **fname** (*str*) – The path to the file to use. If None, a QFileDialog is opened and we ask for a name

maybe_show_btn_reset_columns()
Show the *btn_reset_columns* if the column starts are set

maybe_show_btn_reset_samples()
Show the *btn_reset_samples* if the samples are set

modify_column_ends()
Modify the column ends

After having selected the *column starts*, this method enables the modification of the column ends

See also:

select_column_starts(), *straditize.straditizer.Straditizer.marks_for_column_ends()*, *straditize.straditizer.Straditizer.update_column_ends()*

new_reader_for_selection(*cls=None*)
Create a new child reader for the selected columns

This method finishes the process started by *enable_col_selection_for_new_reader()*

Parameters **cls** (*type*) – The subclass of the *straditize.binary.DataReader* class to use for the new reader. If None, a QInputDialog is opened and we ask for a reader

property reader
The *straditize.straditizer.Straditizer.data_reader*

refresh()
Refresh from the straditizer

remove_hlines()
Remove horizontal lines

This method uses the *straditize.binary.DataReader.recognize_hlines()* method to identify horizontal lines in the plot

remove_vlines()
Remove vertical lines

This method uses the *straditize.binary.DataReader.recognize_vlines()* method to identify vertical lines in the plot

remove_xaxes()
Remove x-axes in the plot

This method uses the *straditize.binary.DataReader.recognize_xaxes()* method to identify x-axes in the plot

remove_yaxes()
Remove y-axes in the plot

This method uses the *straditize.binary.DataReader.recognize_yaxes()* method to identify y-axes in the plot

reset_column_starts()
Reset the column starts

Reset the column starts by calling the *straditize.binary.DataReader.reset_column_starts()* method

reset_samples()
Reset the samples

Reset the samples by calling the *straditize.binary.DataReader.reset_samples()* method

select_column_starts()
Estimate the column starts and draw marks

This methods estimates the column starts (if they are not yet set) based on the threshold in the *txt_column_thresh* and draws `straditize.cross_marks.DraggableVLine` marks on the plot.

See also:

straditize.straditizer.Straditizer.marks_for_column_starts(), *straditize.straditizer.Straditizer.update_column_starts()*

select_data_part(*guess_lims=True*)
Enable the selection of the diagram part

This method uses the *straditize.straditizer.Straditizer.marks_for_data_selection()* method to draw cross marks on the image for the diagram part

select_exaggerated_features()
Enable the selection of exaggerated features

select_occurences()
Save (and potentially remove) the selected occurences

Save the occurences with the *straditize.binary.DataReader.get_occurences()* method and remove them if the *cb_remove_occurences* is checked

property selection_toolbar

set_occurences_value(*value*)
Set the *occurences_value*

Set the `straditize.binary.DataReader.occurence_value` of the data_reader with the given value

Parameters **value** (*float*) – The value to use for occurences

setup_children(*item*)
Set up the child items for a topLevelItem in the control tree

should_be_enabled(*w*)
Check if a widget should be enabled

This function checks if a given widget *w* from the `widgets2disable` attribute should be enabled or not

Parameters **w** (*QWidget*) – The widget to check

Returns True, if the widget should be enabled

Return type bool

show_cross_column_features()
Remove cross column features

This method highlights features that span multiple columns using the *straditize.binary.DataReader.show_cross_column_features()* method. The algorithm can be modified with the *txt_cross_column_px* line editor

show_disconnected_parts()
Remove disconnected parts

This method uses the *straditize.binary.DataReader.show_disconnected_parts()* to highlight and remove disconnected features in the diagram part. The algorithm can be modified by the *txt_fromlast* and *txt_from0* text editors

show_parts_at_column_ends()
Remove parts that touch the column ends

This method highlights features that touch the column ends using the *straditize.binary.DataReader.show_parts_at_column_ends()* method.

show_small_parts()
Remove parts that touch the column ends

This method highlights small features in the data image using the *straditize.binary.DataReader.show_small_parts()* method. The maximal size of the small features can is taken from the *txt_max_small_size* line editor

sp_max_lw = None
A QSpinBox to select the maximum linewidth

sp_min_lw = None
A QSpinBox to select the minimum linewidth

sp_pixel_tol = None
A QSpinBox to set the minimum distance between to samples in the sample finding algorithm

toggle_bar_split_source(*i*)
Fill the *tree_bar_split* based on the *cb_split_source*

Parameters **i** (*int*) – The *BarSplitter.fill_table()* is called with either `'too-long'` (if *i* is 0), `'overlaps'` (if *i* is 1) or `'all'`

toggle_btn_highlight_small_selection()
Enable or disable the *btn_highlight_small_selection*

This method enables the *btn_highlight_small_selection* button if we are selecting something at the moment

toggle_sp_max_lw(*state*)
Toggle *sp_max_lw* based on *cb_max_lw*

toggle_txt_edit_rows(*state*)
Toggle *txt_edit_rows* based on *cb_edit_separate*

toggle_txt_from0(*state*)
Toggle *txt_from0* based on *cb_from0*

toggle_txt_fromlast(*state*)
Toggle *txt_fromlast* based on *cb_fromlast*

toggle_txt_tolerance(*s*)
Set the visibility of the *txt_tolerance* based on the reader

Parameters **s** (*str*) – The reader name. If there is *bars* in *s*, then the *txt_tolerance* is displayed

property tree
The *straditize.widgets.StraditizerWidgets.tree*

tree_bar_split = None
A *BarSplitter* to split too long bars

txt_column_thresh = None
A QLineEdit to set the threshold for the column starts detection

txt_cross_column_px = None
A QLineEdit to select the minimum pixels (*min_px*) for a cross column feature

txt_edit_rows = None
A QLineEdit to specify the number or rows in a plot for editing the samples in a separate figure (see *btn_edit_samples* and *cb_edit_separate*)

txt_exag_factor = None
A QLineEdit for the exageration factor

txt_from0 = None
A QLineEdit to set the *from0* keyword for the *straditize.binary.DataReader.show_disconnected_parts()* method

txt_fromlast = None
A QLineEdit to set the *fromlast* keyword for the *straditize.binary.DataReader.show_disconnected_parts()* method

txt_line_fraction = None
LineEditor for specifying the fraction of vertical and horizontal lines

txt_max_len = None
A QLinEdit to specify the maximum length of a potential sample to be included in the sample finding algorithm (see *btn_find_samples*)

txt_max_small_size = None
A QLineEdit to set the size for small parts

txt_min_highlight = None
A QLineEdit to set the maximal size for *highlighting small features*

txt_min_len = None
A QLinEdit to specify the minimum length of a potential sample to be included in the sample finding algorithm (see *btn_find_samples*)

txt_occurences_value = None
A QLineEdit to set the value for occurences in the final data

txt_tolerance = None
Line edit for setting the tolerance for bars

update_tolerance(*s*)
Set the readers `tolerance`

Parameters **s** (*str or int*) – The tolerance for the `straditizer.binary.BarDataReader.tolerance` attribute

`straditize.widgets.data.`**get_reader_name**(*reader*)
Get the reader key in the `straditize.binary.readers` dictionary

Parameters **reader** (straditize.binary.DataReader) – The reader for which to get the key in the straditize.binary.readers dictionary

Returns The key in the straditize.binary.readers dictionary whose value corresponds to the class of the given *reader*

Return type str

straditize.widgets.data.**int_list2str**(*numbers*)

Create a short string representation of an integer list

Parameters **numbers** (*list of ints*) – Integer list

Returns The string representation

Return type str

Examples

`[1, 2, 3]` becomes `'1-3'`, `[1, 2, 3, 5, 7, 8, 9]` becomes `'1-3, 5, 7-9'`

straditize.widgets.image_correction module

Image correction methods

Disclaimer

Copyright (C) 2018-2019 Philipp S. Sommer

This program is free software: you can redistribute it and/or modify it under the terms of the GNU General Public License as published by the Free Software Foundation, either version 3 of the License, or (at your option) any later version.

This program is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU General Public License for more details.

You should have received a copy of the GNU General Public License along with this program. If not, see <https://www.gnu.org/licenses/>.

Classes

ImageRescaler(straditizer_widgets, item, ...)	A button to rescale the straditize image
ImageRotator(straditizer_widgets[, item])	Widget to rotate the image

class straditize.widgets.image_correction.**ImageRescaler**(*straditizer_widgets*, *item*, **args*, ***kwargs*)

Bases: *straditize.widgets.StraditizerControlBase*, PyQt5.QtWidgets.QPushButton

A button to rescale the straditize image **Methods**

adjust_orig_limits(*args, **kwargs)	Readjust ax_orig after changes in ax_rescale
adjust_rescaled_limits(*args, **kwargs)	Readjust ax_rescale after changes in ax_orig
close_figs()	Close the fig
draw_figure()	
equalize_axes([event])	Set both axes to the same size
raise_figure()	Raise the figure for rescaling

Continued on next page

Table 71 – continued from previous page

rescale([ask])	Rescale and start a new straditizer
rescale_plot(percentage)	Replot `im_rescale` after adjustments of the `slider`
resize_stradi_image(percentage)	Resize the straditizer image
should_be_enabled(w)	Check if a widget should be enabled
start_rescaling()	Create the rescaling figure

Attributes

ax_orig	The matplotlib axes for the `im_orig`
ax_rescale	The matplotlib axes for the `im_rescale`
fig	The matplotlib figure for the rescaling
im_orig	The matplotlib image for the original diagram
im_rescale	The matplotlib image for the rescaled diagram
rescaling	Boolean that is true if one of the axes is rescaling
slider	A `matplotlib.widgets.Slider` for specifying the size of the

adjust_orig_limits(**args*, ***kwargs*)
Readjust *ax_orig* after changes in *ax_rescale*

adjust_rescaled_limits(**args*, ***kwargs*)
Readjust *ax_rescale* after changes in *ax_orig*

ax_orig = None
The matplotlib axes for the *im_orig*

ax_rescale = None
The matplotlib axes for the *im_rescale*

close_figs()
Close the *fig*

draw_figure()

equalize_axes(*event=None*)
Set both axes to the same size

fig = None
The matplotlib figure for the rescaling

im_orig = None
The matplotlib image for the original diagram

im_rescale = None
The matplotlib image for the rescaled diagram

raise_figure()
Raise the figure for rescaling

rescale(*ask=None*)
Rescale and start a new straditizer

Parameters **ask** (*bool*) – Whether to ask with a QMessageBox. If None, it defaults to the `straditize.widgets.StraditizerWidgers.always_yes`

rescale_plot(*percentage*)
Replot *im_rescale* after adjustments of the *slider*

property rescaling
Boolean that is true if one of the axes is rescaling

resize_stradi_image(*percentage*)
Resize the straditizer image

Parameters **percentage** (*float*) – A float between 0 and 100 specifying the target size of the straditize.straditizer.Straditizer.image

Returns The resized image of the current straditizer

Return type PIL.Image.Image

should_be_enabled(*w*)
Check if a widget should be enabled

This function checks if a given widget *w* from the widgets2disable attribute should be enabled or not

Parameters **w** (*QWidget*) – The widget to check

Returns True, if the widget should be enabled

Return type bool

slider = None
A matplotlib.widgets.Slider for specifying the size of the rescaled image

start_rescaling()
Create the rescaling figure

class straditize.widgets.image_correction.**ImageRotator**(*straditizer_widgets*, *item=None*, **args*, ***kwargs*)
Bases: *straditize.widgets.StraditizerControlBase*, PyQt5.QtWidgets.QWidget

Widget to rotate the image

This control mainly adds a QLineEdit *txt_rotate* to the *straditize.widgets.StraditizerWidgets* to rotate the image. It also enables the user to specify the rotation angle using two connected *CrossMarks*. Here the user can decide between a horizontal alignment (*btn_rotate_horizontal*) or a vertical alignment (*btn_rotate_vertical*)

Parameters

- **straditizer_widgets** (StraditizerWidgets) – The main widget for the straditizer GUI
- **item** (*QTreeWidgetItem*) – The parent item in the StraditizerWidgets.tree. If given, the setup_children() is called with this item

Attributes

angle	The rotation angle from txt_rotate as a float
btn_rotate_horizontal	A QPushButton for horizontal alignment
btn_rotate_vertical	A QPushButton for vertical alignment
txt_rotate	A QLineEdit to display the rotation angle

Methods

draw_figure()	

Continued on next page

Table 74 – continued from previous page

enable_or_disable_widgets(b)	Enable or disable the widgets in this control
remove_marks()	Remove the cross marks used for the rotation angle
rotate_image()	Rotate the image based on the specified `angle`
should_be_enabled(w)	Check if a widget should be enabled
start_horizontal_alignment()	Start the horizontal alignment
start_rotation()	Start the rotation (if not already started)
start_vertical_alignment()	Start the vertical alignment
update_txt_rotate(*args[, marks])	Update the `txt_rotate` from the displayed cross marks

property angle
The rotation angle from *txt_rotate* as a float

btn_rotate_horizontal = None
A QPushButton for horizontal alignment

btn_rotate_vertical = None
A QPushButton for vertical alignment

draw_figure()

enable_or_disable_widgets(*b*)
Enable or disable the widgets in this control

This method enables or disables the `widgets2disable` if the *should_be_enabled()* method evaluates to True

Parameters **b** (*bool*) – If True, enable the widgets, if False, disable them

remove_marks()
Remove the cross marks used for the rotation angle

rotate_image()
Rotate the image based on the specified *angle*

should_be_enabled(*w*)
Check if a widget should be enabled

This function checks if a given widget *w* from the `widgets2disable` attribute should be enabled or not

Parameters **w** (*QWidget*) – The widget to check

Returns True, if the widget should be enabled

Return type bool

start_horizontal_alignment()
Start the horizontal alignment

start_rotation()
Start the rotation (if not already started)

start_vertical_alignment()
Start the vertical alignment

txt_rotate = None
A QLineEdit to display the rotation angle

update_txt_rotate(**args*, *marks=[]*, ***kwargs*)
Update the *txt_rotate* from the displayed cross marks

straditize.widgets.marker_control module

A widget for controlling the appearance of markers

This module defines the *MarkerControl* to control the appearance and behaviour of *CrossMarks* instances in the `straditize.straditizer.Straditizer.marks` attribute

Disclaimer

Classes

ColorLabel([color])	A QTableWidget with one cell and no headers to just display a color
MarkerControl(straditizer_widgets, item, ...)	Widget to control the appearance of the marks

class straditize.widgets.marker_control.**ColorLabel**(*color='w'*, **args*, ***kwargs*)

Bases: `PyQt5.QtWidgets.QTableWidget`

A QTableWidget with one cell and no headers to just display a color

The color to display

Parameters **color** (*object*) – Either a QtGui.QColor object or a color that can be converted to RGBA using the `matplotlib.colors.to_rgba()` function

Methods

adjust_height()	Adjust the height to match the row height
select_color(*args)	Select a color using `PyQt5.QtWidgets.QColorDialog.getColor()`
set_color(color)	Set the color of the label
sizeHint()	Reimplemented to use the rowHeight as height

Attributes

color	QtCore.QColor.
color_changed(*args, **kwargs)	a signal that is emitted with an rgba color if the chosen color changes

adjust_height()

Adjust the height to match the row height

color = None

QtCore.QColor. The current color that is displayed

color_changed(**args*, ***kwargs*)
a signal that is emitted with an rgba color if the chosen color changes

select_color(**args*)
Select a color using `PyQt5.QtWidgets.QColorDialog.getColor()`

set_color(*color*)
Set the color of the label

This method sets the given *color* as background color for the cell and emits the *color_changed* signal

Parameters **color** (*object*) – Either a QtGui.QColor object or a color that can be converted to RGBA using the `matplotlib.colors.to_rgba()` function

sizeHint()
Reimplemented to use the rowHeight as height

class straditize.widgets.marker_control.**MarkerControl**(*straditizer_widgets*, *item*, **args*, ***kwargs*)
Bases: *straditize.widgets.StraditizerControlBase*, `PyQt5.QtWidgets.QWidget`

Widget to control the appearance of the marks

This widget controls the appearance of the *straditize.cross_mark.CrossMarks* instances in the `marks` attribute of the *straditizer*

Parameters

- **straditizer_widgets** (`StraditizerWidgets`) – The main widget for the straditizer GUI
- **item** (*QTreeWidgetItem*) – The parent item in the `StraditizerWidgets.tree`. If given, the `setup_children()` is called with this item

Methods

add_toolbar_widgets(mark)	Add the navigation actions to the toolbar
change_auto_hide(auto_hide)	Toggle the `auto_hide`
change_hline_draggable(state)	Enable or disable the dragging of horizontal lines
change_hline_selectable(state)	Enable or disable the selection of horizontal lines
change_line_style(i)	Change the line style of the marks
change_line_widths(lw)	Change the linewidth of the marks
change_marker_size(markersize)	Change the size of the markers
change_marker_style(i)	Change the marker style of the marks
change_select_colors(color)	Change the selection color of the marks
change_selection_line_widths(lw)	Change the linewidth for selected marks
change_show_connected_artists(show)	Change the visibility of connected artists
change_show_hlines(state)	Enable of disable the visibility of horizontal lines
change_show_vlines(state)	Enable of disable the visibility of vertical lines
change_unselect_colors(color)	Change the `straditize.cross_mark.CrossMark.cunselect` color
change_vline_draggable(state)	Enable or disable the dragging of vertical lines
change_vline_selectable(state)	Enable or disable the selection of vertical lines
draw_figs()	Draw the figures of the `marks`
enable_or_disable_widgets(b)	Renabled to use the `refresh()` method
fill_after_adding(mark)	
fill_from_mark(mark)	Set the widgets of this *MarkerControl* from a mark

Continued on next page

Table 78 – continued from previous page

fill_linestyles()	Fill the combo_line_style combobox
fill_markerstyles()	Fill the combo_marker_style combobox
go_to_greater_x_mark(x)	Move the plot to the next mark with a x-position greater than *x*
go_to_greater_y_mark(y)	Move the plot to the next mark with a y-position greater than *y*
go_to_left_mark()	Move the plot to the previous left cross mark
go_to_lower_mark()	Go to the next mark below the current y-limits
go_to_right_mark()	Move the plot to the next right cross mark
go_to_smaller_x_mark(x)	Move the plot to the next mark with a x-position smaller than *x*
go_to_smaller_y_mark(y)	Move the plot to the next mark with a y-position smaller than *x*
go_to_upper_mark()	Go to the next mark above the current y-limits
refresh()	Reimplemented to also set the properties of this widget
remove_actions()	Remove the navigation actions from the toolbar
set_line_style_item(ls)	Switch the combo_line_style to the given linestyle
set_marker_item(marker)	Switch the combo_marker_style to the given *marker*
should_be_enabled(w)	Check if a widget *w* should be enabled or disabled
update_mark(mark)	Update the properties of a mark to match the settings

Attributes

line_props	The properties of the lines as a dict
marks	The *CrossMarks* of the straditizer
select_props	The properties of selected marks as a dict

add_toolbar_widgets(*mark*)
Add the navigation actions to the toolbar

change_auto_hide(*auto_hide*)
Toggle the auto_hide

This method disables or enables the auto_hide of the marks

Parameters **auto_hide** (*bool or PyQt5.QtGui.Qt.Checked or PyQt5.QtGui.Qt.Unchecked*) – The value to use for the auto_hide. PyQt5.QtGui.Qt.Checked is equivalent to True

change_hline_draggable(*state*)
Enable or disable the dragging of horizontal lines

Parameters **state** (*Qt.Checked or Qt.Unchecked*) – If Qt.Checked, the horizontal lines can be dragged and dropped

change_hline_selectable(*state*)
Enable or disable the selection of horizontal lines

Parameters **state** (*Qt.Checked or Qt.Unchecked*) – If Qt.Checked, the horizontal lines can be selected

change_line_style(*i*)

Change the line style of the marks

Parameters **i** (*int*) – The index of the line style in the `line_styles` attribute to use

change_line_widths(*lw*)
Change the linewidth of the marks

Parameters **lw** (*float*) – The line width to use

change_marker_size(*markersize*)
Change the size of the markers

Parameters **markersize** (*float*) – The size of the marker to use

change_marker_style(*i*)
Change the marker style of the marks

Parameters **i** (*int*) – The index of the marker style in the `marker_styles` attribute to use

change_select_colors(*color*)
Change the selection color of the marks

Change the selection color of the marks to the given color.

Parameters **color** (*PyQt5.QtGui.QColor or a matplotlib color*) – The color to use

change_selection_line_widths(*lw*)
Change the linewidth for selected marks

Parameters **lw** (*float*) – The linewidth for selected marks

change_show_connected_artists(*show*)
Change the visibility of connected artists

Parameters **show** (*bool*) – The visibility for the *straditize.cross_mark.CrossMarks.set_connected_artists_visible()* method

change_show_hlines(*state*)
Enable of disable the visibility of horizontal lines

Parameters **state** (*Qt.Checked or Qt.Unchecked*) – If Qt.Checked, all horizontal lines are hidden

change_show_vlines(*state*)
Enable of disable the visibility of vertical lines

Parameters **state** (*Qt.Checked or Qt.Unchecked*) – If Qt.Checked, all vertical lines are hidden

change_unselect_colors(*color*)
Change the `straditize.cross_mark.CrossMark.cunselect` color

Change the unselection color of the marks to the given color.

Parameters **color** (*PyQt5.QtGui.QColor or a matplotlib color*) – The color to use

change_vline_draggable(*state*)
Enable or disable the dragging of vertical lines

Parameters **state** (*Qt.Checked or Qt.Unchecked*) – If Qt.Checked, the vertical lines can be dragged and dropped

change_vline_selectable(*state*)
Enable or disable the selection of vertical lines

Parameters **state** (*Qt.Checked or Qt.Unchecked*) – If Qt.Checked, the vertical lines can be selected

draw_figs()
Draw the figures of the *marks*

enable_or_disable_widgets(*b*)
Renabled to use the *refresh()* method

fill_after_adding(*mark*)

fill_from_mark(*mark*)
Set the widgets of this *MarkerControl* from a mark

This method sets the color labels, combo boxes, check boxes and text edits to match the properties of the given *mark*

Parameters **mark** (*straditize.cross_mark.CrossMark*) – The mark to use the attributes from

fill_linestyles()
Fill the combo_line_style combobox

fill_markerstyles()
Fill the combo_marker_style combobox

go_to_greater_x_mark(*x*)
Move the plot to the next mark with a x-position greater than *x*

Parameters **x** (*float*) – The reference x-position that shall be smaller than the new centered mark

go_to_greater_y_mark(*y*)
Move the plot to the next mark with a y-position greater than *y*

Parameters **y** (*float*) – The reference y-position that shall be smaller than the new centered mark

go_to_left_mark()
Move the plot to the previous left cross mark

go_to_lower_mark()
Go to the next mark below the current y-limits

go_to_right_mark()
Move the plot to the next right cross mark

go_to_smaller_x_mark(*x*)
Move the plot to the next mark with a x-position smaller than *x*

Parameters **x** (*float*) – The reference x-position that shall be greater than the new centered mark

go_to_smaller_y_mark(*y*)
Move the plot to the next mark with a y-position smaller than *x*

Parameters **y** (*float*) – The reference y-position that shall be smaller than the new centered mark

go_to_upper_mark()
Go to the next mark above the current y-limits

property line_props
The properties of the lines as a dict

property marks
The *CrossMarks* of the straditizer

refresh()
Reimplemented to also set the properties of this widget

remove_actions()
Remove the navigation actions from the toolbar

property select_props
The properties of selected marks as a dict

set_line_style_item(*ls*)
Switch the combo_line_style to the given linestyle

Parameters **ls** (*str*) – The matplotlib linestyle string

set_marker_item(*marker*)
Switch the combo_marker_style to the given *marker*

Parameters **marker** (*str*) – A matplotlib marker string

should_be_enabled(*w*)
Check if a widget *w* should be enabled or disabled

Parameters **w** (*PyQt5.QtWidgets.QWidget*) – The widget to (potentially) enable

Returns True if the straditizer of this instance has marks. Otherwise False

Return type bool

update_mark(*mark*)
Update the properties of a mark to match the settings

Parameters **mark** (straditize.cross_mark.CrossMarks) – The mark to update

straditize.widgets.menu_actions module

The main control widget for a straditizer

Disclaimer

Copyright (C) 2018-2019 Philipp S. Sommer

This program is free software: you can redistribute it and/or modify it under the terms of the GNU General Public License as published by the Free Software Foundation, either version 3 of the License, or (at your option) any later version.

This program is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU General Public License for more details.

You should have received a copy of the GNU General Public License along with this program. If not, see <https://www.gnu.org/licenses/>.

Classes

ExportDfDialog(df, straditizer[, fname])	A QDialog to export a pandas.DataFrame to Excel or CSV
StraditizerMenuActions(straditizer_widgets)	An object to control the main functionality of a Straditizer

class straditize.widgets.menu_actions.**ExportDfDialog**(*df*, *straditizer*, *fname=None*, **args*, ***kwargs*)

Bases: PyQt5.QtWidgets.QDialog

A QDialog to export a pandas.DataFrame to Excel or CSV

Parameters

- **df** (*pandas.DataFrame*) – The DataFrame to be exported
- **straditizer** (straditize.straditizer.Straditizer) – The source straditizer
- **fname** (*str*) – The file name to export to

Methods

accept(self)	
cancel()	
export_df(parent, df, straditizer[, fname, ...])	Open a dialog for exporting a DataFrame
get_open_file_name()	Ask the user for a filename for saving the data frame

accept(*self*)

cancel()

classmethod export_df(*parent*, *df*, *straditizer*, *fname=None*, *exec_=True*)
Open a dialog for exporting a DataFrame

Parameters **parent** (*QWidget*) – The parent widget

get_open_file_name()
Ask the user for a filename for saving the data frame

class straditize.widgets.menu_actions.**StraditizerMenuActions**(*straditizer_widgets*)

Bases: *straditize.widgets.StraditizerControlBase*

An object to control the main functionality of a Straditizer

This object is creates menu actions to load the straditizer **Attributes**

all_actions	save_actions, data_actions and text_actions
data_actions	The QActions to save the exported DataFrames and data images
save_actions	The QActions to save the straditizer, straditizer image, etc.
text_actions	The QActions to save the column names images
window_layout_action	The action to

Methods

create_sliders(stradi)	Create sliders to navigate in the given axes
export_final([fname])	Export the final results
export_full([fname])	Export the full digitized data
finish_loading(stradi)	Finish the opening of a straditizer
from_clipboard()	Open a straditizer from an Image in the clipboard

Continued on next page

Table 83 – continued from previous page

import_binary_image([fname])	Import the binary data reader image from an external file
import_data_image([fname])	Import the data reader image from an external file
import_full_image([fname])	Import the straditizer image from an external file
import_text_image(fname)	Import the column names reader image from an external file
open_straditizer([fname])	Open a straditizer from an image or project file
refresh()	Refresh from the straditizer
save_data_image([fname])	Save the binary image of the data reader
save_full_image([fname])	Save the image of the straditizer
save_straditizer()	Save the straditizer to a file
save_straditizer_as([fname])	Save the straditizer to a file
save_text_image([fname])	Save the image of the colnames reader
set_ax_xlim(ax_ref)	Define a function to update xlim from a given centered value
set_ax_ylim(ax_ref)	Define a function to update ylim from a given centered value
set_stradi_in_console()	Set the straditizer in the console of the GUI mainwindow
setup_children(item)	Setup the children for this control
setup_menu_actions(main)	Create the actions for the file menu
setup_shortcuts(main)	Setup the shortcuts when switched to the straditizer layout
stack_zoom_window()	Stack the magnifier image above the help explorer
update_x_navigation_sliders(ax)	Update the horizontal navigation slider for the given *ax‘*
update_y_navigation_sliders(ax)	Update the vertical navigation slider for the given *ax‘*

property all_actions
save_actions, *data_actions* and *text_actions*

create_sliders(*stradi*)
Create sliders to navigate in the given axes

Parameters **stradi** (*straditize.straditizer.Straditizer*) – The straditizer that just has been opened

data_actions = []
The QActions to save the exported DataFrames and data images

export_final(*fname=None*)
Export the final results

This method exports the *straditize.straditizer.Straditizer.final_df* of the current *straditizer*

Parameters **fname** (*str or None*) – The path of the target filename where to save the *df* (see the `ExportDialog.export_df()`)

export_full(*fname=None*)
Export the full digitized data

This method exports the *straditize.straditizer.Straditizer.full_df* of the current *straditizer*

Parameters **fname** (*str or None*) – The path of the target filename where to save the *df* (see the `ExportDialog.export_df()`)

finish_loading(*stradi*)
Finish the opening of a straditizer

This method sets the `straditizer.widgets.StraditizerWidgets.straditizer`, shows the straditizer image, creates the navigation sliders and sets the straditizer in the console

Parameters **stradi** (`straditize.straditizer.Straditizer`) – The straditizer that just has been opened

from_clipboard()
Open a straditizer from an Image in the clipboard

This method uses the `PIL.ImageGrab.grabclipboard()` function to open a new straditizer from the clipboard.

import_binary_image(*fname=None*)
Import the binary data reader image from an external file

This method imports the *straditize.binary.DataReader.binary* from an external file and sets it to the current *data_reader*.

Parameters **fname** (`str`, `PIL.Image.Image` or `None`) – The path of the image file or the `PIL.Image.Image`. If None, a QFileDialog is opened to request the file from the user.

import_data_image(*fname=None*)
Import the data reader image from an external file

This method imports the *straditize.binary.DataReader.image* from an external file and sets it to the current *data_reader*.

Parameters **fname** (`str`, `PIL.Image.Image` or `None`) – The path of the image file or the `PIL.Image.Image`. If None, a QFileDialog is opened to request the file from the user.

See also:

import_binary_image() To import the binary file

import_full_image(*fname=None*)
Import the straditizer image from an external file

This method imports the `straditize.straditizer.Straditizer.image` from an external file and sets it to the current straditizer.

Parameters **fname** (`str`, `PIL.Image.Image` or `None`) – The path of the image file or the `PIL.Image.Image`. If None, a QFileDialog is opened to request the file from the user.

import_text_image(*fname*)
Import the column names reader image from an external file

This method imports the *straditize.colnames.ColNamesReader.highres_image* from an external file and sets it to the current *colnames_reader*.

Parameters **fname** (`str`, `PIL.Image.Image` or `None`) – The path of the image file or the `PIL.Image.Image`. If None, a QFileDialog is opened to request the file from the user.

open_straditizer(*fname=None*, **args*, ***kwargs*)
Open a straditizer from an image or project file

Parameters **fname** (`str`, `PIL.Image.Image` or `None`) – The path to the file to import. If None, a QFileDialog is opened and the user is asked for a file name. The action then depends on the ending of `fname`:

'.nc' or '.nc4' we expect a netCDF file and open it with `xarray.open_dataset()` and load the straditizer with the *straditize.straditizer.Straditizer.from_dataset()* constructor

'.pkl' We expect a pickle file and load the straditizer with `pickle.load()`

any other ending We expect an image file and use the `PIL.Image.open()` function

At the end, the loading is finished with the *finish_loading()* method

refresh()
Refresh from the straditizer

save_actions = []
The QActions to save the straditizer, straditizer image, etc.

save_data_image(*fname=None*)
Save the binary image of the data reader

Save the *straditize.binary.DataReader.binary* of the current *data_reader*

Parameters **%(StraditizerMenuActions._save_image.parameters.fname)s** –

save_full_image(*fname=None*)
Save the image of the straditizer

Save the `straditize.straditizer.Straditizer.image` of the current *straditizer*

Parameters **%(StraditizerMenuActions._save_image.parameters.fname)s** –

save_straditizer()
Save the straditizer to a file

save_straditizer_as(*fname=None*)
Save the straditizer to a file

Parameters **fname** (str or `None`) – If None, a QFileDialog is opened and the user has to provide a filename. The final action then depends on the ending of the chose file:

'.pkl' We save the straditizer with `pickle.dump()`

else We use the *straditize.straditizer.Straditizer.to_dataset()* method and save the resulting dataset using the `xarray.Dataset.to_netcdf()` method

save_text_image(*fname=None*)
Save the image of the colnames reader

Save the *straditize.colnames.ColNamesReader.image* of the current *colnames_reader*

Parameters **fname** (*str or None*) – The path of the target filename where to save the *image*. If None, A QFileDialog is opened and we ask the user for a filename

static set_ax_xlim(*ax_ref*)
Define a function to update xlim from a given centered value

Parameters **ax_ref** (*matplotlib.axes.Axes*) – The axes whose x-limits to update when the returned function is called

Returns The function that can be called with a *val* to set the x-center of the *ax_ref*

Return type callable

static set_ax_ylim(*ax_ref*)
Define a function to update ylim from a given centered value

Parameters **ax_ref** (*matplotlib.axes.Axes*) – The axes whose y-limits to update when the returned function is called

Returns The function that can be called with a *val* to set the y-center of the *ax_ref*

Return type callable

set_stradi_in_console()
Set the straditizer in the console of the GUI mainwindow

This sets the current *straditizer* in psyplots `console` as the `stradi` variable

setup_children(*item*)
Setup the children for this control

This method is called to setup the children in the `StraditizerWidgets.tree`. By default, it just creates a child QTreeWidgetItem and sets this control as it's widget

Parameters **item** (*QTreeWidgetItem*) – The top level item in the `StraditizerWidgets.tree`

setup_menu_actions(*main*)
Create the actions for the file menu

Parameters **main** (*psyplot_gui.main.MainWindow*) – The mainwindow whose menubar shall be adapted

setup_shortcuts(*main*)
Setup the shortcuts when switched to the straditizer layout

Parameters **main** (*psyplot_gui.main.MainWindow*) – The psyplot mainwindow

stack_zoom_window()
Stack the magnifier image above the help explorer

text_actions = []
The QActions to save the column names images

static update_x_navigation_sliders(*ax*)
Update the horizontal navigation slider for the given *ax*'

Set the horizontal slider of the straditizer figure depending on the x-limits of it's corresponding axes

Parameters **ax** (*matplotlib.axes.Axes*) – The *straditize.straditizer.Straditizer.ax* attribute of a straditizer

static update_y_navigation_sliders(*ax*)
Update the vertical navigation slider for the given *ax*'

Set the vertical slider of the straditizer figure depending on the y-limits of it's corresponding axes

Parameters **ax** (*matplotlib.axes.Axes*) – The *straditize.straditizer.Straditizer.ax* attribute of a straditizer

window_layout_action = None
The action to *switch_to_straditizer_layout()*

straditize.widgets.pattern_selection module

A wdiget to select patterns in the image

The *PatternSelectionWidget* is used by the straditize.widget.selection_toolbar.SelectionToolbar to select patterns in the straditizer image

Disclaimer

Classes

EmbededMplCanvas([parent])	Ultimately, this is a QWidget (as well as a FigureCanvasAgg, etc.).
PatternSelectionWidget(arr, data_obj[, ...])	A wdiget to select patterns in the image

class straditize.widgets.pattern_selection.**EmbededMplCanvas**(*parent=None*, **args*, ***kwargs*)

Bases: matplotlib.backends.backend_qt5agg.FigureCanvasQTAgg

Ultimately, this is a QWidget (as well as a FigureCanvasAgg, etc.).

class straditize.widgets.pattern_selection.**PatternSelectionWidget**(*arr*, *data_obj*, *remove_selection=False*, **args*, ***kwargs*)

Bases: PyQt5.QtWidgets.QWidget, psyplot_gui.common.DockMixin

A wdiget to select patterns in the image

This widget consist of an *EmbededMplCanvas* to display the template for the pattern and uses the skimage.feature.match_template() function to identify it in the arr

See also:

straditize.widget.selection_toolbar.SelectionToolbar.start_pattern_selection

Attributes

selector	The selector to select the template in the original image
sl_thresh	A QSlider to set the threshold for the template correlation
template	The template to look for in the arr
template_extents	The extents of the template in the original image
template_fig	The *EmbededMplCanvas* to display the template
template_im	The matplotlib artist of the template in the

Methods

cancel()	
correlate_template(arr, template[, ...])	Correlate a template with the *arr*
maybe_tabify()	
modify_selection(i)	Modify the selection based on the correlation threshold
remove_plugin()	Remove this plugin and close it
start_correlation()	Look for the correlations of template and source
to_dock(main[, title, position, docktype])	
toggle_correlation_plot()	Toggle the correlation plot between `template` and `arr`
toggle_selection()	Modifiy the selection (or not) based on the template correlation
toggle_template_selection()	Enable or disable the template selection
update_image(*args, **kwargs)	Update the template image based on the `selector` extents

Parameters

- **arr** (np.ndarray of shape `(Ny, Nx)`) – The labeled selection array
- **data_obj** (`straditize.label_selection.LabelSelection`) – The data object whose image shall be selected
- **remove_selection** (*bool*) – If True, remove the selection on apply

axes = None

cancel()

correlate_template(*arr*, *template*, *fraction=False*, *increment=1*, *report=True*)

Correlate a template with the *arr*

This method uses the `skimage.feature.match_template()` function to find the given *template* in the source array *arr*.

Parameters

- **arr** (np.ndarray of shape `(Ny,Nx)`) – The labeled selection array (see `arr`), the source of the given *template*
- **template** (np.ndarray of shape `(nx, ny)`) – The template from `arr` that shall be searched
- **fraction** (*float*) – If not null, we will look through the given fraction of the template to look for partial matches as well
- **increment** (*int*) – The increment of the loop with the *fraction*.
- **report** (*bool*) – If True and *fraction* is not null, a QProgressDialog is opened to inform the user about the progress

key_press_cid = None

maybe_tabify()

modify_selection(*i*)

Modify the selection based on the correlation threshold

Parameters **i** (*int*) – An integer between 0 and 100, the value of the *sl_thresh* slider

remove_plugin()
Remove this plugin and close it

selector = None
The selector to select the template in the original image

sl_thresh = None
A QSlider to set the threshold for the template correlation

start_correlation()
Look for the correlations of template and source

template = None
The template to look for in the `arr`

template_extents = None
The extents of the *template* in the original image

template_fig = None
The *EmbededMplCanvas* to display the *template*

template_im = None
The matplotlib artist of the *template* in the *template_fig*

to_dock(*main*, *title=None*, *position=None*, *docktype='df'*, **args*, ***kwargs*)

toggle_correlation_plot()
Toggle the correlation plot between *template* and `arr`

toggle_selection()
Modifiy the selection (or not) based on the template correlation

toggle_template_selection()
Enable or disable the template selection

update_image(**args*, ***kwargs*)
Update the template image based on the *selector* extents

straditize.widgets.plots module

Plot control widgets for straditize

Classes

PlotControl(straditizer_widgets, item, ...)	A widget for controlling the plot
PlotControlTable(straditizer_widgets, *args, ...)	A widget to control the plots
ResultsPlot(straditizer_widgets)	A widget for plotting the final results

class straditize.widgets.plots.**PlotControl**(*straditizer_widgets*, *item*, **args*, ***kwargs*)
Bases: *straditize.widgets.StraditizerControlBase*, PyQt5.QtWidgets.QWidget

A widget for controlling the plot

This widgets holds a *PlotControlTable* to display visual diagnostics in the plot. Additionally it contains zoom buttons (*btn_view_global* and *btn_view_data*) and a widget to plot the results (*results_plot*) **Attributes**

btn_view_data	A button to zoom to the data
btn_view_global	A button to zoom out to the entire stratigraphic diagram
results_plot	A *ResultsPlot* to plot the digitized data in a new diagram
table	A *PlotControlTable* to display visual diagnostics

Methods

refresh()	Refresh from the straditizer
setup_children(item)	Setup the children for this control
zoom_data()	Zoom to the data part
zoom_global()	Zoom out to the full straditgraphic diagram

btn_view_data = None
A button to zoom to the data (see *zoom_data()*)

btn_view_global = None
A button to zoom out to the entire stratigraphic diagram (see *zoom_global()*)

refresh()
Refresh from the straditizer

results_plot = None
A *ResultsPlot* to plot the digitized data in a new diagram

setup_children(*item*)
Setup the children for this control

This method is called to setup the children in the StraditizerWidgets.tree. By default, it just creates a child QTreeWidgetItem and sets this control as it's widget

Parameters **item** (*QTreeWidgetItem*) – The top level item in the StraditizerWidgets.tree

table = None
A *PlotControlTable* to display visual diagnostics

zoom_data()
Zoom to the data part

See also:

straditize.straditizer.Straditizer.show_data_diagram()

zoom_global()
Zoom out to the full straditgraphic diagram

See also:

straditize.straditizer.Straditizer.show_full_image()

class straditize.widgets.plots.**PlotControlTable**(*straditizer_widgets*, **args*, ***kwargs*)

Bases: *straditize.widgets.StraditizerControlBase*, PyQt5.QtWidgets.QTableWidget

A widget to control the plots

This control widget is a table to plot, remove and toggle the visiblity of visual diagnostics for the straditizer. It has two columns: the first to toggle the visibility of the plot, the second to plot and remove the matplotlib artists. The vertical header are the items in the corresponding *plot_funcs*, *can_be_plotted_funcs* and/or *hide_funcs*.

Rows are added to this table using the *add_item()* method which stores the plotting functions in the *plot_funcs* and the functions to hide the plot in the *hide_funcs*. Whether an item can be plotted or not depends on the results of the corresponding callable in the *can_be_plotted_funcs*. **Methods**

add_item(what, get_artists[, plot_func, ...])	Add a plot object to the table
adjust_height()	
can_plot_column_starts()	Test whether the column starts can be visualized
can_plot_data_box()	Test whether the box around the diagram part can be plotted
can_plot_data_reader_color_image()	Test if the reader color image can be plotted
can_plot_full_df()	Test whether the full_df can be plotted
can_plot_potential_samples()	Test whether potential sample regions can be plotted
can_plot_sample_hlines()	Test whether the sample lines can be plotted
can_plot_samples()	Test whether the samples can be plotted
draw_figs(artists)	Draw the figures of the given *artists*
enable_or_disable_widgets(b)	b is ignored and is always set to True
get_column_start_lines()	Get the artists of the column starts
get_data_box()	Get the plotted straditize.straditizer.Straditizer.data_box
get_data_reader_background()	Get the *straditize.binary.DataReader.background*
get_data_reader_color_image()	Get the straditize.binary.DataReader.color_plot_im
get_data_reader_image()	Get the *straditize.binary.DataReader.plot_im*
get_full_df_lines()	Get the artists of the full_df plot
get_potential_samples_lines()	Get the artists of the plot of potential samples
get_sample_hlines()	Get the plotted the sample lines
get_samples_lines()	Get the artists of the plotted samples
get_straditizer_image()	Get the straditize.straditizer.Straditizer.plot_im
plot_column_starts()	Plot horizontal lines for the column starts
plot_data_box()	Plot the data box around the diagram part
plot_data_reader_color_image()	Plot the data reader color image
plot_full_df()	Plot the *full_df* of the reader
plot_potential_samples()	Highlight the regions with potential samples in the plot
plot_sample_hlines()	Plot the horizontal sample lines of the reader
plot_samples()	Plot the samples of the reader
refresh()	Refresh from the straditizer
remove_column_starts()	Remove the plotted lines of the column starts

Continued on next page

Table 90 – continued from previous page

remove_data_box()	Remove the box around the diagram part
remove_data_reader_color_image()	Remove the straditize.binary.DataReader.color_plot_im
remove_full_df_plot()	Remove the plot of the full_df
remove_potential_samples_plot()	Remove the plot of potential samples
remove_sample_hlines_plot()	Remove the sample lines
remove_samples_plot()	Remove the plotted samples
should_be_enabled(w)	Check if a widget should be enabled
sizeHint(self)	

Attributes

can_be_plotted_funcs	A mapping from plot identifier to a callable that returns True if
col_lines	Built-in mutable sequence.
hide_funcs	A mapping from plot identifier to a callable to hide the corresponding
plot_funcs	A mapping from plot identifier to a callable to plot the corresponding
widgets2disable	Built-in mutable sequence.

add_item(*what*, *get_artists*, *plot_func=None*, *remove_func=None*, *can_be_plotted=None*)
Add a plot object to the table

Parameters

- **what** (*str*) – The description of the plot object
- **get_artists** (*function*) – A function that takes no arguments and returns the artists
- **plot_func** (*function, optional*) – A function that takes no arguments and makes the plot.
- **remove_func** (*function, optional*) – A function that takes no arguments and removes the plot.
- **can_be_plotted** (*function, optional*) – A function that takes no argument and returns True if the plot can be made.

adjust_height()

can_be_plotted_funcs = {}
A mapping from plot identifier to a callable that returns True if the corresponding function in the *plot_funcs* mapping can be called

can_plot_column_starts()
Test whether the column starts can be visualized

See also:

plot_column_starts()

can_plot_data_box()
Test whether the box around the diagram part can be plotted

See also:

plot_data_box()

can_plot_data_reader_color_image()
Test if the reader color image can be plotted

See also:

plot_data_reader_color_image()

can_plot_full_df()
Test whether the full_df can be plotted

See also:

plot_full_df()

can_plot_potential_samples()
Test whether potential sample regions can be plotted

See also:

plot_potential_samples()

can_plot_sample_hlines()
Test whether the sample lines can be plotted

See also:

plot_sample_hlines()

can_plot_samples()
Test whether the samples can be plotted

See also:

plot_samples()

col_lines = []

draw_figs(*artists*)
Draw the figures of the given *artists*

Parameters **artists** (list of matplotlib.artist.Artist) – The artists to draw the canvas from

enable_or_disable_widgets(*b*)
b is ignored and is always set to True

get_column_start_lines()
Get the artists of the column starts

See also:

plot_column_starts()

get_data_box()
Get the plotted straditize.straditizer.Straditizer.data_box

See also:

plot_data_box()

get_data_reader_background()
Get the *straditize.binary.DataReader.background*

get_data_reader_color_image()
Get the straditize.binary.DataReader.color_plot_im

See also:

plot_data_reader_color_image()

get_data_reader_image()
Get the *straditize.binary.DataReader.plot_im*

get_full_df_lines()
Get the artists of the full_df plot

See also:

plot_full_df()

get_potential_samples_lines()
Get the artists of the plot of potential samples

See also:

plot_potential_samples()

get_sample_hlines()
Get the plotted the sample lines

See also:

plot_sample_hlines()

get_samples_lines()
Get the artists of the plotted samples

See also:

plot_samples()

get_straditizer_image()
Get the `straditize.straditizer.Straditizer.plot_im`

hide_funcs = {}
A mapping from plot identifier to a callable to hide the corresponding artists

plot_column_starts()
Plot horizontal lines for the column starts

See also:

remove_column_starts(), *get_column_start_lines()*, *can_plot_column_starts()*

plot_data_box()
Plot the data box around the diagram part

See also:

straditize.straditizer.Straditizer.draw_data_box(), *remove_data_box()*, *get_data_box()*, *can_plot_data_box()*

plot_data_reader_color_image()
Plot the data reader color image

See also:

straditize.binary.DataReader.plot_color_image(), *remove_data_reader_color_image()*, *can_plot_data_reader_color_image()*

plot_full_df()
Plot the *full_df* of the reader

See also:

straditize.binary.DataReader.plot_full_df(), *remove_full_df_plot()*, *get_full_df_lines()*, *can_plot_full_df()*

plot_funcs = {}
A mapping from plot identifier to a callable to plot the corresponding artists

plot_potential_samples()
Highlight the regions with potential samples in the plot

See also:

straditize.binary.DataReader.plot_potential_samples(), *remove_potential_samples_plot()*, *get_potential_samples_lines()*, *can_plot_potential_samples()*

plot_sample_hlines()
Plot the horizontal sample lines of the reader

See also:

straditize.binary.DataReader.plot_sample_hlines(), *remove_sample_hlines_plot()*, *get_sample_hlines()*, *can_plot_sample_hlines()*

plot_samples()
Plot the samples of the reader

See also:

straditize.binary.DataReader.plot_samples(), *remove_samples_plot()*, *get_samples_lines()*, *can_plot_samples()*

refresh()
Refresh from the straditizer

remove_column_starts()
Remove the plotted lines of the column starts

See also:

plot_column_starts()

remove_data_box()
Remove the box around the diagram part

See also:

plot_data_box()

remove_data_reader_color_image()
Remove the `straditize.binary.DataReader.color_plot_im`

See also:

plot_data_reader_color_image()

remove_full_df_plot()
Remove the plot of the full_df

See also:

plot_full_df()

remove_potential_samples_plot()
Remove the plot of potential samples

See also:

plot_potential_samples()

remove_sample_hlines_plot()
Remove the sample lines

See also:

plot_sample_hlines()

remove_samples_plot()
Remove the plotted samples

See also:

plot_samples()

should_be_enabled(*w*)
Check if a widget should be enabled

This function checks if a given widget *w* from the *widgets2disable* attribute should be enabled or not

Parameters **w** (*QWidget*) – The widget to check

Returns True, if the widget should be enabled

Return type bool

sizeHint(*self*) → QSize

property widgets2disable
Built-in mutable sequence.

If no argument is given, the constructor creates a new empty list. The argument must be an iterable if specified.

class straditize.widgets.plots.**ResultsPlot**(*straditizer_widgets*)
Bases: *straditize.widgets.StraditizerControlBase*

A widget for plotting the final results

This widgets contains a QPushButton *btn_plot* to plot the results using the *straditize.binary.DataReader.plot_results()* method **Attributes**

btn_plot	The QPushButton to call the plot_results() method
cb_final	A QCheckBox whether the samples or the full digitized data shall be
cb_transformed	A QCheckBox whether x- and y-axis should be translated from pixel to

Methods

plot_results()	Plot the results
refresh()	Refresh from the straditizer
setup_children(item)	Setup the children for this control

btn_plot = None
The QPushButton to call the *plot_results()* method

cb_final = None
A QCheckBox whether the samples or the full digitized data shall be plotted

cb_transformed = None
A QCheckBox whether x- and y-axis should be translated from pixel to data units

plot_results()
Plot the results

What is plotted depends on the *cb_transformed* and the *cb_final*

***cb_transformed* and *cb_final* are checked** Plot the *straditize.straditizer.Straditizer.final_df*

cb_transformed* is checked but not *cb_final Plot the *straditize.straditizer.Straditizer.full_df*

cb_transformed* is not checked but *cb_final Plot the *straditize.binary.DataReader.sample_locs*

***cb_transformed* and *cb_final* are both not checked** Plot the *straditize.binary.DataReader.full_df*

refresh()
Refresh from the straditizer

setup_children(*item*)
Setup the children for this control

This method is called to setup the children in the `StraditizerWidgets.tree`. By default, it just creates a child QTreeWidgetItem and sets this control as it's widget

Parameters **item** (*QTreeWidgetItem*) – The top level item in the `StraditizerWidgets.tree`

straditize.widgets.progress_widget module

Progress widget for the straditization

The ProgressWidget defined here is a ListWidget to show the current state of the straditititization.

Disclaimer

Classes

ColumnNamesTask(parent)	Task to specify the column names
ColumnsTask(parent)	A task to separate columns
DataLimitsTask(parent)	The task to check the data limits
DataReaderTask(parent)	A task for initializing the reader
DigitizeTask(parent)	Task to digitize the data

Continued on next page

Table 94 – continued from previous page

ExportTask(parent)	Task to export the final results
InitStraditizerTask(parent)	The task to initialize a straditizer
OccurencesTask(parent)	Task to handle occurences
ProgressTask(parent)	The base class for an item that should be shown in the ProgressWidget
ProgressWidget(straditizer_widgets, item)	A widget to show the progress of the straditization
RemoveArtifactsTask(parent)	Task to clean the binary image
RemoveLinesTask(parent)	Task to remove y-axes, x-axes, horizontal and vertical lines
SamplesTask(parent)	Task to edit and find samples
SaveProjectTask(*args, **kwargs)	Task to remember saving the project
SelectExaggerationsTask(parent)	Task to handle exaggerations
XTranslationTask(parent)	Task to specify the x-axes conversion from pixel to data units
YTranslationTask(parent)	Task to specify the y-axis conversion from pixel to data units

class straditize.widgets.progress_widget.**ColumnNamesTask**(*parent*)

Bases: *straditize.widgets.progress_widget.ProgressTask*

Task to specify the column names

Parameters **parent** (*QListWidget*) – The list that holds this item

Attributes

dependencies	Built-in mutable sequence.
done_tooltip	
is_finished	A property that is True, when the task is finished
name	str(object='') -> str
rst_file	str(object='') -> str
summary	str(object='') -> str
task_tooltip	str(object='') -> str

dependencies = ['columns']

property done_tooltip

property is_finished

A property that is True, when the task is finished

name = 'column_names'

rst_file = 'column_names'

summary = 'Specify the column names'

task_tooltip = 'Click the Edit column names button to insert the names for each

class straditize.widgets.progress_widget.**ColumnsTask**(*parent*)

Bases: *straditize.widgets.progress_widget.ProgressTask*

A task to separate columns

Parameters **parent** (*QListWidget*) – The list that holds this item

Attributes

dependencies	Built-in mutable sequence.
done_tooltip	
is_finished	A property that is True, when the task is finished
name	str(object='') -> str
rst_file	str(object='') -> str
summary	str(object='') -> str
task_tooltip	str(object='') -> str

dependencies = ['init_reader']

property done_tooltip

property is_finished
A property that is True, when the task is finished

name = 'columns'

rst_file = 'select_column_starts'

summary = 'Separate the columns'

task_tooltip = 'Separate the columns (subdiagrams) of the stratigraphic diagram'

class straditize.widgets.progress_widget.**DataLimitsTask**(*parent*)
Bases: *straditize.widgets.progress_widget.ProgressTask*

The task to check the data limits

Parameters **parent** (*QListWidget*) – The list that holds this item

Attributes

dependencies	Built-in mutable sequence.
done_tooltip	
is_finished	A property that is True, when the task is finished
name	str(object='') -> str
rst_file	str(object='') -> str
summary	str(object='') -> str
task_tooltip	str(object='') -> str

dependencies = ['init_stradi']

property done_tooltip

property is_finished
A property that is True, when the task is finished

name = 'datalim'

rst_file = 'select_data_part'

summary = 'Limits of the diagram'

task_tooltip = 'Specify the corners for the data part of the diagram by clicking the <

class straditize.widgets.progress_widget.**DataReaderTask**(*parent*)
Bases: *straditize.widgets.progress_widget.ProgressTask*

A task for initializing the reader

Parameters **parent** (*QListWidget*) – The list that holds this item

Attributes

dependencies	Built-in mutable sequence.
done_tooltip	
is_finished	A property that is True, when the task is finished
name	str(object='') -> str
rst_file	str(object='') -> str
summary	str(object='') -> str
task_tooltip	str(object='') -> str

dependencies = ['datalim']

property done_tooltip

property is_finished
A property that is True, when the task is finished

name = 'init_reader'

rst_file = 'select_reader'

summary = 'Initialize the diagram reader'

task_tooltip = 'Choose the appropriate reader type and click the Convert image l

class straditize.widgets.progress_widget.**DigitizeTask**(*parent*)
Bases: *straditize.widgets.progress_widget.ProgressTask*

Task to digitize the data

Parameters **parent** (*QListWidget*) – The list that holds this item

Attributes

dependencies	Built-in mutable sequence.
done_tooltip	
is_finished	A property that is True, when the task is finished
name	str(object='') -> str
rst_file	str(object='') -> str
summary	str(object='') -> str
task_tooltip	str(object='') -> str

dependencies = ['remove_artifacts', 'remove_lines']

property done_tooltip

property is_finished
A property that is True, when the task is finished

name = 'digitize'

rst_file = 'digitize'

summary = 'Digitize the diagram'

task_tooltip = 'Click the `Digitize` button to digitize the diagram'

class straditize.widgets.progress_widget.**ExportTask**(*parent*)
Bases: *straditize.widgets.progress_widget.ProgressTask*

Task to export the final results

Parameters **parent** (*QListWidget*) – The list that holds this item

Attributes

dependencies	Built-in mutable sequence.
done_tooltip	
is_finished	A property that is True, when the task is finished
name	str(object='') -> str
rst_file	str(object='') -> str
summary	str(object='') -> str
task_tooltip	str(object='') -> str

dependencies = ['samples']

property done_tooltip

property is_finished
A property that is True, when the task is finished

name = 'export'

rst_file = 'export'

summary = 'Export the final data'

task_tooltip = 'Export the final data'

class straditize.widgets.progress_widget.**InitStraditizerTask**(*parent*)
Bases: *straditize.widgets.progress_widget.ProgressTask*

The task to initialize a straditizer

Parameters **parent** (*QListWidget*) – The list that holds this item

Attributes

done_tooltip	
is_finished	A property that is True, when the task is finished
name	str(object='') -> str
rst_file	str(object='') -> str
summary	str(object='') -> str
task_tooltip	str(object='') -> str

property done_tooltip

property is_finished
A property that is True, when the task is finished

name = 'init_stradi'

rst_file = 'load_image'

summary = 'Load an image or project'

task_tooltip = 'Load a straditizer image or project to get started'

class straditize.widgets.progress_widget.**OccurencesTask**(*parent*)
Bases: *straditize.widgets.progress_widget.ProgressTask*

Task to handle occurences

Parameters **parent** (*QListWidget*) – The list that holds this item

Attributes

dependencies	Built-in mutable sequence.
done_tooltip	
is_finished	A property that is True, when the task is finished
name	str(object='') -> str
rst_file	str(object='') -> str
summary	str(object='') -> str
task_tooltip	str(object='') -> str

dependencies = ['columns']

property done_tooltip

property is_finished

A property that is True, when the task is finished

name = 'occurences'

rst_file = 'occurences'

summary = 'Select occurence markers'

task_tooltip = 'Pollen diagrams often have markers for low taxon percentages to show t

class straditize.widgets.progress_widget.**ProgressTask**(*parent*)

Bases: PyQt5.QtWidgets.QListWidgetItem

The base class for an item that should be shown in the ProgressWidget

Parameters **parent** (*QListWidget*) – The list that holds this item

Attributes

colnames_reader	The column names reader of the straditizer
data_reader	The data reader of the straditizer
dependencies	List of `name` attributes that this task is depending on
dependencies_tasks	
done	True if `is_finished` or `done_by_user`
done_by_user	boolean that is True, when the task is marked as done by the user
done_tooltip	The tooltip when the straditizer is done.
is_finished	A property that is True, when the task is finished
is_ready	Boolean that is True if the task is ready to be solved
name	The name of the task
progress_widget	The progress widget that shows this task
rst_file	rst file that should be displayed on double click. The filename shuould
state	The state of the task
straditizer	The straditizer of the GUI
summary	The summary of the task that is shown in the progress widget
task_tooltip	The tooltip of the item
try_finished	Same as `is_finished` but catches every exception

Methods

refresh()	

property colnames_reader
The column names reader of the straditizer

property data_reader
The data reader of the straditizer

dependencies = []
List of *name* attributes that this task is depending on

property dependencies_tasks

property done
True if *is_finished* or *done_by_user*

property done_by_user
boolean that is True, when the task is marked as done by the user

done_tooltip = None
The tooltip when the straditizer is done. If None, the *task_tooltip* is used

property is_finished
A property that is True, when the task is finished

is_ready = True
Boolean that is True if the task is ready to be solved

name = ''
The name of the task

property progress_widget
The progress widget that shows this task

refresh()

rst_file = None
rst file that should be displayed on double click. The filename shuould be without .rst ending.

property state
The state of the task

property straditizer
The straditizer of the GUI

summary = ''
The summary of the task that is shown in the progress widget

task_tooltip = ''
The tooltip of the item

property try_finished
Same as *is_finished* but catches every exception

class straditize.widgets.progress_widget.**ProgressWidget**(*straditizer_widgets*, *item*)
Bases: PyQt5.QtWidgets.QWidget, *straditize.widgets.StraditizerControlBase*

A widget to show the progress of the straditization **Attributes**

combo_display	A QComboBox to select which tasks to display (todo, done, not yet ready

Continued on next page

Table 105 – continued from previous page

info_label	A QLabel to display the tooltip of the selected task
progress_list	A QListWidget to display the *ProgressTask* instances

Methods

contextMenuEvent(event)	Reimplement Qt method
populate_list()	Populate the progress_list
refresh()	Refresh from the straditizer
setup_menu()	Set up the context menu
show_rst(item[, old])	Show the documentation corresponding to a :class:‘ProgressTask
toggle_done_by_user()	
update_info_label(item[, old])	Update the info_label from a *ProgressTask*

combo_display = None
A QComboBox to select which tasks to display (todo, done, not yet ready or all tasks)

contextMenuEvent(*event*)
Reimplement Qt method

info_label = None
A QLabel to display the tooltip of the selected task

populate_list()
Populate the *progress_list*

This method adds instances of the *ProgressTask* class (or it’s subclasses) to the the *progress_list*

progress_list = None
A QListWidget to display the *ProgressTask* instances

refresh()
Refresh from the straditizer

setup_menu()
Set up the context menu

show_rst(*item*, *old=None*)
Show the documentation corresponding to a :class:‘ProgressTask

Parameters

- **item** (ProgressTask) – The task to display it’s rst_file in the psyplot_gui.main.MainWindow.help_explorer
- **old** (ProgressTask) – The old task that has been selected previously (this parameter is ignored)

toggle_done_by_user()

update_info_label(*item*, *old=None*)
Update the *info_label* from a *ProgressTask*

Parameters

- **item** (ProgressTask) – The selected task whose tooltip the *info_label* shall display

- **old** (ProgressTask) – The old task that has been selected previously (this parameter is ignored)

class straditize.widgets.progress_widget.**RemoveArtifactsTask**(*parent*)

Bases: *straditize.widgets.progress_widget.ProgressTask*

Task to clean the binary image

Parameters **parent** (*QListWidget*) – The list that holds this item

Attributes

dependencies	Built-in mutable sequence.
done_tooltip	
is_finished	A property that is True, when the task is finished
name	str(object='') -> str
rst_file	str(object='') -> str
summary	str(object='') -> str
task_tooltip	str(object='') -> str

dependencies = ['columns']

property done_tooltip

property is_finished

A property that is True, when the task is finished

name = 'remove_artifacts'

rst_file = 'removing_features'

summary = 'Clean the diagram image'

task_tooltip = 'Remove all artifacts in the diagram part that do not represent data. Ma

class straditize.widgets.progress_widget.**RemoveLinesTask**(*parent*)

Bases: *straditize.widgets.progress_widget.RemoveArtifactsTask*

Task to remove y-axes, x-axes, horizontal and vertical lines

Parameters **parent** (*QListWidget*) – The list that holds this item

Attributes

dependencies	Built-in mutable sequence.
done_tooltip	
is_finished	A property that is True, when the task is finished
name	str(object='') -> str
rst_file	str(object='') -> str
summary	str(object='') -> str
task_tooltip	str(object='') -> str

dependencies = ['columns']

property done_tooltip

property is_finished

A property that is True, when the task is finished

name = 'remove_lines'

rst_file = 'remove_lines'

summary = 'Remove y-, x-axes and other lines'

task_tooltip = 'In order to clean the diagram part, remove all vertical and horizontal

class straditize.widgets.progress_widget.**SamplesTask**(*parent*)

Bases: *straditize.widgets.progress_widget.ProgressTask*

Task to edit and find samples

Parameters **parent** (*QListWidget*) – The list that holds this item

Attributes

dependencies	Built-in mutable sequence.
done_tooltip	
is_finished	A property that is True, when the task is finished
name	str(object='') -> str
rst_file	str(object='') -> str
summary	str(object='') -> str
task_tooltip	str(object='') -> str

dependencies = ['digitize']

property done_tooltip

property is_finished

A property that is True, when the task is finished

name = 'samples'

rst_file = 'samples'

summary = 'Find and edit the samples'

task_tooltip = 'Find and edit the samples using the `Edit samples` menu'

class straditize.widgets.progress_widget.**SaveProjectTask**(**args*, ***kwargs*)

Bases: *straditize.widgets.progress_widget.ProgressTask*

Task to remember saving the project

Parameters **parent** (*QListWidget*) – The list that holds this item

Attributes

dependencies	Built-in mutable sequence.
is_finished	True if the project was saved less than 5 minutes ago
name	str(object='') -> str
rst_file	str(object='') -> str
summary	str(object='') -> str
task_tooltip	str(object='') -> str

Methods

refresh_tooltip()	
tooltip()	

dependencies = ['init_stradi']

property is_finished
True if the project was saved less than 5 minutes ago

name = 'save_project'

refresh_tooltip()

rst_file = 'save_and_load'

summary = 'Save the straditizer'

property task_tooltip
str(object=’’) -> str str(bytes_or_buffer[, encoding[, errors]]) -> str

Create a new string object from the given object. If encoding or errors is specified, then the object must expose a data buffer that will be decoded using the given encoding and error handler. Otherwise, returns the result of object.__str__() (if defined) or repr(object). encoding defaults to sys.getdefaultencoding(). errors defaults to ‘strict’.

tooltip()

class straditize.widgets.progress_widget.**SelectExaggerationsTask**(*parent*)
Bases: *straditize.widgets.progress_widget.ProgressTask*

Task to handle exaggerations

Parameters **parent** (*QListWidget*) – The list that holds this item

Attributes

dependencies	Built-in mutable sequence.
done_tooltip	
is_finished	A property that is True, when the task is finished
name	str(object=’’) -> str
rst_file	str(object=’’) -> str
summary	str(object=’’) -> str
task_tooltip	str(object=’’) -> str

dependencies = ['columns']

property done_tooltip

property is_finished
A property that is True, when the task is finished

name = 'exag'

rst_file = 'exaggerations'

summary = 'Select exagerations'

task_tooltip = 'Pollen diagrams often display an exaggerated value of of the taxon per

class straditize.widgets.progress_widget.**XTranslationTask**(*parent*)
Bases: *straditize.widgets.progress_widget.ProgressTask*

Task to specify the x-axes conversion from pixel to data units

Parameters **parent** (*QListWidget*) – The list that holds this item

Attributes

dependencies	Built-in mutable sequence.
done_tooltip	
is_finished	A property that is True, when the task is finished
name	str(object='') -> str
rst_file	str(object='') -> str
summary	str(object='') -> str
task_tooltip	str(object='') -> str

dependencies = ['columns']

property done_tooltip

property is_finished
A property that is True, when the task is finished

name = 'xaxes_trans'

rst_file = 'xaxis_translation'

summary = 'Transform the x-axes'

property task_tooltip
str(object='') -> str str(bytes_or_buffer[, encoding[, errors]]) -> str

Create a new string object from the given object. If encoding or errors is specified, then the object must expose a data buffer that will be decoded using the given encoding and error handler. Otherwise, returns the result of object.__str__() (if defined) or repr(object). encoding defaults to sys.getdefaultencoding(). errors defaults to 'strict'.

class straditize.widgets.progress_widget.**YTranslationTask**(*parent*)
Bases: *straditize.widgets.progress_widget.ProgressTask*

Task to specify the y-axis conversion from pixel to data units

Parameters **parent** (*QListWidget*) – The list that holds this item

Attributes

dependencies	Built-in mutable sequence.
done_tooltip	
is_finished	A property that is True, when the task is finished
name	str(object='') -> str
rst_file	str(object='') -> str
summary	str(object='') -> str
task_tooltip	str(object='') -> str

dependencies = ['init_reader']

property done_tooltip

property is_finished
A property that is True, when the task is finished

name = 'yaxis_trans'

rst_file = 'yaxis_translation'

summary = 'Transform the y-axis'

task_tooltip = 'Transform the y-axis from pixel to data units'

straditize.widgets.samples_table module

A table for manipulating samples

This module defines the sample editors, either for editing the samples in the straditizer image (*SingleCrossMarksEditor*) or in a separate Figure (*MultiCrossMarksEditor*)

Disclaimer

Classes

MultiCrossMarksEditor(straditizer[, axes])	An editor for cross marks in multiple axes
MultiCrossMarksModel(marks, columns, straditizer)	A table model to handle multiple connected cross marks in different axes
MultiCrossMarksView(marks, full_df, *args, ...)	A table view set up by cross marks from multiple axes
SingleCrossMarksEditor(straditizer[, axes])	The editor for cross marks on a single axes
SingleCrossMarksModel(*args, **kwargs)	A table model to handle cross marks within one single axis
SingleCrossMarksView(marks, full_df, *args, ...)	A table for visualizing marks from a single axes

class straditize.widgets.samples_table.**MultiCrossMarksEditor**(*straditizer*, *axes=None*, **args*, ***kwargs*)

Bases: psyplot_gui.common.DockMixin, PyQt5.QtWidgets.QWidget

An editor for cross marks in multiple axes

Parameters

- **straditizer** (*weakref.ref*) – The reference to the straditizer
- **axes** (*matplotlib.axes.Axes*) – The matplotlib axes corresponding to the marks

Attributes

cb_plot_lines	Plot the reconstructed data
straditizer	A weakref to the

Methods

create_view([axes])	Create the *MultiCrossMarksView* of the editor
maybe_show_selection_only()	
maybe_tabify()	

Continued on next page

Table 117 – continued from previous page

maybe_zoom_to_selection()	
save_samples()	Save the samples to the straditizer without removing them
to_dock(main[, title, position, docktype])	
toggle_cb_selection_only()	
toggle_cb_zoom_to_selection()	
toggle_fit2selection()	Enable the fitting so selected digitized data
toggle_fmt_button(text)	
toggle_plot_lines()	
update_format()	Update the format of the table

Classes

dock_cls	The QDockWidget for the DataFrameEditor

cb_plot_lines = None
Plot the reconstructed data

create_view(*axes=None*)
Create the *MultiCrossMarksView* of the editor

Parameters **axes** (list of matplotlib.axes.Axes) – The matplotlib axes for the marks

dock_cls
alias of psyplot_gui.dataframeeditor.DataFrameDock

maybe_show_selection_only()

maybe_tabify()

maybe_zoom_to_selection()

save_samples()
Save the samples to the *straditizer* without removing them

straditizer = None
A weakref to the *straditizer*

to_dock(*main*, *title=None*, *position=None*, *docktype='df'*, **args*, ***kwargs*)

toggle_cb_selection_only()

toggle_cb_zoom_to_selection()

toggle_fit2selection()
Enable the fitting so selected digitized data

toggle_fmt_button(*text*)

toggle_plot_lines()

update_format()
Update the format of the table

class straditize.widgets.samples_table.**MultiCrossMarksModel**(*marks*, *columns*, *straditizer*, *axes=None*, *occurences_value=-9999*)

Bases: PyQt5.QtCore.QAbstractTableModel

A table model to handle multiple connected cross marks in different axes

Parameters

- **marks** (list of *straditize.cross_mark.CrossMarks*) – the initial marks
- **columns** (*list of str*) – the column names to use
- **straditizer** (straditize.straditizer.Straditizer) – The straditizer that manages the *marks*
- **axes** (list of matplotlib.axes.Axes) – The matplotlib axes that contain the *marks*
- **occurences_value** (*float*) – The value that marks an occurence

Methods

columnCount([index])	The number of rows in the table
data(self, QModelIndex, role)	
delRow(irow)	
flags(index)	Set flags
get_cell_mark(row, column)	Get the mark for a given cell in the table
get_format()	Return current format
headerData(section, orientation[, role])	Set header data
insertRow(irow[, xa, ya])	Insert a row into the table
load_new_marks(mark)	Add a new mark into the table after they have been added by the user
plot_lines()	Connect the samples through visual `lines`
remove_lines()	Remove the `lines`
remove_mark(mark)	Remove a mark from the table after it has been removed by the user
reset()	Reset the model
rowCount([index])	The number of rows in the table
setData(index, value[, role, change_type])	Cell content change
set_format(fmt)	Change display format
set_marks(marks, columns)	Set the `marks` attribute from the given *columns*
sort_marks()	Sort the marks based on there y-position
update_after_move(old_pos, mark)	
update_lines()	Update the `lines` or plot them

Attributes

df	Get the samples_locs
fig	The figure of the cross marks
iter_marks	Iter over all marks in the `marks` attribute
lines	A list of matplotlib.lines.Line2D that connects the cross marks
marks	A list `[float, list]` where the first `float` is the vertical

columnCount(*index=<PyQt5.QtCore.QModelIndex object>*)
The number of rows in the table

data(*self*, *QModelIndex*, *role: int = Qt.DisplayRole*) → Any

delRow(*irow*)

property df
Get the samples_locs

property fig
The figure of the cross marks

flags(*index*)
Set flags

get_cell_mark(*row*, *column*)
Get the mark for a given cell in the table

Parameters

- **row** (*int*) – The row of the cell
- **column** (*int*) – The column of the cell

Returns The corresponding mark from the *marks* attribute

Return type *straditize.cross_mark.CrossMarks*

get_format()
Return current format

headerData(*section*, *orientation*, *role=0*)
Set header data

insertRow(*irow*, *xa=None*, *ya=None*)
Insert a row into the table

Parameters **irow** (*int*) – The row index. If *irow* is equal to the length of the *marks*, the rows will be appended

property iter_marks
Iter over all marks in the *marks* attribute

lines = []
A list of `matplotlib.lines.Line2D` that connects the cross marks and plots a reconstruction based on them

load_new_marks(*mark*)
Add a new mark into the table after they have been added by the user

Parameters **mark** (`straditize.cross_mark.CrossMarks`) – The added mark

marks = []
A list `[float, list]` where the first `float` is the vertical position and the second `list` is a list of the corresponding *CrossMarks* instances

plot_lines()
Connect the samples through visual *lines*

remove_lines()
Remove the *lines*

remove_mark(*mark*)
Remove a mark from the table after it has been removed by the user

Parameters **mark** (`straditize.cross_mark.CrossMarks`) – The removed mark

reset()
Reset the model

rowCount(*index=<PyQt5.QtCore.QModelIndex object>*)
The number of rows in the table

setData(*index*, *value*, *role=2*, *change_type=None*)
Cell content change

set_format(*fmt*)
Change display format

set_marks(*marks*, *columns*)
Set the *marks* attribute from the given *columns*

Parameters

- **marks** (list of *straditize.cross_mark.CrossMarks*) – the initial marks
- **columns** (*list of str*) – the column names to use

sort_marks()
Sort the marks based on there y-position

update_after_move(*old_pos*, *mark*)

update_lines()
Update the *lines* or plot them

See also:

plot_lines()

class straditize.widgets.samples_table.**MultiCrossMarksView**(*marks*, *full_df*, **args*, ***kwargs*)

Bases: PyQt5.QtWidgets.QTableView

A table view set up by cross marks from multiple axes

The model for this table is the *MultiCrossMarksModel*

Parameters

- **marks** (list of *straditize.cross_mark.CrossMarks*) – the initial marks
- **full_df** (*pandas.DataFrame*) – The data fame of the full digitized data

Methods

contextMenuEvent(event)	Reimplement Qt method
delete_selected_rows()	
fit2data()	Fit the selected cells to the full_df
init_model(marks, *args, **kwargs)	Initialize the table *MultiCrossMarksModel*
insert_row_above_selection()	Insert a row above the selection
insert_row_below_selection()	Insert a row below the selection
moveCursor(cursor_action, modifiers)	Update the table position.
resizeEvent(event)	Update the frozen column dimensions.
scrollTo(index, hint)	Scroll the table.
setup_menu()	Setup context menu
show_all_marks()	Show all marks
show_selected_marks_only()	Show only the marks selected in the table
update_section_height(logical_index, ...)	Update the vertical width of the frozen column when a

Continued on next page

Table 121 – continued from previous page

update_section_width(logical_index, ...)	Update the horizontal width of the frozen column when a
zoom_to_cells(rows, cols)	Zoom to specific cells in the plot
zoom_to_selection()	Zoom to the selected cells in the plot

Attributes

full_df	The pandas.DataFrame representing the full digitized data

contextMenuEvent(*event*)
Reimplement Qt method

delete_selected_rows()

fit2data()
Fit the selected cells to the *full_df*

full_df = None
The pandas.DataFrame representing the full digitized data from the *straditize.binary.DataReader.full_df* data frame

init_model(*marks*, **args*, ***kwargs*)
Initialize the table *MultiCrossMarksModel*

insert_row_above_selection()
Insert a row above the selection

insert_row_below_selection()
Insert a row below the selection

moveCursor(*cursor_action*, *modifiers*)
Update the table position.

Updates the position along with the frozen column when the cursor (selector) changes its position

resizeEvent(*event*)
Update the frozen column dimensions.

Updates takes place when the enclosing window of this table reports a dimension change

scrollTo(*index*, *hint*)
Scroll the table.

It is necessary to ensure that the item at index is visible. The view will try to position the item according to the given hint. This method does not takes effect only if the frozen column is scrolled.

setup_menu()
Setup context menu

show_all_marks()
Show all marks

See also:

show_selected_marks_only()

show_selected_marks_only()
Show only the marks selected in the table

See also:

show_all_marks()

update_section_height(*logical_index*, *old_size*, *new_size*)
Update the vertical width of the frozen column when a change takes place on any of the rows

update_section_width(*logical_index*, *old_size*, *new_size*)
Update the horizontal width of the frozen column when a change takes place in the first column of the table

zoom_to_cells(*rows*, *cols*)
Zoom to specific cells in the plot

Parameters

- **rows** (*list of int*) – The row indices of the cells
- **cols** (*list of int*) – The column indicies of the cells

zoom_to_selection()
Zoom to the selected cells in the plot

class straditize.widgets.samples_table.**SingleCrossMarksEditor**(*straditizer*, *axes=None*, **args*, ***kwargs*)
Bases: *straditize.widgets.samples_table.MultiCrossMarksEditor*

The editor for cross marks on a single axes

Parameters

- **straditizer** (*weakref.ref*) – The reference to the straditizer
- **axes** (*matplotlib.axes.Axes*) – The matplotlib axes corresponding to the marks

Methods

create_view([axes])	Create the *SingleCrossMarksView* of the editor
save_samples()	Save the samples to the straditizer without removing them

create_view(*axes=None*)
Create the *SingleCrossMarksView* of the editor

Parameters **axes** (list of matplotlib.axes.Axes) – The matplotlib axes for the marks

save_samples()
Save the samples to the straditizer without removing them

class straditize.widgets.samples_table.**SingleCrossMarksModel**(**args*, ***kwargs*)
Bases: *straditize.widgets.samples_table.MultiCrossMarksModel*

A table model to handle cross marks within one single axis

Parameters

- **column_bounds** (np.ndarray of shape (N, 2)) – The column boundaries
- **y0** (*float*) – The upper extent of the data image

Methods

columnCount([index])	The number of rows in the table
delRow(irow)	
get_cell_mark(row, column)	Get the mark for a given cell in the table
insertRow(irow[, xa, ya])	Insert a row into the table
load_new_marks(mark)	Add a new mark into the table after they have been added by the user
plot_lines()	Connect the samples through visual `lines`
remove_mark(mark)	Remove a mark from the table after it has been removed by the user
set_marks(marks, columns)	Set the `marks` attribute from the given *columns*
update_after_move(old_pos, mark)	
update_lines()	Update the `lines` or plot them

Attributes

df	Get the samples_locs
iter_marks	Iter over all marks in the `marks` attribute
marks	A list of tuples like `(float, mark)` where `float` is the y-pixel

columnCount(*index=<PyQt5.QtCore.QModelIndex object>*)
The number of rows in the table

delRow(*irow*)

property df
Get the samples_locs

get_cell_mark(*row*, *column*)
Get the mark for a given cell in the table

Parameters

- **row** (*int*) – The row of the cell
- **column** (*int*) – The column of the cell

Returns The corresponding mark from the *marks* attribute

Return type *straditize.cross_mark.CrossMarks*

insertRow(*irow*, *xa=None*, *ya=None*)
Insert a row into the table

Parameters **irow** (*int*) – The row index. If *irow* is equal to the length of the *marks*, the rows will be appended

property iter_marks
Iter over all marks in the *marks* attribute

load_new_marks(*mark*)
Add a new mark into the table after they have been added by the user

Parameters **mark** (`straditize.cross_mark.CrossMarks`) – The added mark

marks = []
A list of tuples like `(float, mark)` where `float` is the y-pixel and `mark` is the corresponding *straditize.cross_mark.CrossMarks* instance

plot_lines()
Connect the samples through visual `lines`

remove_mark(*mark*)
Remove a mark from the table after it has been removed by the user

Parameters **mark** (straditize.cross_mark.CrossMarks) – The removed mark

set_marks(*marks*, *columns*)
Set the *marks* attribute from the given *columns*

Parameters

- **marks** (list of *straditize.cross_mark.CrossMarks*) – the initial marks
- **columns** (*list of str*) – the column names to use

update_after_move(*old_pos*, *mark*)

update_lines()
Update the `lines` or plot them

See also:

plot_lines()

class straditize.widgets.samples_table.**SingleCrossMarksView**(*marks*, *full_df*, **args*, ***kwargs*)
Bases: *straditize.widgets.samples_table.MultiCrossMarksView*

A table for visualizing marks from a single axes

Parameters

- **marks** (list of *straditize.cross_mark.CrossMarks*) – the initial marks
- **full_df** (*pandas.DataFrame*) – The data fame of the full digitized data

Methods

fit2data()	Fit the selected cells to the `full_df`
init_model(marks, *args, **kwargs)	Initialize the table *SingleCrossMarksModel*
zoom_to_cells(rows, cols)	Zoom to specific cells in the plot

fit2data()
Fit the selected cells to the `full_df`

init_model(*marks*, **args*, ***kwargs*)
Initialize the table *SingleCrossMarksModel*

Parameters **%(SingleCrossMarksModel.parameters)s** –

zoom_to_cells(*rows*, *cols*)
Zoom to specific cells in the plot

Parameters

- **rows** (*list of int*) – The row indices of the cells
- **cols** (*list of int*) – The column indicies of the cells

straditize.widgets.selection_toolbar module

Module for the selection toolbar

This module defines the selection toolbar that is added to the psyplot_gui.main.MainWindow for selecting features in the stratigraphic diagram and the data reader image.

Disclaimer

Classes

PointOrRectangleSelector(ax, onselect[, ...])	RectangleSelector that allows to select points
SelectionToolbar(straditizer_widgets, *args, ...)	A toolbar for selecting features in the straditizer and data image

class straditize.widgets.selection_toolbar.**PointOrRectangleSelector**(*ax*, *onselect*, *drawtype='box'*, *minspanx=None*, *minspany=None*, *useblit=False*, *lineprops=None*, *rectprops=None*, *spancoords='data'*, *button=None*, *maxdist=10*, *marker_props=None*, *interactive=False*, *state_modifier_keys=None*)

Bases: matplotlib.widgets.RectangleSelector

RectangleSelector that allows to select points

This class reimplements the matplotlib.widgets.RectangleSelector to select points

Create a selector in *ax*. When a selection is made, clear the span and call onselect with:

```
onselect(pos_1, pos_2)
```

Methods

press(*args, **kwargs)	Button press handler and validator

and clear the drawn box/line. The `pos_1` and `pos_2` are arrays of length 2 containing the x- and y-coordinate.

If *minspanx* is not *None* then events smaller than *minspanx* in x direction are ignored (it's the same for y).

The rectangle is drawn with *rectprops*; default:

```
rectprops = dict(facecolor='red', edgecolor = 'black',
                 alpha=0.2, fill=True)
```

The line is drawn with *lineprops*; default:

```
lineprops = dict(color='black', linestyle='-',
                 linewidth = 2, alpha=0.5)
```

Use *drawtype* if you want the mouse to draw a line, a box or nothing between click and actual position by setting

`drawtype = 'line'`, `drawtype='box'` or `drawtype = 'none'`. Drawing a line would result in a line from vertex A to vertex C in a rectangle ABCD.

spancoords is one of 'data' or 'pixels'. If 'data', *minspanx* and *minspanx* will be interpreted in the same coordinates as the x and y axis. If 'pixels', they are in pixels.

button is a list of integers indicating which mouse buttons should be used for rectangle selection. You can also specify a single integer if only a single button is desired. Default is *None*, which does not limit which button can be used.

Note, typically: 1 = left mouse button 2 = center mouse button (scroll wheel) 3 = right mouse button

interactive will draw a set of handles and allow you interact with the widget after it is drawn.

state_modifier_keys are keyboard modifiers that affect the behavior of the widget.

The defaults are: dict(move=' ', clear='escape', square='shift', center='ctrl')

Keyboard modifiers, which: 'move': Move the existing shape. 'clear': Clear the current shape. 'square': Makes the shape square. 'center': Make the initial point the center of the shape. 'square' and 'center' can be combined.

press(**args*, ***kwargs*)
Button press handler and validator

class straditize.widgets.selection_toolbar.**SelectionToolbar**(*straditizer_widgets*, **args*, ***kwargs*)

Bases: `PyQt5.QtWidgets.QToolBar`, *straditize.widgets.StraditizerControlBase*

A toolbar for selecting features in the straditizer and data image

The current data object is set in the *combo* and can be accessed through the *data_obj* attribute. It's either the straditizer or the data_reader that is accessed **Methods**

add_or_remove_pattern()	Enable the removing or adding of the pattern selection
clear_selection()	Clear the current selection
create_actions()	Define the actions for the toolbar and set everything up
disable_actions()	
disconnect(self)	

Continued on next page

Table 129 – continued from previous page

enable_or_disable_widgets(b)	Enable or disable the widgets in this control
end_selection()	Finish the selection and disconnect everything
expand_selection()	Expand the selected areas to select the full labels
get_xy_slice(lastx, lasty, x, y)	Transform x- and y-coordinates to `slice` objects
invert_selection()	Invert the current selection
on_poly_select(points)	Call the `poly_callbacks` after a polygon selection
on_rect_select(e0, e1)	Call the `rect_callbacks` after a rectangle selection
refresh()	Refresh from the straditizer
select_all()	Select all features in the image
select_everything_to_the_right()	Selects everything to the right of the current selection
select_poly(points)	Select the data defined by a polygon
select_rect(slx, sly)	Select the data defined by a rectangle
set_binary_pattern_mode()	Set the current pattern mode to the binary pattern
set_col_wand_mode()	Set the current wand tool to the color wand
set_color_wand_mode()	Set the current wand tool to the color wand
set_grey_pattern_mode()	Set the current pattern mode to the binary pattern
set_label_wand_mode()	Set the current wand tool to the color wand
set_poly_select_mode()	Set the current wand tool to the color wand
set_rect_select_mode()	Set the current wand tool to the color wand
set_row_wand_mode()	Set the current wand tool to the color wand
should_be_enabled(w)	Check if a widget should be enabled
start_pattern_selection()	Open the pattern selection dialog
start_selection([arr, rgba, rect_callbacks, ...])	Start the selection in the current `data_obj`
toggle_selection()	Activate selection mode
uncheck_pattern_selection()	Disable the pattern selection
update_alpha(i)	Set the transparency of the selection image

Attributes

add_select_action	The action to add to the current selection with the selection tools
ax	The `matplotlib.axes.Axes` of the `data_obj`
canvas	The canvas of the `data_obj`
clear_select_action	An action to clear the current selection
combo	The QCombobox that defines the data object to be used
data	The np.ndarray of the `data_obj` image
data_obj	The data object as set in the `combo`.
expand_select_action	An action to expand the current selection to the full feature
fig	The `Figure` of the `data_obj`
invert_select_action	An action to invert the current selection
labels	The labeled data that is displayed
new_select_action	The action to make new selection with one of the selection tools
poly_callbacks	The functions to call after the polygon selection

Continued on next page

Table 130 – continued from previous page

rect_callbacks	The functions to call after the rectangle selection.
remove_select_action	An action to remove from the current selection with the selection tools
select_action	The rectangle selection tool
select_all_action	An action to select all features in the `data`
select_pattern_action	An action to start a pattern selection
select_right_action	An action to select everything in the data column to the right
selected(*args, **kwargs)	A signal that is emitted when something is selected
selector	A *PointOrRectangleSelector* to select features in the image
toolbar	The toolbar of the `canvas`
wand_action	The wand selection tool
widgets2disable	Built-in mutable sequence.

add_or_remove_pattern()
Enable the removing or adding of the pattern selection

property add_select_action
The action to add to the current selection with the selection tools

property ax
The `matplotlib.axes.Axes` of the *data_obj*

property canvas
The canvas of the *data_obj*

property clear_select_action
An action to clear the current selection

clear_selection()
Clear the current selection

combo = None
The QCombobox that defines the data object to be used

create_actions()
Define the actions for the toolbar and set everything up

property data
The np.ndarray of the *data_obj* image

property data_obj
The data object as set in the *combo*.

Either a *Straditizer* or a *straditize.binary.DataReader* instance.

disable_actions()

disconnect(*self*)

enable_or_disable_widgets(*b*)
Enable or disable the widgets in this control

This method enables or disables the *widgets2disable* if the *should_be_enabled()* method evaluates to True

Parameters **b** (*bool*) – If True, enable the widgets, if False, disable them

end_selection()
Finish the selection and disconnect everything

property expand_select_action
An action to expand the current selection to the full feature

expand_selection()
Expand the selected areas to select the full labels

property fig
The `Figure` of the *data_obj*

get_xy_slice(*lastx*, *lasty*, *x*, *y*)
Transform x- and y-coordinates to `slice` objects

Parameters

- **lastx** (*int*) – The initial x-coordinate
- **lasty** (*int*) – The initial y-coordinate
- **x** (*int*) – The final x-coordinate
- **y** (*int*) – The final y-coordinate

Returns

- *slice* – The `slice(lastx, x)`
- *slice* – The `slice(lasty, y)`

property invert_select_action
An action to invert the current selection

invert_selection()
Invert the current selection

property labels
The labeled data that is displayed

property new_select_action
The action to make new selection with one of the selection tools

on_poly_select(*points*)
Call the *poly_callbacks* after a polygon selection

Parameters

- **e0** (*matplotlib.backend_bases.Event*) – The initial event
- **e1** (*matplotlib.backend_bases.Event*) – The final event

on_rect_select(*e0*, *e1*)
Call the *rect_callbacks* after a rectangle selection

Parameters

- **e0** (*matplotlib.backend_bases.Event*) – The initial event
- **e1** (*matplotlib.backend_bases.Event*) – The final event

property poly_callbacks
The functions to call after the polygon selection

If not set manually, it is the *select_poly()* method. Note that this is cleared at every call of the *end_selection()*.

Callables in this list must accept one argument, a `np.ndarray` of shape `(N, 2)`. This array defines the `N` x- and y-coordinates of the points of the polygon

property rect_callbacks
The functions to call after the rectangle selection.

If not set manually, it is the *select_rect()* method. Note that this is cleared at every call of the *end_selection()*.

Callables in this list must accept two arguments `(slx, sly)`: the first one is the x-slice, and the second one the y-slice. They both correspond to the *data* attribute.

refresh()
Refresh from the straditizer

property remove_select_action
An action to remove from the current selection with the selection tools

reset_cursor_id = None

property select_action
The rectangle selection tool

select_all()
Select all features in the image

See also:

straditize.label_selection.LabelSelection.select_all_labels()

property select_all_action
An action to select all features in the *data*

select_everything_to_the_right()
Selects everything to the right of the current selection

property select_pattern_action
An action to start a pattern selection

select_poly(*points*)
Select the data defined by a polygon

Parameters **points** (*np.ndarray of shape (N, 2)*) – The x- and y-coordinates of the vertices of the polygon

See also:

poly_callbacks()

select_rect(*slx*, *sly*)
Select the data defined by a rectangle

Parameters

- **slx** (*slice*) – The x-slice of the rectangle
- **sly** (*slice*) – The y-slice of the rectangle

See also:

rect_callbacks()

property select_right_action
An action to select everything in the data column to the right

selected(**args*, ***kwargs*)
A signal that is emitted when something is selected

selector = None
A *PointOrRectangleSelector* to select features in the image

set_binary_pattern_mode()
Set the current pattern mode to the binary pattern

set_col_wand_mode()
Set the current wand tool to the color wand

set_color_wand_mode()
Set the current wand tool to the color wand

set_cursor_id = None

set_grey_pattern_mode()
Set the current pattern mode to the binary pattern

set_label_wand_mode()
Set the current wand tool to the color wand

set_poly_select_mode()
Set the current wand tool to the color wand

set_rect_select_mode()
Set the current wand tool to the color wand

set_row_wand_mode()
Set the current wand tool to the color wand

should_be_enabled(*w*)
Check if a widget should be enabled

This function checks if a given widget *w* from the *widgets2disable* attribute should be enabled or not

Parameters **w** (*QWidget*) – The widget to check

Returns True, if the widget should be enabled

Return type bool

start_pattern_selection()
Open the pattern selection dialog

This method will enable the pattern selection by starting a *straditize.widgets.pattern_selection.PatternSelectionWidget*

start_selection(*arr=None*, *rgba=None*, *rect_callbacks=None*, *poly_callbacks=None*, *apply_funcs=()*, *cancel_funcs=()*, *remove_on_apply=True*)
Start the selection in the current *data_obj*

Parameters

- **arr** (*np.ndarray*) – The labeled selection array that is used. If specified, the `enable_label_selection()` method is called of the *data_obj* with the given *arr*. If this parameter is `None`, then we expect that this method has already been called
- **rgba** (*np.ndarray*) – The RGBA image that shall be used for the color selection (see the *set_color_wand_mode()*)
- **rect_callbacks** (*list*) – A list of callbacks that shall be called after a rectangle selection has been made by the user (see *rect_callbacks*)

- **poly_callbacks** (*list*) – A list of callbacks that shall be called after a polygon selection has been made by the user (see *poly_callbacks*)
- **apply_funcs** (*list*) – A list of callables that shall be connected to the *apply_button*
- **cancel_funcs** (*list*) – A list of callables that shall be connected to the *cancel_button*
- **remove_on_apply** (*bool*) – If True and the *apply_button* is clicked, the selected labels will be removed.

toggle_selection()
Activate selection mode

property toolbar
The toolbar of the *canvas*

uncheck_pattern_selection()
Disable the pattern selection

update_alpha(*i*)
Set the transparency of the selection image

Parameters **i** (*int*) – The transparency between 0 and 100

property wand_action
The wand selection tool

property widgets2disable
Built-in mutable sequence.

If no argument is given, the constructor creates a new empty list. The argument must be an iterable if specified.

straditize.widgets.stacked_area_reader module

DataReader for stacked area plots

Disclaimer

Classes

StackedReader(image[, ax, extent, plot, ...])	A DataReader for stacked area plots

class straditize.widgets.stacked_area_reader.**StackedReader**(*image*, *ax=None*, *extent=None*, *plot=True*, *children=[]*, *parent=None*, *magni=None*, *plot_background=False*, *binary=None*)

Bases: *straditize.binary.DataReader*, *straditize.widgets.StraditizerControlBase*

A DataReader for stacked area plots

This reader only works within the straditizer GUI because the digitization (see *digitize()*) is interactive. The user has to manually distinguish the stacked variables.

Parameters

- **image** (*PIL.Image.Image*) – The image of the diagram
- **ax** (*matplotlib.axes.Axes*) – The matplotlib axes to plot on
- **extent** (*list*) – List of four number specifying the extent of the image in it's source. This extent will be used for the call of `matplotlib.pyplot.imshow()`
- **children** (list of `DataReader`) – Child readers for other columns in case the newly created instance is the parent reader
- **parent** (`DataReader`) – The parent reader.
- **magni** (`straditize.magnifier.Magnifier`) – The magnifier for the given *ax*
- **plot_background** (*bool*) – If True (and *plot* is True), a white, opaque are is plotted below the `plot_im`
- **binary** (*None*) – The binary version of the given *image*. If not provided, the `to_binary_pil()` method is used with the given *image*

Methods

add_col()	Create a column out of the current selection
decrease_current_col()	Take the previous column as the current column
digitize()	Digitize the data interactively
enable_or_disable_navigation_buttons()	Enable or disable `btn_prev` and `btn_next`
get_binary_for_col(col)	Get the binary array for a specific column
increase_current_col()	Take the next column as the current column
plot_full_df([ax])	Plot the lines for the digitized diagram
plot_potential_samples([excluded, ax, plot_kws])	Plot the ranges for potential samples
reset_lbl_col()	Reset the `lbl_col` to display the current column
resize_axes(grouper, bounds)	Reimplemented to do nothing
select_and_add_current_column()	Select the features for a column and create it as a new one
select_current_column()	Select the features of the current column
update_col()	Update the current column based on the selection.
update_plotted_full_df()	Update the plotted full_df if it is shown

Attributes

btn_add	A QPushButton to add a new variable to the current ones
btn_edit	A QPushButton to select the features in the image for the current
btn_next	A QPushButton to select the next variable during the digitization
btn_prev	A QPushButton to select the previous variable during the digitization
digitize_child	The QTreeWidgetItem that holds the digitization widgets
lbl_col	A QLabel to display the current column
start_of_current_col	The first x-pixel of the current column
strat_plot_identifier	str(object='') -> str

add_col()
Create a column out of the current selection

btn_add = None
A QPushButton to add a new variable to the current ones (see *select_and_add_current_column()*)

btn_edit = None
A QPushButton to select the features in the image for the current variable (see *select_current_column()*)

btn_next = None
A QPushButton to select the next variable during the digitization (see *increase_current_col()*)

btn_prev = None
A QPushButton to select the previous variable during the digitization (see *decrease_current_col()*)

decrease_current_col()
Take the previous column as the current column

digitize()
Digitize the data interactively

This method creates a new child item for the digitize button in the straditizer control to manually distinguish the variables in the stacked diagram.

digitize_child = None
The QTreeWidgetItem that holds the digitization widgets

enable_or_disable_navigation_buttons()
Enable or disable *btn_prev* and *btn_next*

Depending on the current column, we disable the navigation buttons *btn_prev* and *btn_next*

get_binary_for_col(*col*)
Get the binary array for a specific column

increase_current_col()
Take the next column as the current column

lbl_col = None
A QLabel to display the current column

plot_full_df(*ax=None*)
Plot the lines for the digitized diagram

plot_potential_samples(*excluded=False*, *ax=None*, *plot_kws={}*, **args*, ***kwargs*)
Plot the ranges for potential samples

reset_lbl_col()
Reset the *lbl_col* to display the current column

resize_axes(*grouper*, *bounds*)
Reimplemented to do nothing

select_and_add_current_column()
Select the features for a column and create it as a new one

select_current_column()
Select the features of the current column

property start_of_current_col
The first x-pixel of the current column

strat_plot_identifier = 'stacked'

update_col()
Update the current column based on the selection.

This method updates the end of the current column and adds or removes the changes from the columns to the right.

update_plotted_full_df()
Update the plotted full_df if it is shown

See also:

plot_full_df()

1.7.2 Submodules

straditize.__main__ module

main module of straditize

Disclaimer

Functions

get_parser([create])	Create an argument parser for the command line handling
main([exec_])	
start_app([fname, output, xlim, ylim, full, ...])	Start the psyplot GUI with the straditizer setup

straditize.__main__.**get_parser**(*create=True*)
Create an argument parser for the command line handling

This function creates a funcargparse.FuncArgParser for the usage from the command line

Parameters **create** (*bool*) – If True, the funcargparse.FuncArgParser.create_arguments() method is called

straditize.__main__.**main**(*exec_=True*)

straditize.__main__.**start_app**(*fname=None*, *output=None*, *xlim=None*, *ylim=None*, *full=False*, *reader_type='area'*, ***kwargs*)
Start the psyplot GUI with the straditizer setup

Parameters

- **fname** (*str*) – Either the path to a picture to digitize or a previously saved straditizer project (ending with '.pkl')
- **output** (*str*) – The path to the csv file where to save the digitized diagram
- **xlim** (*list of int of length 2*) – The x-limits of the data part of the diagram
- **ylim** (*list of int of length 2*) – The y-limits of the data part of the diagram
- **full** (*bool*) – If True, the image is digitzed and x- and ylim are set to the entire share of the array
- **reader_type** ({ *'area'* | *'bars'* | *'rounded bars'* | *'stacked area'* | *'line'* }) – Specify the reader type
- **name** (*list of str*) – The variable names to plot if the *output* parameter is set
- **dims** (*dict*) – A mapping from coordinate names to integers if the *project* is not given
- **plot_method** (*str*) – The name of the plot_method to use
- **project** (*str*) – If set, the project located at the given file name is loaded
- **engine** (*str*) – The engine to use for opening the dataset (see psyplot.data.open_dataset())
- **formatoptions** (*dict*) – A dictionary of formatoption that is applied to the data visualized by the chosen *plot_method*
- **tight** (*bool*) – If True/set, it is tried to figure out the tight bbox of the figure and adjust the paper size of the *output* to it
- **rc_file** (*str*) – The path to a yaml configuration file that can be used to update the rcParams
- **encoding** (*str*) – The encoding to use for loading the project. If None, it is automatically determined by pickle. Note: Set this to 'latin1' if using a project created with python2 on python3.
- **enable_post** (*bool*) – Enable the post processing formatoption. If True/set, post processing scripts are enabled in the given *project*. Only set this if you are sure that you can trust the given project file because it may be a security vulnerability.
- **seaborn_style** (*str*) – The name of the style of the seaborn package that can be used for the seaborn.set_style() function
- **output_project** (*str*) – The name of a project file to save the project to
- **concat_dim** (*str*) – The concatenation dimension if multiple files in *fnames* are provided

- **chname** (*dict*) – A mapping from variable names in the project to variable names in the datasets that should be used instead
- **backend** (*None or str*) – The backend to use. By default, the `'gui.backend'` key in the `rcParams` dictionary is used. Otherwise it can be None to use the standard matplotlib backend or a string identifying the backend
- **new_instance** (*bool*) – If True/set and the *output* parameter is not set, a new application is created
- **rc_gui_file** (*str*) – The path to a yaml configuration file that can be used to update the `rcParams`
- **include_plugins** (*list of str*) – The plugin widget to include. Can be either None to load all that are not explicitly excluded by *exclude_plugins* or a list of plugins to include. List items can be either module names, plugin names or the module name and widget via `'<module_name>:<widget>'`
- **exclude_plugins** (*list of str*) – The plugin widgets to exclude. Can be either `'all'` to exclude all plugins or a list like in *include_plugins*.
- **offline** (*bool*) – If True/set, psyplot will be started in offline mode without intersphinx and remote access for the help explorer
- **pwd** (*str*) – The path to the working directory to use. Note if you do not provide any *fnames* or *project*, but set the *pwd*, it will switch the *pwd* of the current GUI.
- **script** (*str*) – The path to a python script that shall be run in the GUI. If the GUI is already running, the commands will be executed in this GUI.
- **command** (*str*) – Python commands that shall be run in the GUI. If the GUI is already running, the commands will be executed in this GUI
- **use_all** (*bool*) – If True, use all variables. Note that this is the default if the *output* is specified and not *name*
- **exec_** (*bool*) – If True, the main loop is entered.
- **callback** (*str*) – A unique identifier for the method that should be used if psyplot is already running. Set this parameter to None to avoid sending
- **opengl_implementation** (*{'software', 'desktop', 'gles', 'automatic'}*) – OpenGL implementation to pass to Qt. Possible options are 'software', 'desktop', 'gles' and 'automatic' (which let's PyQt decide).

straditize.binary module

A module to read in and digitize the pollen diagram

Disclaimer

You should have received a copy of the GNU General Public License along with this program. If not, see <https://www.gnu.org/licenses/>.

Classes

BarDataReader(*args, **kwargs)	A DataReader for digitizing bar pollen diagrams
DataReader(image[, ax, extent, plot, ...])	A class to read in and digitize the data files of the pollen diagram
LineDataReader(image[, ax, extent, plot, ...])	A data reader for digitizing line diagrams
RoundedBarDataReader(*args, **kwargs)	A bar data reader that can be used for rounded bars

Functions

groupby_arr(arr)	Groupby a boolean array
only_parent(func)	Call the given *func* only from the parent reader

class straditize.binary.**BarDataReader**(**args*, ***kwargs*)

Bases: *straditize.binary.DataReader*

A DataReader for digitizing bar pollen diagrams

Compared to the base *DataReader* class, this reader implements a different strategy in digitizing and finding the samples. When digitizing the full diagram, we try to find the distinct bars using the *get_bars()* method. These bars might have to be splitted manually if they are not easy to distinguish. One key element to distinguish to adjacent bars is the specified *tolerance*.

The base class works for rectangular bars. If you require rounded bars, use the *RoundedBarDataReader*

Parameters **tolerance** (*int*) – If x0 is the value in a pixel row y and x1 the value in the next pixel row y+1, then the two pixel rows are considered as belonging to different bars if `abs(x1 - x0) > tolerance` (see the *get_bars()* method and the *tolerance* attribute)

Methods

create_grouper(ds, columns, *args, **kwargs)	Create the grouper that plots the results
digitize([do_split, inplace])	Reimplemented to ignore the rows between the bars
find_potential_samples(col[, min_len, ...])	Find the bars in the column
from_dataset(ds, *args, **kwargs)	Create a new *DataReader* from a *xarray.Dataset*
get_bars(arr[, do_split])	Find the distinct bars in an array
shift_vertical(pixels)	Shift the columns vertically.
to_dataset([ds])	All the necessary data as a *xarray.Dataset*

Attributes

min_fract	The minimum fraction of overlap for two bars to be considered as the
nc_meta	dict() -> new empty dictionary
samples_at_boundaries	There should not be samples at the boundaries because the first
tolerance	Tolerance to distinguish bars.

create_grouper(*ds*, *columns*, **args*, ***kwargs*)

Create the grouper that plots the results

Parameters

- **ds** (*xarray.Dataset*) – The dataset with the data
- **columns** (*list of int*) – The numbers of the columns for which the grouper should be created
- **fig** (*matplotlib.figure.Figure*) – The matplotlib figure to plot on
- **x0** (*float*) – The left boundary of the larger Bbox of the stratigraphic diagram
- **y0** (*int*) – The upper boundary of the larger Bbox of the stratigraphic diagram
- **width** (*float*) – The width of the final axes between 0 and 1
- **height** (*float*) – The height of the final axis between 0 and 1
- **ax0** (*matplotlib.axes.Axes*) – The larger matplotlib axes whose bounding box shall be used.
- **transformed** (*bool*) – If True, y-axes and x-axes have been translated (see the `px2data_x()` and *px2data_y()* methods)
- **colnames** (*list of str*) – The column names to use in the plot
- ****kwargs** – any other keyword argument that is passed to the `psy_strat.stratplot.StratGroup.from_dataset()` method

Returns The grouper that visualizes the given *columns* in the *fig*

Return type psy_strat.stratplot.StratGroup

digitize(*do_split=False*, *inplace=True*)

Reimplemented to ignore the rows between the bars

Parameters

- **do_split** (*bool*) – If True and a bar is 1.7 times longer than the mean, it is splitted into two.
- **inplace** (*bool*) – If True (default), the `full_df` attribute is updated. Otherwise a DataFrame is returned

find_potential_samples(*col*, *min_len=None*, *max_len=None*, *filter_func=None*)

Find the bars in the column

This method gets the bars in the given *col* and returns the distinct indices

Parameters

- **col** (*int*) – The column for which to find the extrema
- **min_len** (*int*) – The minimum length of one extremum. If the width of the interval where we found an extrumum is smaller than that, the extremum is ignored. If None, this parameter does not have an effect (i.e. `min_len=1`).
- **max_len** (*int*) – The maximum length of one extremum. If the width of the interval where we found an extrumum is greater than that, the extremum is ignored. If None, this parameter does not have an effect.
- **filter_func** (*function*) – A function to filter the extreme. It must accept one argument which is a list of integers representing the indices of the extremum in *a*

Returns

- *list of list of int of shape (N, 2)* – The list of N extremum locations. Each tuple in this list represents an interval *a* where one extremum might be located
- *list of list of int* – The excluded extremum locations that are ignored because we could not find a change of sign in the slope.

See also:

`find_samples()`

classmethod from_dataset(*ds*, **args*, ***kwargs*)
Create a new *DataReader* from a `xarray.Dataset`

Parameters

- **ds** (*xarray.Dataset*) – The dataset that has been stored with the *to_dataset()* method
- ***args,**kwargs** – Any other arguments passed to the *DataReader* constructor

Returns The reader recreated from *ds*

Return type *DataReader*

get_bars(*arr*, *do_split=False*)
Find the distinct bars in an array

Parameters

- **arr** (*np.ndarray*) – The array to find the bars in
- **do_split** (*bool*) – If True and a bar is 1.7 times longer than the mean, it is splitted into two.

Returns

- *list of list of ints* – The list of the distinct positions of the bars
- *list of floats* – The heights for each of the bars
- *list of list of ints* – The indices of bars that are longer than 1.7 times the mean of the other bars and should be splitted. If *do_split* is True, they have been splitted already

max_len = None

min_fract = 0.9
The minimum fraction of overlap for two bars to be considered as the same sample (see `unique_bars()`)

min_len = None

nc_meta = {'bars{reader}_bars': {'dims': ('bars{reader}_bar', 'limit'), 'long_name':

samples_at_boundaries = False
There should not be samples at the boundaries because the first sample is in the middle of the first bar

shift_vertical(*pixels*)
Shift the columns vertically.

Parameters **pixels** (*list of floats*) – The y-value for each column for which to shift the values. Note that theses values have to be greater than or equal to 0

to_dataset(*ds=None*)
All the necessary data as a `xarray.Dataset`

Parameters **ds** (*xarray.Dataset*) – The dataset in which to insert the data. If None, a new one will be created

Returns Either the given *ds* or a new `xarray.Dataset` instance

Return type xarray.Dataset

tolerance = 2
Tolerance to distinguish bars. If x0 is the value in a pixel row y and x1 the value in the next pixel row y+1, then the two pixel rows are considered as belonging to different bars if `abs(x1 - x0) > tolerance`

class straditize.binary.**DataReader**(*image*, *ax=None*, *extent=None*, *plot=True*, *children=[]*, *parent=None*, *magni=None*, *plot_background=False*, *binary=None*)

Bases: *straditize.label_selection.LabelSelection*

A class to read in and digitize the data files of the pollen diagram

The source image is stored in the *image* attribute, the binary array of it is stored in the *binary* attribute. A labeled version created by the `skimage.morphology.label()` function, is stored in the *labels* attribute and can regenerated using the *reset_labels()* method.

Subclasses of this class should reimplement the *digitize()* method that digitizes the diagram, and the *find_potential_samples()* method.

There is always one parent reader stored in the *parent* attribute. This is then the reader that is accessible through the *straditize.straditizer.Straditizer.data_reader* attribute and holds the references to other readers in it's *children* attribute

Parameters

- **image** (*PIL.Image.Image*) – The image of the diagram
- **ax** (*matplotlib.axes.Axes*) – The matplotlib axes to plot on
- **extent** (*list*) – List of four number specifying the extent of the image in it's source. This extent will be used for the call of `matplotlib.pyplot.imshow()`
- **children** (list of *DataReader*) – Child readers for other columns in case the newly created instance is the parent reader
- **parent** (*DataReader*) – The parent reader.
- **magni** (`straditize.magnifier.Magnifier`) – The magnifier for the given *ax*
- **plot_background** (*bool*) – If True (and *plot* is True), a white, opaque are is plotted below the *plot_im*
- **binary** (*None*) – The binary version of the given *image*. If not provided, the *to_binary_pil()* method is used with the given *image*

Methods

add_samples(samples[, rough_locs])	Add samples to the found ones
close()	
color_labels([categorize])	The labels of the colored array
create_exaggerations_reader(factor[, cls])	Create a new exaggerations reader for this reader
create_grouper(ds, columns, fig, x0, y0, ...)	Create the grouper that plots the results
create_variable(ds, vname, data, **kwargs)	Insert the data into a variable in an `xr.Dataset`
digitize([use_sum, inplace])	Digitize the binary image to create the full dataframe
digitize_exaggerated([fraction, absolute, ...])	Merge the exaggerated values into the original digitized result
disable_label_selection(*args, **kwargs)	Disable the label selection

Continued on next page

Table 139 – continued from previous page

draw_figure()	Draw the matplotlib `fig` and the `magni` figure
end_column_selection()	End the column selection and rmove the artists
estimated_column_starts([threshold])	The estimated column starts as `numpy.ndarray`.
find_potential_samples(col[, min_len, ...])	Find potential samples in an array
find_samples([min_fract, pixel_tol])	Find the samples in the diagram
found_extrema_per_row()	Calculate how many columns have a potential sample in each pixel row
from_dataset(ds, *args, **kwargs)	Create a new *DataReader* from a `xarray.Dataset`
get_bbox_for_cols(columns, x0, y0, width, height)	Get the boundary boxes for the columns of this reader in the results
get_binary_for_col(col)	Get the binary array for a specific column
get_cross_column_features([min_px])	Get features that are contained in two or more columns
get_disconnected_parts([fromlast, from0, ...])	Identify parts in the `binary` data that are not connected
get_labeled_array()	Create a connectivity-based labeled array of the `binary` data
get_occurences()	Extract the positions of the occurences from the selection
get_parts_at_column_ends([npixels])	Identify parts in the `binary` data that touch the next column
get_reader_for_col(col)	Get the reader for a specific column
get_surrounding_slopes(indices, arr)	
image_array()	The RGBA values of the colored image
is_obstacle(indices, arr)	Check whether the found extrema is only an obstacle of the picture
mark_as_exaggerations(mask)	Mask the given array as exaggerated
merge_close_samples(locs[, rough_locs, ...])	
merge_occurences(locs)	
merged_binaries()	Get the binary data from all children and merge them into one array
merged_labels()	Get the labeled binary data from all children merged into one array
new_child_for_cols(columns, cls[, plot])	Create a new child reader for specific columns
plot_background([ax])	Plot a white layer below the `plot_im`
plot_color_image([ax])	Plot the colored `image` on a matplotlib axes
plot_full_df([ax])	Plot the lines for the digitized diagram
plot_image([ax])	Plot the `binary` data image on a matplotlib axes
plot_other_potential_samples([tol, ...])	Plot potential samples that are not yet in the `samples`
plot_potential_samples([excluded, ax, plot_kws])	Plot the ranges for potential samples
plot_results(df[, ax, fig, transformed])	Plot the reconstructed diagram
plot_sample_hlines([ax])	Plot one horizontal line per sample in the `sample_locs`
plot_samples([ax])	Plot the diagram as lines reconstructed from the samples
px2data_x(coord)	Transform the pixel coordinates into data coordinates

Continued on next page

Table 139 – continued from previous page

recognize_hlines([fraction, min_lw, max_lw, ...])	Recognize horizontal lines in the plot and subtract them
recognize_vlines([fraction, min_lw, max_lw, ...])	Recognize horizontal lines in the plot and subtract them
recognize_xaxes([fraction, min_lw, max_lw, ...])	Recognize (and potentially remove) x-axes at bottom and top
recognize_yaxes([fraction, min_lw, max_lw, ...])	Find (and potentially remove) y-axes in the image
remove_in_children(arr, amask)	Update the child reader images after having removed binary data
remove_plots()	Remove all plotted artists by this reader
reset_column_starts()	Reset the column starts, `full_df`, `shifted`
reset_image(image[, binary])	Reset the image for this straditizer
reset_labels()	Reset the `labels` array
reset_samples()	Reset the samples
resize_axes(grouper, bounds)	Resize the axes based on column boundaries
set_as_parent()	Set this instance as the parent reader
set_hline_locs_from_selection([selection])	Save the locations of horizontal lines
set_vline_locs_from_selection([selection])	Save the locations of vertical lines
shift_vertical(pixels[, draw])	Shift the columns vertically.
show_cross_column_features([min_px, remove])	Highlight and maybe remove cross column features
show_disconnected_parts([fromlast, from0, ...])	Highlight or remove disconnected parts
show_parts_at_column_ends([npixels, remove])	Highlight or remove features that touch the column ends
show_small_parts([n, remove])	Highlight and potentially remove small features in the image
start_column_selection([use_all])	Enable the user to select columns
to_binary_pil(image[, threshold])	Convert an image to a binary
to_dataset([ds])	All the necessary data as a `xarray.Dataset`
to_grey_pil(image[, threshold])	Convert an image to a greyscale image
unique_bars([min_fract, asdict])	Estimate the unique bars
update_image(arr, amask)	Update the image after having removed binary data
update_rgba_image(arr, mask)	Update the RGBA image from the given 3D-array

Attributes

all_column_bounds	The boundaries for the data columns
all_column_ends	1D numpy array with the ends for all column (including child reader)
all_column_starts	1D numpy array with the ends for all column (including child reader)
ax	The matplotlib axes where the `plot_im` is plotted on
background	White rectangle that represents the background of the binary image.
binary	A 2D numpy array representing the binary version of the `image`
children	Child readers for specific columns.

Continued on next page

Table 140 – continued from previous page

column_bounds	The boundaries for the data columns
column_ends	1D numpy array with the ends for each column of this reader
column_starts	1D numpy array with the starts for each column of this reader
columns	The indices of the columns that are handled by this reader
exaggerated_reader	The reader that represents the exaggerations
extent	The extent of the `plot_im`
fig	The matplotlib figure of the `ax`
full_df	The full `pandas.DataFrame` of the digitized image
hline_locs	`list` or floats. The indexes of horizontal lines
image	PIL.Image.Image of the diagram part with mode RGBA
is_exaggerated	Exaggeration factor that is not 0 if this reader represents exaggeration
iter_all_readers	Iter through the `parent` reader and it's `children`
label_arrs	Built-in mutable sequence.
labels	A connectivity-based labeled version of the `binary` data
magni	the *straditize.magnifier.Magnifier* for the `ax`
magni_background	White rectangle that represents the background of the binary image in the magnifier.
magni_plot_im	magnified `plot_im`
min_fract	The minimum fraction of overlap for two bars to be considered as the
nc_meta	A mapping from variable name to meta information
non_exaggerated_reader	The reader that represents the exaggerations
num_labels	The maximum label in the `labels` array
occurences	A set of tuples marking the position of an occurence
occurences_dict	A mapping from column number to an numpy array with the indices of
occurences_value	The value that is given to the occurences in the measurements
parent	Parent reader for this instance.
plot_im	the matplotlib image artist
rough_locs	The `pandas.DataFrame` with rough locations for the samples.
sample_locs	The `pandas.DataFrame` with locations and values of the
samples_at_boundaries	a boolean flag that shall indicate if we assume that the first and last
shifted	The number of pixels the columns have been shifted
strat_plot_identifier	str(object='') -> str
vline_locs	`list` or floats. The indexes of vertical lines
xaxis_px	The x indices in column pixel coordinates that are used for x-axes

add_samples(*samples*, *rough_locs=None*)
Add samples to the found ones

Parameters

- **samples** (*series, 1d-array or DataFrame*) – The samples. If it is series, we assume that the index represents the y-value of the sample and the value the x-position (see *xcolumns*). In case of a 1d-array, we assume that the data represents the y-values of the samples. In case of a DataFrame, we assume that the columns correspond to columns in the *full_df* attribute and are True where we have a sample.

  Note that the y-values must be in image coordinates (see *extent* attribute).

- **rough_locs** (*DataFrame*) – The rough locations of the new samples (see the *rough_locs* attribute)

See also:

samples(), *rough_locs()*, *find_samples()*, *sample_locs()*

property all_column_bounds
The boundaries for the data columns

property all_column_ends
1D numpy array with the ends for all column (including child reader)

See also:

all_column_starts The starts for all column

all_column_bounds The (start, end)-tuple for all of the columns

column_ends The ends for this specific reader

reader

property all_column_starts
1D numpy array with the ends for all column (including child reader)

See also:

all_column_ends The ends for all column

all_column_bounds The (start, end)-tuple for all of the columns

column_starts The starts for this specific reader

reader

ax = None
The matplotlib axes where the *plot_im* is plotted on

background = None
White rectangle that represents the background of the binary image. This is only plotted by the parent reader

binary = None
A 2D numpy array representing the binary version of the *image*

children = []
Child readers for specific columns. Is not empty if and only if the *parent* attribute is this instance

close()

color_labels(*categorize=1*)
The labels of the colored array

property column_bounds
The boundaries for the data columns

property column_ends
1D numpy array with the ends for each column of this reader

See also:

column_starts The starts for each column

column_bounds The (start, end)-tuple for each of the columns

all_column_ends The ends for all columns, including child

reader

property column_starts
1D numpy array with the starts for each column of this reader

See also:

column_ends The ends for each column

column_bounds The (start, end)-tuple for each of the columns

all_column_starts The starts for all columns, including child

reader

property columns
The indices of the columns that are handled by this reader

create_exaggerations_reader(*factor*, *cls=None*)
Create a new exaggerations reader for this reader

Parameters

- **factor** (*float*) – The exaggeration factor
- **cls** (*type*) – The *DataReader* subclass

Returns The new exaggerated reader

Return type instance of *cls*

create_grouper(*ds*, *columns*, *fig*, *x0*, *y0*, *width*, *height*, *ax0=None*, *transformed=True*, *col-names=None*, ***kwargs*)
Create the grouper that plots the results

Parameters

- **ds** (*xarray.Dataset*) – The dataset with the data
- **columns** (*list of int*) – The numbers of the columns for which the grouper should be created
- **fig** (*matplotlib.figure.Figure*) – The matplotlib figure to plot on
- **x0** (*float*) – The left boundary of the larger Bbox of the stratigraphic diagram
- **y0** (*int*) – The upper boundary of the larger Bbox of the stratigraphic diagram
- **width** (*float*) – The width of the final axes between 0 and 1
- **height** (*float*) – The height of the final axis between 0 and 1

- **ax0** (*matplotlib.axes.Axes*) – The larger matplotlib axes whose bounding box shall be used.
- **transformed** (*bool*) – If True, y-axes and x-axes have been translated (see the *px2data_x()* and *px2data_y()* methods)
- **colnames** (*list of str*) – The column names to use in the plot
- ****kwargs** – any other keyword argument that is passed to the psy_strat.stratplot.StratGroup.from_dataset() method

Returns The grouper that visualizes the given *columns* in the *fig*

Return type psy_strat.stratplot.StratGroup

create_variable(*ds*, *vname*, *data*, ***kwargs*)
Insert the data into a variable in an `xr.Dataset`

Parameters

- **ds** (*xarray.Dataset*) – The destination dataset
- **vname** (*str*) – The name of the variable in the *nc_meta* mapping. This name might include `{reader}` which will then be replaced by the number of the reader in the *iter_all_readers* attribute
- **data** (*np.ndarray*) – The numpy array to store in the variable specified by *vname*
- ****kwargs** – A mapping from dimension to slicer that should be used to slice the dataset

Returns The resolved *vname* that has been used in the dataset

Return type str

digitize(*use_sum=False*, *inplace=True*)
Digitize the binary image to create the full dataframe

Parameters

- **use_sum** (*bool*) – If True, the sum of cells that are not background are used for each column, otherwise the value of the cell is used that has the maximal distance to the column start for each row
- **inplace** (*bool*) – If True (default), the *full_df* attribute is updated. Otherwise a DataFrame is returned

Returns The digitization result if *inplace* is `True`, otherwise None

Return type None or `pandas.DataFrame`

digitize_exaggerated(*fraction=0.05*, *absolute=8*, *inplace=True*, *return_mask=False*)
Merge the exaggerated values into the original digitized result

Parameters

- **fraction** (*float between 0 and 1*) – The fraction under which the exaggerated data should be used. Set this to 0 to ignore it.
- **absolute** (*int*) – The absolute value under which the exaggerated data should be used. Set this to 0 to ignore it.
- **inplace** (*bool*) – If True (default), the *full_df* attribute is updated. Otherwise a DataFrame is returned
- **return_mask** (*bool*) – If True, a boolean 2D array is returned indicating where the exaggerations have been used

Returns

- *pandas.DataFrame or None* – If *inplace* is False, the digitized result. Otherwise, if *return_mask* is True, the mask where the exaggerated results have been used. Otherwise None
- *pandas.DataFrame, optionally* – If *inplace* is False and *return_mask* is True, a pandas.DataFrame containing the boolean mask where the exaggerated results have been used. Otherwise, this is skipped

disable_label_selection(**args*, ***kwargs*)
Disable the label selection

This will disconnect the *pick_event* and remove the selection images

Parameters **remove** (*bool*) – Whether to remove the selection image from the plot. If None, the `_remove` attribute is used

See also:

`enable_label_selection()`, `remove_selected_labels()`

draw_figure()
Draw the matplotlib *fig* and the *magni* figure

end_column_selection()
End the column selection and rmove the artists

estimated_column_starts(*threshold=None*)
The estimated column starts as `numpy.ndarray`.

We assume a new column a pixel column i if

1. the previous pixel column $i-1$ did not contain any data ($D(i-1) = 0$)
2. THE amount of data points doubled compared to $i-1$ ($D(i) geq 2cdot D(i-1)$)
3. the amount of data points steadily increases within the next few columns to a value twice as large as the previous column ($D(i+n) geq 2cdot D(i-1)$ with $n>0$ and $D(i+j) geq D(i)$ for all $0 < j geq n$)

Each potential column starts must also be covered by a given *threshold*.

Parameters **threshold** (*float between 0 and 1*) – The fraction that has to be covered to assume a valid column start. By default, 0.1 (i.e. 10 percent)

Returns The starts for each column

Return type np.ndarray

property exaggerated_reader
The reader that represents the exaggerations

property extent
The extent of the *plot_im*

property fig
The matplotlib figure of the *ax*

find_potential_samples(*col*, *min_len=None*, *max_len=None*, *filter_func=None*)
Find potential samples in an array

This method finds extrema in an array and returns the indices where the extremum might be. The algorithm thereby filters out obstacles by first going over the array, making sure, that there is a change of sign in the slope in the found extremum, and if not, ignores it and flattens it out.

Parameters

- **col** (*int*) – The column for which to find the extrema
- **min_len** (*int*) – The minimum length of one extremum. If the width of the interval where we found an extrumum is smaller than that, the extremum is ignored. If None, this parameter does not have an effect (i.e. `min_len=1`).
- **max_len** (*int*) – The maximum length of one extremum. If the width of the interval where we found an extrumum is greater than that, the extremum is ignored. If None, this parameter does not have an effect.
- **filter_func** (*function*) – A function to filter the extreme. It must accept one argument which is a list of integers representing the indices of the extremum in *a*

Returns

- *list of list of int of shape (N, 2)* – The list of N extremum locations. Each tuple in this list represents an interval *a* where one extremum might be located
- *list of list of int* – The excluded extremum locations that are ignored because we could not find a change of sign in the slope.

See also:

find_samples()

find_samples(*min_fract=None*, *pixel_tol=5*, **args*, ***kwargs*)

Find the samples in the diagram

This function finds the samples using the *find_potential_samples()* function. It combines the found extrema from all columns and estimates the exact location using an interpolation of the slope

Parameters

- **min_fract** (*float*) – The minimum fraction between 0 and 1 that two bars have to overlap such that they are considered as representing the same sample. If None, the *min_fract* attribute is used
- **min_len** (*int*) – The minimum length of one extremum. If the width of the interval where we found an extrumum is smaller than that, the extremum is ignored. If None, this parameter does not have an effect (i.e. `min_len=1`).
- **max_len** (*int*) – The maximum length of one extremum. If the width of the interval where we found an extrumum is greater than that, the extremum is ignored. If None, this parameter does not have an effect.
- **filter_func** (*function*) – A function to filter the extreme. It must accept one argument which is a list of integers representing the indices of the extremum in *a*

Returns

- *pandas.DataFrame* – The x- and y-locations of the samples. The index is the y-location, the columns are the columns in the *full_df*.
- *pandas.DataFrame* – The rough locations of the samples. The index is the y-location of the columns, the values are lists of the potential sample locations.

found_extrema_per_row()

Calculate how many columns have a potential sample in each pixel row

Returns A series with one entry per pixel row. The values are the number of columns in the diagram that have a potential sample noted in the *rough_locs*

Return type pandas.Series

classmethod from_dataset(*ds*, **args*, ***kwargs*)
Create a new *DataReader* from a xarray.Dataset

Parameters

- **ds** (*xarray.Dataset*) – The dataset that has been stored with the *to_dataset()* method
- ***args,**kwargs** – Any other arguments passed to the *DataReader* constructor

Returns The reader recreated from *ds*

Return type *DataReader*

property full_df
The full pandas.DataFrame of the digitized image

get_bbox_for_cols(*columns*, *x0*, *y0*, *width*, *height*)
Get the boundary boxes for the columns of this reader in the results plot

This method is used by the *plot_results()* method to get the Bbox for a psy_strat.stratplot.StratGroup grouper

Parameters

- **columns** (*list of int*) – The column numbers to use
- **x0** (*float*) – The left boundary of the larger Bbox of the stratigraphic diagram
- **y0** (*int*) – The upper boundary of the larger Bbox of the stratigraphic diagram
- **width** (*float*) – The width of the final axes between 0 and 1
- **height** (*float*) – The height of the final axis between 0 and 1

Returns The boundary box for the given *columns* in the matplotlib figure

Return type matplotlib.transforms.Bbox

See also:

plot_results()

get_binary_for_col(*col*)
Get the binary array for a specific column

get_cross_column_features(*min_px=50*)
Get features that are contained in two or more columns

Parameters **min_px** (*int*) – The number of pixels that have to be contained in each column

Returns The 2D boolean mask with the same shape as the *binary* array that is True if a data pixel is considered as to belong to a cross column feature

Return type np.ndarray of dtype bool

get_disconnected_parts(*fromlast=5*, *from0=10*, *cross_column=False*)
Identify parts in the *binary* data that are not connected

Parameters

- **fromlast** (*int*) – A pixel `x1 > x0` is considered as disconnected, if it is at least `x1 - x0 >= fromlast`. If this is 0, it is ignored and only `from0` is considered.
- **from0** (*int*) – A pixel is considered as disconnected if it is more than *from0* pixels away from the column start. If this is 0, it is ignored and only `fromlast` is considered

- **cross_column** (*bool*) – If False, disconnected features are only marked in the column where the disconnection has been detected. Otherwise the entire feature is marked

Returns The 2D boolean mask with the same shape as the *binary* array that is True if a data pixel is considered as to be disconnected

Return type np.ndarray of dtype bool

get_labeled_array()
Create a connectivity-based labeled array of the *binary* data

get_occurences()
Extract the positions of the occurences from the selection

get_parts_at_column_ends(*npixels=2*)
Identify parts in the *binary* data that touch the next column

Parameters **npixels** (*int*) – If a data pixel is less than *npixels* away from the column end, it is considered to be at the column end and marked

Returns A boolean mask with the same shape as the *binary* data that is True where a pixel is considered to be at the column end

Return type np.ndarray of dtype bool

get_reader_for_col(*col*)
Get the reader for a specific column

Parameters **col** (*int*) – The column of interest

Returns Either the reader or None if no reader could be found

Return type *DataReader* or None

get_surrounding_slopes(*indices*, *arr*)

hline_locs = None
list or floats. The indexes of horizontal lines

image = None
PIL.Image.Image of the diagram part with mode RGBA

image_array()
The RGBA values of the colored image

is_exaggerated = 0
Exaggeration factor that is not 0 if this reader represents exaggeration plots

is_obstacle(*indices*, *arr*)
Check whether the found extrema is only an obstacle of the picture

property iter_all_readers
Iter through the *parent* reader and it's *children*

label_arrs = ['binary', 'labels', 'image_array']

labels = None
A connectivity-based labeled version of the *binary* data

magni = None
the *straditize.magnifier.Magnifier* for the *ax*

magni_background = None
White rectangle that represents the background of the binary image in the magnifier. This is only plotted by the parent reader

magni_color_plot_im = None

magni_plot_im = None
magnified *plot_im*

mark_as_exaggerations(*mask*)
Mask the given array as exaggerated

Parameters **mask** (*2D np.ndarray of dtype bool*) – A mask with the same shape as the *binary* array that is True if a cell should be interpreted as the visualization of an exaggeration

merge_close_samples(*locs*, *rough_locs=None*, *pixel_tol=5*)

merge_occurences(*locs*)

merged_binaries()
Get the binary data from all children and merge them into one array

Returns The binary image with the same shape as the *binary* data

Return type np.ndarray of dtype int

merged_labels()
Get the labeled binary data from all children merged into one array

Returns The labeled binary image with the same shape as the `label` data

Return type np.ndarray of dtype int

min_fract = 0.9
The minimum fraction of overlap for two bars to be considered as the same sample (see *unique_bars()*)

nc_meta = {'binary': {'dims': ('reader', 'ydata', 'xdata'), 'long_name': 'Binary ima
A mapping from variable name to meta information

new_child_for_cols(*columns*, *cls*, *plot=True*)
Create a new child reader for specific columns

Parameters

- **columns** (*list of int*) – The columns for the new reader
- **cls** (*type*) – The *DataReader* subclass
- **plot** (*bool*) – Plot the binary image

Returns The new reader for the specified *columns*

Return type instance of *cls*

property non_exaggerated_reader
The reader that represents the exaggerations

property num_labels
The maximum label in the *labels* array

property occurences
A set of tuples marking the position of an occurence

An occurence, motivated by pollen diagrams, just highlights the existence at a certain point without giving the exact value. In pollen diagrams, these are usually taxa that were found but have a percentage of less than 0.5 %.

This set of tuples (x, y) contains the coordinates of the occurences. The first value in each tuple is the y-value, the second the x-value.

See also:

occurences_dict A mapping from column number to occurences

property occurences_dict
A mapping from column number to an numpy array with the indices of an occurence

occurences_value = -9999
The value that is given to the occurences in the measurements

parent = None
Parent reader for this instance. Might be the instance itself

plot_background(*ax=None*, ***kwargs*)
Plot a white layer below the *plot_im*

Parameters

- **ax** (*matplotlib.axes.Axes*) – The matplotlib axes to plot on. If not given, the *ax* attribute is used
- ****kwargs** – Any other keyword that is given to the matplotlib.pyplot.imshow() function

plot_color_image(*ax=None*, ***kwargs*)
Plot the colored *image* on a matplotlib axes

Parameters

- **ax** (*matplotlib.axes.Axes*) – The matplotlib axes to plot on. If not given, the *ax* attribute is used
- ****kwargs** – Any other keyword that is given to the matplotlib.pyplot.imshow() function

plot_full_df(*ax=None*, **args*, ***kwargs*)
Plot the lines for the digitized diagram

Parameters

- **ax** (*matplotlib.axes.Axes*) – The matplotlib axes to plot on
- ***args,**kwargs** – Any other argument and keyword argument that is passed to the matplotlib.pyplot.plot() function

plot_im = None
the matplotlib image artist

plot_image(*ax=None*, ***kwargs*)
Plot the *binary* data image on a matplotlib axes

Parameters

- **ax** (*matplotlib.axes.Axes*) – The matplotlib axes to plot on. If not given, the *ax* attribute is used and (if this is None, too) a new figure is created
- ****kwargs** – Any other keyword that is given to the matplotlib.pyplot.imshow() function

plot_other_potential_samples(*tol=1*, *already_found=None*, **args*, ***kwargs*)
Plot potential samples that are not yet in the samples attribute

Parameters

- **tol** (*int*) – The pixel tolerance for a sample. If the distance between a potential sample and all already existing sample is greater than tolerance, the potential sample will be plotted
- **already_found** (*np.ndarray*) – The pixel rows of samples that have already been found. If not specified, the index of the *sample_locs* is used
- **excluded** (*bool*) – If True, plot the excluded samples instead of the included samples (see the return values in *find_potential_samples()*)
- **ax** (*matplotlib.axes.Axes*) – The matplotlib axes to plot on
- **plot_kws** (*dict*) – Any other keyword argument that is passed to the `matplotlib.pyplot.plot()` function. By default, this is equal to `{'marker': '+'}`
- **min_len** (*int*) – The minimum length of one extremum. If the width of the interval where we found an extrumum is smaller than that, the extremum is ignored. If None, this parameter does not have an effect (i.e. `min_len=1`).
- **max_len** (*int*) – The maximum length of one extremum. If the width of the interval where we found an extrumum is greater than that, the extremum is ignored. If None, this parameter does not have an effect.
- **filter_func** (*function*) – A function to filter the extreme. It must accept one argument which is a list of integers representing the indices of the extremum in *a*

plot_potential_samples(*excluded=False*, *ax=None*, *plot_kws={}*, **args*, ***kwargs*)

Plot the ranges for potential samples

This method plots the rough locations of potential samples (see *find_potential_samples()*

Parameters

- **excluded** (*bool*) – If True, plot the excluded samples instead of the included samples (see the return values in *find_potential_samples()*)
- **ax** (*matplotlib.axes.Axes*) – The matplotlib axes to plot on
- **plot_kws** (*dict*) – Any other keyword argument that is passed to the `matplotlib.pyplot.plot()` function. By default, this is equal to `{'marker': '+'}`
- **min_len** (*int*) – The minimum length of one extremum. If the width of the interval where we found an extrumum is smaller than that, the extremum is ignored. If None, this parameter does not have an effect (i.e. `min_len=1`).
- **max_len** (*int*) – The maximum length of one extremum. If the width of the interval where we found an extrumum is greater than that, the extremum is ignored. If None, this parameter does not have an effect.
- **filter_func** (*function*) – A function to filter the extreme. It must accept one argument which is a list of integers representing the indices of the extremum in *a*

plot_results(*df*, *ax=None*, *fig=None*, *transformed=True*)

Plot the reconstructed diagram

This method plots the reconstructed diagram using the psy-strat module.

Parameters

- **df** (*pandas.DataFrame*) – The data to plot. E.g. the *sample_locs* or the *straditize.straditizer.Straditizer.final_df* data

- **ax** (*matplotlib.axes.Axes*) – The axes to plot on. If None, a new one is created inside the given *fig*
- **fig** (*matplotlib.figure.Figure*) – The matplotlib figure to plot on. If not given, the current figure (see `matplotlib.pyplot.gcf()`) is used
- **transformed** (*bool*) – If True, y-axes and x-axes have been translated (see the *px2data_x()* and *px2data_y()* methods)

Returns

- *psyplot.project.Project* – The newly created psyplot project with the plotters
- list of `psy_strat.stratplot.StratGroup` instances – The groupers for the different columns

plot_sample_hlines(*ax=None*, ***kwargs*)
Plot one horizontal line per sample in the *sample_locs*

Parameters

- **ax** (*matplotlib.axes.Axes*) – The matplotlib axes to plot on
- ***args,**kwargs** – Any other keyword argument that is passed to the `matplotlib.pyplot.hlines()` function

plot_samples(*ax=None*, **args*, ***kwargs*)
Plot the diagram as lines reconstructed from the samples

Parameters

- **ax** (*matplotlib.axes.Axes*) – The matplotlib axes to plot on
- ***args,**kwargs** – Any other argument and keyword argument that is passed to the `matplotlib.pyplot.plot()` function

px2data_x(*coord*)
Transform the pixel coordinates into data coordinates

Parameters **coord** (*1D np.ndarray*) – The coordinate values in pixels

Returns The numpy array starting from 0 with transformed coordinates

Return type np.ndarray

Notes

Since the x-axes for stratographic plots are usually interrupted, the return values here are relative and therefore always start from 0

recognize_hlines(*fraction=0.3*, *min_lw=1*, *max_lw=None*, *remove=False*, ***kwargs*)
Recognize horizontal lines in the plot and subtract them

This method removes horizontal lines in the data diagram, i.e. rows whose non-background cells cover at least the specified *fraction* of the row.

Parameters

- **fraction** (*float*) – The fraction (between 0 and 1) that has to be covered to recognize a horizontal line
- **min_lw** (*int*) – The minimum line width for a line
- **max_lw** (*int*) – The maximum line width for a line or None if it should be ignored

- **remove** (*bool*) – If True, they will be removed immediately, otherwise they are displayed using the `enable_label_selection()` method and can be removed through the `remove_selected_labels()` method

Other Parameters **"**kwargs"** – Additional keywords are parsed to the `enable_label_selection()` method in case *remove* is `False`

Notes

This method has to be called before the *digitize()* method!

recognize_vlines (*fraction=0.3*, *min_lw=1*, *max_lw=None*, *remove=False*, ***kwargs*)
Recognize horizontal lines in the plot and subtract them

This method removes horizontal lines in the data diagram, i.e. rows whose non-background cells cover at least the specified *fraction* of the row.

Parameters

- **fraction** (*float*) – The fraction (between 0 and 1) that has to be covered to recognize a horizontal line
- **min_lw** (*int*) – The minimum line width for a line
- **max_lw** (*int*) – The maximum line width for a line or None if it should be ignored
- **remove** (*bool*) – If True, they will be removed immediately, otherwise they are displayed using the `enable_label_selection()` method and can be removed through the `remove_selected_labels()` method

Other Parameters **"**kwargs"** – Additional keywords are parsed to the `enable_label_selection()` method in case *remove* is `False`

Notes

This method should be called before the column starts are set

recognize_xaxes (*fraction=0.3*, *min_lw=1*, *max_lw=None*, *remove=False*, ***kwargs*)
Recognize (and potentially remove) x-axes at bottom and top

Parameters

- **fraction** (*float*) – The fraction (between 0 and 1) that has to be covered to recognize an x-axis
- **min_lw** (*int*) – The minimum line width of an axis
- **max_lw** (*int*) – Tha maximum line width of an axis. If not specified, it will be ignored
- **remove** (*bool*) – If True, they will be removed immediately, otherwise they are displayed using the `enable_label_selection()` method and can be removed through the `remove_selected_labels()` method

recognize_yaxes (*fraction=0.3*, *min_lw=0*, *max_lw=None*, *remove=False*)
Find (and potentially remove) y-axes in the image

Parameters

- **fraction** (*float*) – The fraction (between 0 and 1) that has to be covered to recognize a y-axis
- **min_lw** (*int*) – The minimum line width of an axis

- **max_lw** (*int*) – Tha maximum line width of an axis. If not specified, the median if the axes widths is taken
- **remove** (*bool*) – If True, they will be removed immediately, otherwise they are displayed using the `enable_label_selection()` method and can be removed through the `remove_selected_labels()` method

remove_in_children(*arr*, *amask*)
Update the child reader images after having removed binary data

Calls the *update_image()* and *update_rgba_image()* methods for all *children*

remove_plots()
Remove all plotted artists by this reader

reset_column_starts()
Reset the column starts, *full_df*, *shifted* and *occurences*

reset_image(*image*, *binary=False*)
Reset the image for this straditizer

Parameters

- **image** (*PIL.Image.Image*) – The new image
- **binary** (*bool*) – If True, then the *image* is considered as the binary image and the *image* attribute is not touched

reset_labels()
Reset the *labels* array

reset_samples()
Reset the samples

resize_axes(*grouper*, *bounds*)
Resize the axes based on column boundaries

This method sets the x-limits for the different columns to the given *bounds* and resizes the axes

Parameters

- **grouper** (*psy_strat.stratplot.StratGroup*) – The grouper that manages the plot
- **bounds** (*np.ndarray of shape (N, 2)*) – The boundaries for the columns handled by the *grouper*

property rough_locs
The `pandas.DataFrame` with rough locations for the samples. It has one row per sample in the *sample_locs* dataframe and `ncols * 2` columns, where `ncols` is the number of columns in the *sample_locs*.

If the potential sample *sample_locs*`.iloc[i, col]` ranges `j` to `k` (see the *find_potential_samples()* method), the cell at `rough_locs.iloc[i, col * 2]` specifies the first y-pixel (`j`) and `rough_locs.iloc[i, col * 2 + 1]` the last y-pixel (+1), i.e. `k` where this sample might be located

property sample_locs
The `pandas.DataFrame` with locations and values of the samples

samples_at_boundaries = True
a boolean flag that shall indicate if we assume that the first and last rows shall be a sample if they contain non-zero values

set_as_parent()
Set this instance as the parent reader

set_hline_locs_from_selection(*selection=None*)
Save the locations of horizontal lines

This methods takes every pixel row in the *hline_locs* attribute where at least 30% is selected. The digitize method will interpolate at these indices.

set_vline_locs_from_selection(*selection=None*)
Save the locations of vertical lines

This methods takes every pixel column in the *vline_locs* attribute where at least 30% is selected.

shift_vertical(*pixels*, *draw=True*)
Shift the columns vertically.

Parameters

- **pixels** (*list of floats*) – The y-value for each column for which to shift the values. Note that theses values have to be greater than or equal to 0
- **draw** (*bool*) – If True, the *ax* is drawn at the end

shifted = None
The number of pixels the columns have been shifted

show_cross_column_features(*min_px=50*, *remove=False*, ***kwargs*)
Highlight and maybe remove cross column features

Parameters

- **min_px** (*int*) – The number of pixels that have to be contained in each column
- **remove** (*bool*) – If True, remove the data in the *binary* array, etc. If False, the *enable_label_selection()* method is envoked and the user can select the features to remove
- **select_all** (*bool*) – If True and *remove* is False, all labels in *arr* will be selected and the given *selection* is ignored
- **selection** (*np.ndarray of dtype bool*) – A boolean mask with the same shape as *arr* that is True where a pixel should be selected. If *remove* is True, only this mask will be used.
- **img** (*matplotlib image*) – The image for the selection. If not provided, a new image is created
- **set_picker** (*bool*) – If True, connect the matplotlib pick_event to the pick_label() method

show_disconnected_parts(*fromlast=5*, *from0=10*, *remove=False*, ***kwargs*)
Highlight or remove disconnected parts

Parameters

- **%(DataReader.get_disconnected_parts.parameters.fromlast|from0)s** –
- **%(DataReader._show_parts2remove.parameters.no_arr)s** –

show_parts_at_column_ends(*npixels=2*, *remove=False*, ***kwargs*)
Highlight or remove features that touch the column ends

Parameters

- **%(DataReader.get_parts_at_column_ends.parameters)s** –
- **%(DataReader._show_parts2remove.parameters.no_arr)s** –

show_small_parts(*n=10*, *remove=False*, ***kwargs*)
Highlight and potentially remove small features in the image

Parameters

- **n** (*int*) – The maximal size of a feature to be considered as small
- **remove** (*bool*) – If True, remove the data in the *binary* array, etc. If False, the *enable_label_selection()* method is envoked and the user can select the features to remove
- **select_all** (*bool*) – If True and *remove* is False, all labels in *arr* will be selected and the given *selection* is ignored
- **selection** (*np.ndarray of dtype bool*) – A boolean mask with the same shape as *arr* that is True where a pixel should be selected. If *remove* is True, only this mask will be used.
- **img** (*matplotlib image*) – The image for the selection. If not provided, a new image is created
- **set_picker** (*bool*) – If True, connect the matplotlib pick_event to the `pick_label()` method

See also:

`skimage.morphology.remove_small_objects()`

start_column_selection(*use_all=False*)
Enable the user to select columns

Parameters **use_all** (*bool*) – If True, all columns can be selected. Otherwise only the columns in the *columns* attribute can be selected

strat_plot_identifier = 'percentages'

static to_binary_pil(*image*, *threshold=690*)
Convert an image to a binary

Parameters

- **image** (*PIL.Image.Image*) – The RGBA image file
- **threshold** (*float*) – If the multiplied RGB values in a cell are above the threshold, the cell is regarded as background and will be set to 0

Returns The binary image of integer type

Return type np.ndarray of ndim 2

to_dataset(*ds=None*)
All the necessary data as a `xarray.Dataset`

Parameters **ds** (*xarray.Dataset*) – The dataset in which to insert the data. If None, a new one will be created

Returns Either the given *ds* or a new `xarray.Dataset` instance

Return type xarray.Dataset

static to_grey_pil(*image*, *threshold=690*)
Convert an image to a greyscale image

Parameters

- **image** (*PIL.Image.Image*) – The RGBA image file
- **threshold** (*float*) – If the multiplied RGB values in a cell are above the threshold, the cell is regarded as background and will be set to 0

Returns The greyscale image of integer type

Return type np.ndarray of ndim 2

unique_bars(*min_fract=None*, *asdict=True*, **args*, ***kwargs*)
Estimate the unique bars

This method puts the overlapping bars of the different columns together

Parameters

- **min_fract** (*float*) – The minimum fraction between 0 and 1 that two bars have to overlap such that they are considered as representing the same sample. If None, the *min_fract* attribute is used
- **asdict** (*bool*) – If True, dictionaries are returned

Returns A list of the bar locations. If asdict is True (default), each item in the returned list is a dictionary whose keys are the column indices and whose values are the indices for the corresponding column. Otherwise, a list of `_Bar` objects is returned

Return type list

update_image(*arr*, *amask*)
Update the image after having removed binary data

This method is in the `remove_callbacks` mapping and is called after a pixel has been removed from the *binary* data. It mainly just calls the *reset_labels()* method and updates the plot

update_rgba_image(*arr*, *mask*)
Update the RGBA image from the given 3D-array

This method is in the `remove_callbacks` mapping and is called after a pixel has been removed from the *binary* data. It updates the *image* attribute

Parameters

- **arr** (*3D np.ndarray of dtype float*) – The image array
- **mask** (boolean mask of the same shape as *arr*) – The mask of features that shall be set to 0 in *arr*

vline_locs = None
`list` or floats. The indexes of vertical lines

xaxis_data = None

property xaxis_px
The x indices in column pixel coordinates that are used for x-axes translations

class straditize.binary.**LineDataReader**(*image*, *ax=None*, *extent=None*, *plot=True*, *children=[]*, *parent=None*, *magni=None*, *plot_background=False*, *binary=None*)
Bases: *straditize.binary.DataReader*

A data reader for digitizing line diagrams

This class does not have a significantly different behaviour than the base *DataReader* class, but might be improved with more specific features in the future

Parameters

- **image** (*PIL.Image.Image*) – The image of the diagram
- **ax** (*matplotlib.axes.Axes*) – The matplotlib axes to plot on
- **extent** (*list*) – List of four number specifying the extent of the image in it's source. This extent will be used for the call of `matplotlib.pyplot.imshow()`
- **children** (list of *DataReader*) – Child readers for other columns in case the newly created instance is the parent reader
- **parent** (*DataReader*) – The parent reader.
- **magni** (`straditize.magnifier.Magnifier`) – The magnifier for the given *ax*
- **plot_background** (*bool*) – If True (and *plot* is True), a white, opaque are is plotted below the `plot_im`
- **binary** (*None*) – The binary version of the given *image*. If not provided, the `to_binary_pil()` method is used with the given *image*

Attributes

strat_plot_identifier	str(object='') -> str

strat_plot_identifier = 'default'

class straditize.binary.**RoundedBarDataReader**(**args*, ***kwargs*)

Bases: *straditize.binary.BarDataReader*

A bar data reader that can be used for rounded bars

Parameters **tolerance** (*int*) – If x0 is the value in a pixel row y and x1 the value in the next pixel row y+1, then the two pixel rows are considered as belonging to different bars if `abs(x1 - x0) > tolerance` (see the `get_bars()` method and the *tolerance* attribute)

Attributes

tolerance	int([x]) -> integer

tolerance = 10

straditize.binary.**groupby_arr**(*arr*)

Groupby a boolean array

Parameters **arr** (*np.ndarray of ndim 1 of dtype bool*) – An array that can be converted to a numeric array

Returns

- **keys** (*np.ndarrayrdi*) – The keys in the array
- **starts** (*np.ndarray*) – The index of the first element that correspond to the key in *keys*

straditize.binary.**only_parent**(*func*)

Call the given *func* only from the parent reader

straditize.colnames module

Module for text recognition

Disclaimer

Classes

Bbox	A bounding box for a column name
ColNamesReader(image, bounds[, rotate, ...])	A class to recognize the text in an image

class straditize.colnames.**Bbox**

Bases: straditize.colnames._Bbox

A bounding box for a column name

Create new instance of _Bbox(x, y, w, h) **Attributes**

bottom	The bottom of the box
bounds	A list `[x, y, width, height]`
corners	A np.ndarray of shape (4, 2) with the corners of the box
crop_extents	The extents necessary for PIL.Image.crop
extents	A list `[x0, x1, y0, y1]` with `x0 <= x1` and `y0 <= y1`
height	The (positive) height
left	The left edge of the box
right	The right edge of the box
top	The top of the box
width	The (positive) width
x0	The left edge
x1	The right edge
y0	The lower (bottom) edge
y1	The upper (top) edge

Methods

from_dict(d)	Construct a box from the dictionary

property bottom
The bottom of the box

property bounds
A list `[x, y, width, height]`

property corners
A np.ndarray of shape (4, 2) with the corners of the box

property crop_extents
The extents necessary for PIL.Image.crop

property extents
A list `[x0, x1, y0, y1]` with `x0 <= x1` and `y0 <= y1`

classmethod from_dict(*d*)
Construct a box from the dictionary

property height
The (positive) height

property left
The left edge of the box

property right
The right edge of the box

property top
The top of the box

property width
The (positive) width

property x0
The left edge

property x1
The right edge

property y0
The lower (bottom) edge

property y1
The upper (top) edge

class straditize.colnames.**ColNamesReader**(*image*, *bounds*, *rotate=45*, *mirror=False*, *flip=False*, *highres_image=None*, *data_ylim=None*)

Bases: object

A class to recognize the text in an image

This object handles the column names in the *column_names* attribute. It also implements several algorithms to automatically read in the column names using the tesserocr package. In particular these are the *recognize_text()* method to read in one small image and the *find_colnames()* method to find the column names automatically.

Parameters

- **image** (*PIL.Image.Image*) – The RGBA image that has the same shape as the original stratigraphic diagram
- **bounds** (*np.ndarray of shape (N, 2)*) – The boundaries for each column. These are essential for the *find_colnames()* and the *highlight_column()* methods
- **rotate** (*float*) – An angle between 0 and 90 that corresponds to the rotation of the column names
- **mirror** (*bool*) – If True, the image is mirrored (horizontally)
- **flip** (*bool*) – If True, the image is flipped (vertically)

- **highres_image** (*PIL.Image.Image*) – A high resolution version of the *image* with the same width-to-height ratio
- **data_ylim** (*tuple (y0, y1)*) – The vertical data limits of the data part that should be ignored in the *find_colnames()* method if the *ignore_data_part* is True

Methods

close()	Close the column names reader
create_variable(ds, vname, data, **kwargs)	Insert the data into a variable in an `xr.Dataset`
find_colnames([extents])	Find the names for the columns using tesserocr
from_dataset(ds)	Create a *ColNamesReader* for a xarray.Dataset
get_colpic(x0, y0, x1, y1)	Extract the picture of the column name
highlight_column(col, ax)	Highlight the column in the given axes displaying the
navigate_to_col(col, ax)	Navigate to the specified column
recognize_text(image)	Recognize the text in an image using tesserocr
rotate_image(image)	Modify an image with `rotate`, `flip`, `mirror`
to_dataset([ds])	All the necessary data as a `xarray.Dataset`
transform_point(x, y[, invert, image])	Transform a point between un-rotated and rotated coordinate system

Attributes

colpics	The pictures of the column names
column_names	The names of the columns
data_ylim	The vertical data limits of the data part that shall be exluded in the
highres_image	The `image` attribute with higher resolution and with masked out data part if the `ignore_data_part` attribute is True and the `data_ylim` attribute is not None.
ignore_data_part	Boolean flag.
image	The RGBA `PIL.Image.Image` that stores the column names
nc_meta	dict() -> new empty dictionary
rotated_image	The rotated `image` based on the `rotate_image()` method

close()
Close the column names reader

property colpics
The pictures of the column names

property column_names
The names of the columns

create_variable(*ds*, *vname*, *data*, ***kwargs*)
Insert the data into a variable in an `xr.Dataset`

data_ylim = None
The vertical data limits of the data part that shall be exluded in the *highres_image* if the *ignore_data_part* is True

find_colnames(*extents=None*)
Find the names for the columns using tesserocr

Parameters **extents** (*list of floats (x0, y0, x1, y1)*) – The extents to crop the *rotated_image*. We only look for column names in this image

Returns

- *dict* – A mapping from column number to a string (the column name)
- *dict* – A mapping from column number to a PIL.Image.Image (the image of the column name)
- *dict* – A mapping from column number to a *Bbox* (the bounding box of the corresponding column name)

classmethod from_dataset(*ds*)
Create a *ColNamesReader* for a xarray.Dataset

Parameters **ds** (*xarray.Dataset*) – The dataset as obtained from the *to_dataset()* method

get_colpic(*x0*, *y0*, *x1*, *y1*)
Extract the picture of the column name

Parameters

- **x0** (*int*) – The left edge
- **y0** (*int*) – The upper edge
- **x1** (*int*) – The right edge
- **y1** (*int*) – The lower edge

Returns The part of the rotated *highres_image* cropped out from the given parameters

Return type PIL.Image.Image

highlight_column(*col*, *ax*)
Highlight the column in the given axes displaying the *rotated_image*

This method draws a rotated rectangle highlighting the given column *col* in the given *ax*.

Parameters

- **col** (*int*) – The column number
- **ax** (*matplotlib.axes.Axes*) – The matplotlib axes on which to plot the rectangle. This *ax* is expected to show the *rotated_image*

property highres_image
The *image* attribute with higher resolution and with masked out data part if the *ignore_data_part* attribute is True and the *data_ylim* attribute is not None. The data part is then set to white with 0 alpha

ignore_data_part = True
Boolean flag. If True, the data part is masked out in the *highres_image*

image = None
The RGBA PIL.Image.Image that stores the column names

navigate_to_col(*col*, *ax*)
Navigate to the specified column

Change the x- and y-limits of the *ax* to display the given *col* based on the column_bounds

Parameters

- **col** (*int*) – The column number

- **ax** (*matplotlib.axes.Axes*) – The matplotlib axes for which to update the limits. This *ax* is expected to show the *rotated_image*

nc_meta = {'colname': {'dims': 'column', 'long_name': 'Name of the columns'}, 'colna

recognize_text(*image*)
Recognize the text in an image using tesserocr

This method uses the `tesserocr.image_to_text()` to read in the text in a given *image*

Parameters **image** (*PIL.Image.Image*) – The image to read in

Returns The text found in it without newline characters

Return type str

rotate_image(*image*)
Modify an image with `rotate`, `flip`, `mirror`

This method rotated, mirrors and/or flips the given *image* based on the `rotate`, `mirror` and `flip` attributes

Parameters **image** (*PIL.Image.Image*) – The source image

Returns The target image

Return type PIL.Image.Image

property rotated_image
The rotated *image* based on the *rotate_image()* method

to_dataset(*ds=None*)
All the necessary data as a `xarray.Dataset`

Parameters **ds** (*xarray.Dataset*) – The dataset in which to insert the data. If None, a new one will be created

Returns Either the given *ds* or a new `xarray.Dataset` instance

Return type xarray.Dataset

transform_point(*x*, *y*, *invert=False*, *image=None*)
Transform a point between un-rotated and rotated coordinate system

Parameters

- **x** (*float*) – The x-coordinate of the point in the source coordinate system
- **y** (*float*) – The y-coordinate of the point in the source coordinate system
- **invert** (*bool*) – If True, the source coordinate system is the rotated one (i.e. this method transform from the *rotated_image* to the coordinate system of the *image*), other wise from the *image* to the *rotated_image*
- **image** (*PIL.Image.Image*) – The unrotated source image. If None, the *image* is used. This image defines the source coordinate system (or the target coordinate system if *invert* is True)

Returns

- *float* – The transformed *x*-coordinate
- *float* – The transformed *y*-coordinate

straditize.common module

Module of commonly use python objects

Disclaimer

Functions

rgba2rgb(image[, color])	Alpha composite an RGBA Image with a specified color.

straditize.common.**rgba2rgb**(*image*, *color=(255, 255, 255)*)

Alpha composite an RGBA Image with a specified color.

Source: http://stackoverflow.com/a/9459208/284318

Parameters

- **image** (*PIL.Image*) – The PIL RGBA Image object
- **color** (*tuple*) – The rgb color for the background

Returns The rgb image

Return type PIL.Image

straditize.cross_mark module

Module for a cross mark to select one point in a matplotlib axes

Disclaimer

Classes

CrossMarkText(*args, **kwargs)	A CrossMarks that opens a QInputDialog after changing the position
CrossMarks([pos, ax, selectable, draggable, ...])	A set of draggable marks in a matplotlib axes
DraggableHLine(y[, ax])	A draggable horizontal line
DraggableHLineText(*args, **kwargs)	A CrossMarks that opens a QInputDialog after changing the position
DraggableVLine(x[, ax])	A draggable vertical line
DraggableVLineText(*args, **kwargs)	A CrossMarks that opens a QInputDialog after changing the position

class straditize.cross_mark.**CrossMarkText**(**args*, ***kwargs*)

Bases: *straditize.cross_mark.CrossMarks*

A CrossMarks that opens a QInputDialog after changing the position

Parameters

- **pos** (*tuple of 2 arrays*) – The initial positions of the crosses. The first item marks the x-coordinates of the points, the second the y-coordinates
- **ax** (*matplotlib.axes.Axes*) – The axes object to draw to. If not specified and draw_lines is True, the current axes object is used
- **selectable** (*list of {'x', 'y'}*) – Determine whether only the x-, y-, or both lines should be selectable
- **draggable** (*list of {'x', 'y'}*) – Determine whether only the x-, y-, or both lines should be draggable
- **idx_h** (*pandas.Index*) – The index for the horizontal coordinates. If not provided, we use a continuous movement along x.
- **idx_v** (*pandas.Index*) – The index for the vertical coordinates. If not provided, we use a continuous movement along y.
- **xlim** (*tuple of floats (xmin, xmax)*) – The minimum and maximum x value for the lines
- **ylim** (*tuple for floats (ymin, ymax)*) – The minimum and maximum y value for the lines
- **select_props** (*color*) – The line properties for selected marks
- **auto_hide** (*bool*) – If True, the lines are hidden if they are not selected.
- **connected_artists** (*list of artists*) – List of artists whose properties should be changed to *select_props* when this marks is selected
- **lock** (*bool*) – If True, at most one mark can be selected at a time
- **draw_lines** (*bool*) – If True, the cross mark lines are drawn. Otherwise, you must call the *draw_lines* method explicitly
- **hide_vertical** (*bool*) – Boolean to control whether the vertical lines should be hidden. If None, the default class attribute is used
- **hide_horizontal** (*bool*) – Boolean to control whether the horizontal lines should be hidden. If None, the default class attribute is used
- ****kwargs** – Any other keyword argument that is passed to the matplotlib.pyplot.plot() function

- **dtype** (*object*) – The data type for the data conversion
- **message** (*str*) – The message to display in the dialog
- **label** (*str*) – The label to how this value should be named
- **value** (*float*) – The initial value to use

Methods

ask_for_value([val, label])	Ask for a value for the cross mark
on_release(event, *args, **kwargs)	Release the mark and unselect it

Attributes

value	The value of this cross mark

ask_for_value(*val=None*, *label=None*)
Ask for a value for the cross mark

This method opens a QInputDialog to ask for a new *value*

Parameters

- **val** (*float*) – The initial value
- **label** (*str*) – the name of what to ask for

on_release(*event*, **args*, ***kwargs*)
Release the mark and unselect it

Parameters

- **event** (*matplotlib.backend_bases.MouseEvent*) – The mouseevent that releases the mark
- **force** (*bool*) – If True, the mark is released although it does not contain the *event*
- **connected** (*bool*) – If True, connected marks that should maintain a constant x- and y-distance are released, too
- **draw** (*bool*) – If True, the figure is drawn
- ****kwargs** (**args,*) – Any other parameter that is passed to the connected lines

value = None
The value of this cross mark

class straditize.cross_mark.**CrossMarks**(*pos=(0, 0), ax=None, selectable=['h', 'v'], draggable=['h', 'v'], idx_h=None, idx_v=None, xlim=None, ylim=None, select_props={'c': 'r'}, auto_hide=False, connected_artists=[], lock=True, draw_lines=True, hide_vertical=None, hide_horizontal=None, **kwargs*)

Bases: object

A set of draggable marks in a matplotlib axes

Parameters

- **pos** (*tuple of 2 arrays*) – The initial positions of the crosses. The first item marks the x-coordinates of the points, the second the y-coordinates

- **ax** (*matplotlib.axes.Axes*) – The axes object to draw to. If not specified and draw_lines is True, the current axes object is used
- **selectable** (*list of {'x', 'y'}*) – Determine whether only the x-, y-, or both lines should be selectable
- **draggable** (*list of {'x', 'y'}*) – Determine whether only the x-, y-, or both lines should be draggable
- **idx_h** (*pandas.Index*) – The index for the horizontal coordinates. If not provided, we use a continuous movement along x.
- **idx_v** (*pandas.Index*) – The index for the vertical coordinates. If not provided, we use a continuous movement along y.
- **xlim** (*tuple of floats (xmin, xmax)*) – The minimum and maximum x value for the lines
- **ylim** (*tuple for floats (ymin, ymax)*) – The minimum and maximum y value for the lines
- **select_props** (*color*) – The line properties for selected marks
- **auto_hide** (*bool*) – If True, the lines are hidden if they are not selected.
- **connected_artists** (*list of artists*) – List of artists whose properties should be changed to *select_props* when this marks is selected
- **lock** (*bool*) – If True, at most one mark can be selected at a time
- **draw_lines** (*bool*) – If True, the cross mark lines are drawn. Otherwise, you must call the *draw_lines* method explicitly
- **hide_vertical** (*bool*) – Boolean to control whether the vertical lines should be hidden. If None, the default class attribute is used
- **hide_horizontal** (*bool*) – Boolean to control whether the horizontal lines should be hidden. If None, the default class attribute is used
- ****kwargs** – Any other keyword argument that is passed to the `matplotlib.pyplot.plot()` function

Attributes

ax	The matplotlib axes to plot on
block_signals	Block the emitting of signals of this instance
connected_artists	a list of `matplotlib.artist.Artist` whose colors are changed
fig	The `matplotlib.figure.Figure` that this mark plots on
hide_horizontal	Boolean to control whether the horizontal lines should be hidden
hide_vertical	Boolean to control whether the vertical lines should be hidden
hline	The current horizontal line
hlines	the list of horizontal lines
idx_h	The index for vertical lines
idx_v	The index for horizontal lines
line_connections	The line connections to the current position

Continued on next page

Table 152 – continued from previous page

lock	Class attribute that is set to a `CrossMark` instance to lock the
moved	A signal that is emitted when the mark is moved.
other_connections	All other connections to the current position
points	The x-y-coordinates of the points as a (N, 2)-shaped array
pos	The position of the current line
show_connected_artists	A boolean to control whether the connected artists should be shown
vline	The current vertical line
vlines	the list of vertical lines
x	The x-position of the mark
xlim	The x-limits of the `hlines`
y	The y-position of the mark
ylim	The x-limits of the `vlines`

Methods

connect()	Connect the marks matplotlib events
connect_marks(marks[, visible])	Connect multiple marks to each other
connect_to_marks(marks[, visible, append])	Append other marks that should be considered for aligning the lines
contains(event)	Test if the mark is selected by the given *event*
disconnect()	Disconnect all the stored connection ids
draw_lines(**kwargs)	Draw the vertical and horizontal lines
is_selected_by(event[, buttons])	Test if the given *event* selects the mark
maintain_x(marks)	Connect marks and maintain a constant horizontal distance
maintain_y(marks)	Connect marks and maintain a constant vertical distance between them
on_motion(event[, force, move_connected, ...])	Move the lines of this mark
on_press(event[, force, connected])	Select the mark
on_release(event[, force, connected, draw])	Release the mark and unselect it
remove([artists])	Remove all lines and disconnect the mark
set_connected_artists(artists)	Set the connected artists
set_connected_artists_visible(visible)	Set the visibility of the connected artists
set_current_point(x, y[, nearest])	Set the current point that is selected
set_pos(pos)	Move the point(s) to another position
set_visible(b)	Set the visibility of the mark

ax = None
The matplotlib axes to plot on

property block_signals
Block the emitting of signals of this instance

connect()
Connect the marks matplotlib events

static connect_marks(*marks*, *visible=False*)
Connect multiple marks to each other

Parameters

- **marks** (*list of CrossMarks*) – A list of marks
- **visible** (*bool*) – If True, the marks are connected through visible lines

Notes

Different from the *connect_to_marks()* method, this static function connects each of the marks to the others.

connect_to_marks(*marks*, *visible=False*, *append=True*)
Append other marks that should be considered for aligning the lines

Parameters

- **marks** (*list of CrossMarks*) – A list of other marks
- **visible** (*bool*) – If True, the marks are connected through visible lines
- **append** (*bool*) – If True, the marks are appended. This is important if the mark will be moved by the *set_pos* method

Notes

This method can only be used to connect other marks with this mark. If you want to connect multiple marks within each other, use the *connect_marks()* static method

connected_artists = []
a list of matplotlib.artist.Artist whose colors are changed when this mark is selected

contains(*event*)
Test if the mark is selected by the given *event*

Parameters **event** (*ButtonPressEvent*) – The ButtonPressEvent that has been triggered

disconnect()
Disconnect all the stored connection ids

draw_lines(***kwargs*)
Draw the vertical and horizontal lines

Parameters ****kwargs** – An keyword that is passed to the matplotlib.pyplot.plot() function

property fig
The matplotlib.figure.Figure that this mark plots on

hide_horizontal = False
Boolean to control whether the horizontal lines should be hidden

hide_vertical = False
Boolean to control whether the vertical lines should be hidden

property hline
The current horizontal line

hlines = []
the list of horizontal lines

property idx_h
The index for vertical lines

property idx_v
The index for horizontal lines

is_selected_by(*event, buttons=[1]*)
Test if the given *event* selects the mark

Parameters

- **event** (*matplotlib.backend_bases.MouseEvent*) – The matplotlib event
- **button** (*list of int*) – Possible buttons to select this mark

Returns True, if it is selected

Return type bool

property line_connections
The line connections to the current position

lock = None
Class attribute that is set to a `CrossMark` instance to lock the selection of marks

static maintain_x(*marks*)
Connect marks and maintain a constant horizontal distance

Parameters **marks** (*list of CrossMarks*) – A list of marks. If one of the marks is moved horizontally, the others are, too

static maintain_y(*marks*)
Connect marks and maintain a constant vertical distance between them

Parameters **marks** (*list of CrossMarks*) – A list of marks. If one of the marks is moved vertically, the others are, too

moved
A signal that is emitted when the mark is moved. Connected function are expected to accept two arguments. One tuple with the old position and the CrossMarks instance itself

on_motion(*event*, *force=False*, *move_connected=True*, *restore=True*)
Move the lines of this mark

Parameters

- **event** (*matplotlib.backend_bases.MouseEvent*) – The mouseevent that moves the mark
- **force** (*bool*) – If True, the mark is moved although it does not contain the *event*
- **move_connected** (*bool*) – If True, connected marks that should maintain a constant x- and y-distance are moved, too
- **restore** (*bool*) – If True, the axes background is restored

on_press(*event*, *force=False*, *connected=True*)
Select the mark

Parameters

- **event** (*matplotlib.backend_bases.MouseEvent*) – The mouseevent that selects the mark
- **force** (*bool*) – If True, the mark is selected although it does not contain the *event*
- **connected** (*bool*) – If True, connected marks that should maintain a constant x- and y-distance are selected, too

on_release(*event*, *force=False*, *connected=True*, *draw=True*, **args*, ***kwargs*)
Release the mark and unselect it

Parameters

- **event** (*matplotlib.backend_bases.MouseEvent*) – The mouseevent that releases the mark
- **force** (*bool*) – If True, the mark is released although it does not contain the *event*
- **connected** (*bool*) – If True, connected marks that should maintain a constant x- and y-distance are released, too
- **draw** (*bool*) – If True, the figure is drawn
- ****kwargs** (**args,*) – Any other parameter that is passed to the connected lines

property other_connections
All other connections to the current position

property points
The x-y-coordinates of the points as a (N, 2)-shaped array

property pos
The position of the current line

remove(*artists=True*)
Remove all lines and disconnect the mark

Parameters **artists** (*bool*) – If True, the *connected_artists* list is cleared and the corresponding artists are removed as well

set_connected_artists(*artists*)
Set the connected artists

Parameters **artists** (*matplotlib.artist.Artist*) – The artists (e.g. other lines) that should be connected and highlighted if this mark is selected

set_connected_artists_visible(*visible*)
Set the visibility of the connected artists

Parameters **visible** (*bool*) – True, show the connected artists, else don't

set_current_point(*x*, *y*, *nearest=False*)
Set the current point that is selected

Parameters

- **x** (*int*) – The index of the x-value in the `xa` attribute
- **y** (*int*) – The index of the y-value in the `ya` attribute
- **nearest** (*bool*) – If not None, *x* and *y* are interpreted as x- and y-values and we select the closest one

set_pos(*pos*)
Move the point(s) to another position

Parameters **pos** (*tuple of 2 arrays*) – The positions of the crosses. The first item marks the x-coordinates of the points, the second the y-coordinates

set_visible(*b*)
Set the visibility of the mark

Parameters **b** (*bool*) – If False, hide all horizontal and vertical lines, and the *connected_artists*

show_connected_artists = True
A boolean to control whether the connected artists should be shown at all

property vline
The current vertical line

vlines = []
the list of vertical lines

property x
The x-position of the mark

xlim = None
The x-limits of the *hlines*

property y
The y-position of the mark

ylim = None
The x-limits of the *vlines*

class straditize.cross_mark.**DraggableHLine**(*y*, *ax=None*, **args*, ***kwargs*)
Bases: *straditize.cross_mark.CrossMarks*

A draggable horizontal line

Parameters

- **y** (*float*) – The y-position for the horizontal line
- **ax** (*matplotlib.axes.Axes*) – The matplotlib axes
- **idx_h** (*pandas.Index*) – The index for the horizontal coordinates. If not provided, we use a continuous movement along x.
- **idx_v** (*pandas.Index*) – The index for the vertical coordinates. If not provided, we use a continuous movement along y.
- **xlim** (*tuple of floats (xmin, xmax)*) – The minimum and maximum x value for the lines
- **ylim** (*tuple for floats (ymin, ymax)*) – The minimum and maximum y value for the lines
- **select_props** (*color*) – The line properties for selected marks
- **auto_hide** (*bool*) – If True, the lines are hidden if they are not selected.
- **connected_artists** (*list of artists*) – List of artists whose properties should be changed to *select_props* when this marks is selected
- **lock** (*bool*) – If True, at most one mark can be selected at a time
- **draw_lines** (*bool*) – If True, the cross mark lines are drawn. Otherwise, you must call the *draw_lines* method explicitly
- **hide_vertical** (*bool*) – Boolean to control whether the vertical lines should be hidden. If None, the default class attribute is used
- **hide_horizontal** (*bool*) – Boolean to control whether the horizontal lines should be hidden. If None, the default class attribute is used
- ****kwargs*** – Any other keyword argument that is passed to the `matplotlib.pyplot.plot()` function

Attributes

hide_vertical	bool(x) -> bool
x	The x-position of the mark

Methods

set_visible(b)	Set the visibility of the mark

hide_vertical = True

set_visible(*b*)
Set the visibility of the mark

Parameters **b** (*bool*) – If False, hide all horizontal and vertical lines, and the connected_artists

property x
The x-position of the mark

class straditize.cross_mark.**DraggableHLineText**(**args*, ***kwargs*)
Bases: *straditize.cross_mark.DraggableHLine*

A CrossMarks that opens a QInputDialog after changing the position

Parameters

- **y** (*float*) – The y-position for the horizontal line
- **ax** (*matplotlib.axes.Axes*) – The matplotlib axes
- **idx_h** (*pandas.Index*) – The index for the horizontal coordinates. If not provided, we use a continuous movement along x.
- **idx_v** (*pandas.Index*) – The index for the vertical coordinates. If not provided, we use a continuous movement along y.
- **xlim** (*tuple of floats (xmin, xmax)*) – The minimum and maximum x value for the lines
- **ylim** (*tuple for floats (ymin, ymax)*) – The minimum and maximum y value for the lines
- **select_props** (*color*) – The line properties for selected marks
- **auto_hide** (*bool*) – If True, the lines are hidden if they are not selected.
- **connected_artists** (*list of artists*) – List of artists whose properties should be changed to *select_props* when this marks is selected
- **lock** (*bool*) – If True, at most one mark can be selected at a time
- **draw_lines** (*bool*) – If True, the cross mark lines are drawn. Otherwise, you must call the *draw_lines* method explicitly
- **hide_vertical** (*bool*) – Boolean to control whether the vertical lines should be hidden. If None, the default class attribute is used
- **hide_horizontal** (*bool*) – Boolean to control whether the horizontal lines should be hidden. If None, the default class attribute is used
- ****kwargs** – Any other keyword argument that is passed to the matplotlib.pyplot.plot() function

- **dtype** (*object*) – The data type for the data conversion
- **message** (*str*) – The message to display in the dialog
- **label** (*str*) – The label to how this value should be named
- **value** (*float*) – The initial value to use

Methods

ask_for_value([val, label])	Ask for a value for the cross mark
on_release(event, *args, **kwargs)	Release the mark and unselect it

ask_for_value(*val=None*, *label=None*)
Ask for a value for the cross mark

This method opens a QInputDialog to ask for a new `value`

Parameters

- **val** (*float*) – The initial value
- **label** (*str*) – the name of what to ask for

on_release(*event*, **args*, ***kwargs*)
Release the mark and unselect it

Parameters

- **event** (*matplotlib.backend_bases.MouseEvent*) – The mouseevent that releases the mark
- **force** (*bool*) – If True, the mark is released although it does not contain the *event*
- **connected** (*bool*) – If True, connected marks that should maintain a constant x- and y-distance are released, too
- **draw** (*bool*) – If True, the figure is drawn
- ****kwargs** (**args,*) – Any other parameter that is passed to the connected lines

class straditize.cross_mark.**DraggableVLine**(*x*, *ax=None*, **args*, ***kwargs*)
Bases: *straditize.cross_mark.CrossMarks*

A draggable vertical line

Parameters

- **x** (*float*) – The x-position for the vertical line
- **ax** (*matplotlib.axes.Axes*) – The matplotlib axes
- **idx_h** (*pandas.Index*) – The index for the horizontal coordinates. If not provided, we use a continuous movement along x.
- **idx_v** (*pandas.Index*) – The index for the vertical coordinates. If not provided, we use a continuous movement along y.
- **xlim** (*tuple of floats (xmin, xmax)*) – The minimum and maximum x value for the lines
- **ylim** (*tuple for floats (ymin, ymax)*) – The minimum and maximum y value for the lines
- **select_props** (*color*) – The line properties for selected marks

- **auto_hide** (*bool*) – If True, the lines are hidden if they are not selected.
- **connected_artists** (*list of artists*) – List of artists whose properties should be changed to *select_props* when this marks is selected
- **lock** (*bool*) – If True, at most one mark can be selected at a time
- **draw_lines** (*bool*) – If True, the cross mark lines are drawn. Otherwise, you must call the *draw_lines* method explicitly
- **hide_vertical** (*bool*) – Boolean to control whether the vertical lines should be hidden. If None, the default class attribute is used
- **hide_horizontal** (*bool*) – Boolean to control whether the horizontal lines should be hidden. If None, the default class attribute is used
- ****kwargs** – Any other keyword argument that is passed to the `matplotlib.pyplot.plot()` function

Attributes

hide_horizontal	bool(x) -> bool
y	The y-position of the mark

Methods

set_visible(b)	Set the visibility of the mark

hide_horizontal = True

set_visible(*b*)
Set the visibility of the mark

Parameters **b** (*bool*) – If False, hide all horizontal and vertical lines, and the connected_artists

property y
The y-position of the mark

class straditize.cross_mark.**DraggableVLineText**(**args*, ***kwargs*)
Bases: *straditize.cross_mark.DraggableVLine*

A CrossMarks that opens a QInputDialog after changing the position

Parameters

- **x** (*float*) – The x-position for the vertical line
- **ax** (*matplotlib.axes.Axes*) – The matplotlib axes
- **idx_h** (*pandas.Index*) – The index for the horizontal coordinates. If not provided, we use a continuous movement along x.
- **idx_v** (*pandas.Index*) – The index for the vertical coordinates. If not provided, we use a continuous movement along y.
- **xlim** (*tuple of floats (xmin, xmax)*) – The minimum and maximum x value for the lines
- **ylim** (*tuple for floats (ymin, ymax)*) – The minimum and maximum y value for the lines
- **select_props** (*color*) – The line properties for selected marks

- **auto_hide** (*bool*) – If True, the lines are hidden if they are not selected.
- **connected_artists** (*list of artists*) – List of artists whose properties should be changed to *select_props* when this marks is selected
- **lock** (*bool*) – If True, at most one mark can be selected at a time
- **draw_lines** (*bool*) – If True, the cross mark lines are drawn. Otherwise, you must call the *draw_lines* method explicitly
- **hide_vertical** (*bool*) – Boolean to control whether the vertical lines should be hidden. If None, the default class attribute is used
- **hide_horizontal** (*bool*) – Boolean to control whether the horizontal lines should be hidden. If None, the default class attribute is used
- ****kwargs** – Any other keyword argument that is passed to the `matplotlib.pyplot.plot()` function
- **dtype** (*object*) – The data type for the data conversion
- **message** (*str*) – The message to display in the dialog
- **label** (*str*) – The label to how this value should be named
- **value** (*float*) – The initial value to use

Methods

ask_for_value([val, label])	Ask for a value for the cross mark
on_release(event, *args, **kwargs)	Release the mark and unselect it

ask_for_value(*val=None*, *label=None*)
Ask for a value for the cross mark

This method opens a QInputDialog to ask for a new `value`

Parameters

- **val** (*float*) – The initial value
- **label** (*str*) – the name of what to ask for

on_release(*event*, **args*, ***kwargs*)
Release the mark and unselect it

Parameters

- **event** (*matplotlib.backend_bases.MouseEvent*) – The mouseevent that releases the mark
- **force** (*bool*) – If True, the mark is released although it does not contain the *event*
- **connected** (*bool*) – If True, connected marks that should maintain a constant x- and y-distance are released, too
- **draw** (*bool*) – If True, the figure is drawn
- ****kwargs** (**args,*) – Any other parameter that is passed to the connected lines

straditize.evaluator module

Evaluator class for the straditize algorithms

Classes

BaselineScenario([output_dir])	The baseline evaluation scenario for straditize with data from POLNET
BlackWhiteScenario([output_dir])	An evaluation scenario with a binary (black and white) image
DPI150Scenario([output_dir])	Another evaluation scenario but with a resolution of 150 dpi
DPI600Scenario([output_dir])	Another evaluation scenario but with a resolution of 600 dpi
ExaggerationsEvaluator(*args, **kwargs)	An evaluator with exaggerations
ExaggerationsScenario([output_dir])	An evaluation scenario with an exaggerated plot of low percentages
NoVerticalsEvaluator(data, *args[, name, ...])	An evaluator for an image without y-axis
NoVerticalsScenario([output_dir])	An evaluation scenario without y-axes in the plot
StraditizeEvaluator(data, *args[, name, ...])	An evaluator for the straditize components

Functions

print_progressbar(iteration, total[, ...])	Print iterations progress
rmse(sim, ref)	Calculate the root mean squared error between simulation and reference

class straditize.evaluator.**BaselineScenario**(*output_dir='.'*)

Bases: object

The baseline evaluation scenario for straditize with data from POLNET

This class uses the default settings of the *StraditizeEvaluator* and runs the analysis for a given dataset from POLNET. **Methods**

export_evaluator(evaluator, *args, **kwargs)	
init_evaluator(name, data, *args, **kwargs)	Initialize an evaluator for a given data set
run(data[, processes])	

Attributes

index_names	Built-in mutable sequence.

export_evaluator(*evaluator*, **args*, ***kwargs*)

index_names = ['e_', 'ntaxa', 'nsamples']

init_evaluator(*name*, *data*, **args*, ***kwargs*)

Initialize an evaluator for a given data set

run(*data*, *processes=None*)

class straditize.evaluator.**BlackWhiteScenario**(*output_dir='.'*)

Bases: *straditize.evaluator.BaselineScenario*

An evaluation scenario with a binary (black and white) image **Methods**

init_evaluator(*args, **kwargs)	Initialize an evaluator for a given data set

init_evaluator(**args*, ***kwargs*)
Initialize an evaluator for a given data set

class straditize.evaluator.**DPI150Scenario**(*output_dir='.'*)
Bases: *straditize.evaluator.BaselineScenario*

Another evaluation scenario but with a resolution of 150 dpi **Methods**

export_evaluator(*args, **kwargs)

export_evaluator(**args*, ***kwargs*)

class straditize.evaluator.**DPI600Scenario**(*output_dir='.'*)
Bases: *straditize.evaluator.BaselineScenario*

Another evaluation scenario but with a resolution of 600 dpi **Methods**

export_evaluator(*args, **kwargs)

export_evaluator(**args*, ***kwargs*)

class straditize.evaluator.**ExaggerationsEvaluator**(**args*, ***kwargs*)
Bases: *straditize.evaluator.StraditizeEvaluator*

An evaluator with exaggerations

Parameters

- **df** (*pandas.DataFrame*) – The dataframe containing the data to plot.
- **group_func** (*function*) – A function that groups the columns in the input *df* together. It must accept the name of a column and return the corresponding group name:

```
def group_func(col_name: str):
    return "name of it's group"
```

  If this parameter is not specified, each column will be assigned to the *'nogroup'* group that can then be used in the other parameters, such as *formatoptions* and *percentages*. Each group may also be divided into *subgroups* (see below), in this case, the *group_func* should return the corresponding subgroup.
- **formatoptions** (*dict*) – The formatoption for each group. Depending on the chosen plot method, this contains the formatoptions for the psyplot plotter.
- **ax** (*matplotlib.axes.Axes*) – The matplotlib axes to plot on. New axes will be created that cover all the space of the given axes. If this parameter is not specified and *fig* is None, a new matplotlib figure is created with a new matplotlib axes.
- **thresh** (*float*) – A minimum number between 0 and 100 (by default 1%) that a *percentages* column has to fullfil in order to be included in the plot. If a variable is always below this threshold, it will not be included in the figure
- **percentages** (*list of str or bool*) – The group names (see *group_func*) that represent percentage values. This variables will be visualized using an area plot and can be rescaled to sum up to 100% using the *calculate_percentages* parameter. This parameter can also be set to True if all groups shall be considered as percentage data

- **exclude** (*list of str*) – Either group names of column names in *df* that should be excluded in the plot
- **widths** (*dict*) – A mapping from group name to it's relative width in the plot. The values of this mapping should some up to 1, e.g.:

```
widths = {'group1': 0.3, 'group2': 0.5, 'group3': 0.2}
```

- **calculate_percentages** (*bool or list of str*) – If True, rescale the groups mentioned in the *percentages* parameter to sum up to 100%. In case of a list of str, this parameter represents the group (or variable) names that shall be used for the normalization
- **min_percentage** (*float*) – The minimum percentage (between 0 and 100) that should be covered by variables displaying *percentages* data. Each plot in one of the *percentages* groups will have at least have a xlim from 0 to *min_percentage*
- **trunc_height** (*float*) – A float between 0 and 1. The fraction of the *ax* that should be reserved for the group titles.
- **fig** (*matplotlib.Figure*) – The matplotlib figure to draw the plot on. If neither *ax* nor *fig* is specified, a new figure will be created.
- **all_in_one** (*list of str*) – The groups mentioned in this parameter will all be plotted in one single axes whereas the default is to plot each variable in a separate plot
- **stacked** (*list of str*) – The groups mentioned in this parameter will all be plotted in one single axes, stacked onto each other
- **summed** (*list of str*) – The groups (or subgroups) mentioned in this parameter will be summed and an extra plot will be appended to the right of the stratigraphic diagram
- **use_bars** (*list of str or bool*) – The variables specified in this parameter (or all variables if *use_bars* is `True`) will be visualized by a bar diagram, instead of a line or area plot.
- **subgroups** (*dict*) – A mapping from group name to a list of subgroups, e.g.:

```
subgroups = {'Pollen': ['Trees', 'Shrubs']}
```

to divide an overarching group into subgroups.

Methods

init_stradi(*args, **kwargs)	

init_stradi(**args*, ***kwargs*)

class straditize.evaluator.**ExaggerationsScenario**(*output_dir='.'*)

Bases: *straditize.evaluator.BaselineScenario*

An evaluation scenario with an exaggerated plot of low percentages **Methods**

init_evaluator(name, data, *args, **kwargs)	Initialize an evaluator for a given data set

init_evaluator(*name*, *data*, **args*, ***kwargs*)

Initialize an evaluator for a given data set

class straditize.evaluator.**NoVerticalsEvaluator**(*data*, **args*, *name='data'*, *axislinestyle={'bottom': '-', 'left': '-', 'right': '-', 'top': '-'}*, ***kwargs*)

Bases: *straditize.evaluator.StraditizeEvaluator*

An evaluator for an image without y-axis

Parameters

- **df** (*pandas.DataFrame*) – The dataframe containing the data to plot.
- **group_func** (*function*) – A function that groups the columns in the input *df* together. It must accept the name of a column and return the corresponding group name:

```
def group_func(col_name: str):
    return "name of it's group"
```

  If this parameter is not specified, each column will be assigned to the *'nogroup'* group that can then be used in the other parameters, such as *formatoptions* and *percentages*. Each group may also be divided into *subgroups* (see below), in this case, the *group_func* should return the corresponding subgroup.
- **formatoptions** (*dict*) – The formatoption for each group. Depending on the chosen plot method, this contains the formatoptions for the psyplot plotter.
- **ax** (*matplotlib.axes.Axes*) – The matplotlib axes to plot on. New axes will be created that cover all the space of the given axes. If this parameter is not specified and *fig* is None, a new matplotlib figure is created with a new matplotlib axes.
- **thresh** (*float*) – A minimum number between 0 and 100 (by default 1%) that a *percentages* column has to fullfil in order to be included in the plot. If a variable is always below this threshold, it will not be included in the figure
- **percentages** (*list of str or bool*) – The group names (see *group_func*) that represent percentage values. This variables will be visualized using an area plot and can be rescaled to sum up to 100% using the *calculate_percentages* parameter. This parameter can also be set to True if all groups shall be considered as percentage data
- **exclude** (*list of str*) – Either group names of column names in *df* that should be excluded in the plot
- **widths** (*dict*) – A mapping from group name to it's relative width in the plot. The values of this mapping should some up to 1, e.g.:

```
widths = {'group1': 0.3, 'group2': 0.5, 'group3': 0.2}
```

- **calculate_percentages** (*bool or list of str*) – If True, rescale the groups mentioned in the *percentages* parameter to sum up to 100%. In case of a list of str, this parameter represents the group (or variable) names that shall be used for the normalization
- **min_percentage** (*float*) – The minimum percentage (between 0 and 100) that should be covered by variables displaying *percentages* data. Each plot in one of the *percentages* groups will have at least have a xlim from 0 to *min_percentage*
- **trunc_height** (*float*) – A float between 0 and 1. The fraction of the *ax* that should be reserved for the group titles.
- **fig** (*matplotlib.Figure*) – The matplotlib figure to draw the plot on. If neither *ax* nor *fig* is specified, a new figure will be created.
- **all_in_one** (*list of str*) – The groups mentioned in this parameter will all be plotted in one single axes whereas the default is to plot each variable in a separate plot
- **stacked** (*list of str*) – The groups mentioned in this parameter will all be plotted in one single axes, stacked onto each other

- **summed** (*list of str*) – The groups (or subgroups) mentioned in this parameter will be summed and an extra plot will be appended to the right of the stratigraphic diagram
- **use_bars** (*list of str or bool*) – The variables specified in this parameter (or all variables if *use_bars* is `True`) will be visualized by a bar diagram, instead of a line or area plot.
- **subgroups** (*dict*) – A mapping from group name to a list of subgroups, e.g.:

```
subgroups = {'Pollen': ['Trees', 'Shrubs']}
```

to divide an overarching group into subgroups.

Methods

evaluate_column_starts([close, base])	
evaluate_yaxes_removal([close])	
export(*args, **kwargs)	

evaluate_column_starts(*close=True*, *base='starts_'*)

evaluate_yaxes_removal(*close=True*)

export(**args*, ***kwargs*)

class straditize.evaluator.**NoVerticalsScenario**(*output_dir='.'*)

Bases: *straditize.evaluator.BaselineScenario*

An evaluation scenario without y-axes in the plot **Methods**

init_evaluator(name, data, *args, **kwargs)	Initialize an evaluator for a given data set

init_evaluator(*name*, *data*, **args*, ***kwargs*)

Initialize an evaluator for a given data set

class straditize.evaluator.**StraditizeEvaluator**(*data*, **args*, *name='data'*, *axislinestyle={'bottom': '-', 'left': '-', 'right': '-', 'top': '-'}*, ***kwargs*)

Bases: `object`

An evaluator for the straditize components

Parameters

- **df** (*pandas.DataFrame*) – The dataframe containing the data to plot.
- **group_func** (*function*) – A function that groups the columns in the input *df* together. It must accept the name of a column and return the corresponding group name:

```
def group_func(col_name: str):
    return "name of it's group"
```

If this parameter is not specified, each column will be assigned to the *'nogroup'* group that can then be used in the other parameters, such as *formatoptions* and *percentages*. Each group may also be divided into *subgroups* (see below), in this case, the *group_func* should return the corresponding subgroup.

- **formatoptions** (*dict*) – The formatoption for each group. Depending on the chosen plot method, this contains the formatoptions for the psyplot plotter.

- **ax** (*matplotlib.axes.Axes*) – The matplotlib axes to plot on. New axes will be created that cover all the space of the given axes. If this parameter is not specified and *fig* is None, a new matplotlib figure is created with a new matplotlib axes.
- **thresh** (*float*) – A minimum number between 0 and 100 (by default 1%) that a *percentages* column has to fullfil in order to be included in the plot. If a variable is always below this threshold, it will not be included in the figure
- **percentages** (*list of str or bool*) – The group names (see *group_func*) that represent percentage values. This variables will be visualized using an area plot and can be rescaled to sum up to 100% using the *calculate_percentages* parameter. This parameter can also be set to True if all groups shall be considered as percentage data
- **exclude** (*list of str*) – Either group names of column names in *df* that should be excluded in the plot
- **widths** (*dict*) – A mapping from group name to it's relative width in the plot. The values of this mapping should some up to 1, e.g.:

```
widths = {'group1': 0.3, 'group2': 0.5, 'group3': 0.2}
```

- **calculate_percentages** (*bool or list of str*) – If True, rescale the groups mentioned in the *percentages* parameter to sum up to 100%. In case of a list of str, this parameter represents the group (or variable) names that shall be used for the normalization
- **min_percentage** (*float*) – The minimum percentage (between 0 and 100) that should be covered by variables displaying *percentages* data. Each plot in one of the *percentages* groups will have at least have a xlim from 0 to *min_percentage*
- **trunc_height** (*float*) – A float between 0 and 1. The fraction of the *ax* that should be reserved for the group titles.
- **fig** (*matplotlib.Figure*) – The matplotlib figure to draw the plot on. If neither *ax* nor *fig* is specified, a new figure will be created.
- **all_in_one** (*list of str*) – The groups mentioned in this parameter will all be plotted in one single axes whereas the default is to plot each variable in a separate plot
- **stacked** (*list of str*) – The groups mentioned in this parameter will all be plotted in one single axes, stacked onto each other
- **summed** (*list of str*) – The groups (or subgroups) mentioned in this parameter will be summed and an extra plot will be appended to the right of the stratigraphic diagram
- **use_bars** (*list of str or bool*) – The variables specified in this parameter (or all variables if *use_bars* is `True`) will be visualized by a bar diagram, instead of a line or area plot.
- **subgroups** (*dict*) – A mapping from group name to a list of subgroups, e.g.:

```
subgroups = {'Pollen': ['Trees', 'Shrubs']}
```

to divide an overarching group into subgroups.

Attributes

all_results
column_bounds
column_ends
column_starts

Continued on next page

Table 171 – continued from previous page

data	
data_xlim	
data_ylim	
dpi	
full_df	
height	
results	
results_column	The column name in `all_results`
summed_perc	
transformed_data	The `data` in pixel coordinates
width	

Methods

close()	
evaluate_column_starts([close, base])	
evaluate_full([close])	
evaluate_sample_accuracy([close, stradi, base])	
evaluate_sample_position([close, stradi, base])	
evaluate_yaxes_removal([close])	
export(filepath[, dpi, labels])	
from_polnet(data, *args, **kwargs)	
init_stradi([datalim, columns, names, ...])	
run()	Run all evaluations
set_xtranslation(stradi)	

property all_results

close()

property column_bounds

property column_ends

property column_starts

property data

property data_xlim

property data_ylim

property dpi

evaluate_column_starts(*close=True*, *base='starts_'*)

evaluate_full(*close=True*)

evaluate_sample_accuracy(*close=True*, *stradi=None*, *base='samples_'*)

evaluate_sample_position(*close=True*, *stradi=None*, *base='samples_'*)

evaluate_yaxes_removal(*close=True*)

export(*filepath*, *dpi=300*, *labels={}*)

classmethod from_polnet(*data*, **args*, ***kwargs*)

property full_df

property height

init_stradi(*datalim=True*, *columns=True*, *names=True*, *digitize=True*, *samples=True*, *axes=False*)

property results

property results_column
The column name in *all_results*

run()
Run all evaluations

set_xtranslation(*stradi*)

property summed_perc

property transformed_data
The *data* in pixel coordinates

property width

straditize.evaluator.**print_progressbar**(*iteration*, *total*, *prefix=''*, *suffix=''*, *length=100*, *fill='*')
Print iterations progress

Taken from https://stackoverflow.com/a/34325723

Parameters

- **iteration** (*int*) – current iteration
- **total** (*int*) – total iterations
- **prefix** (*str*) – prefix string
- **suffix** (*str*) – suffix string
- **decimals** (*int*) – positive number of decimals in percent complete
- **length** (*int*) – character length of bar
- **fill** (*str*) – bar fill character

straditize.evaluator.**rmse**(*sim*, *ref*)
Calculate the root mean squared error between simulation and reference

Parameters

- **sim** (*np.ndarray*) – The simluated data
- **ref** (*np.ndarray*) – The reference data

straditize.label_selection module

Module for the *LabelSelection* class

This module defines the *LabelSelection* class, a base class for the *straditize.straditizer.Straditizer* *straditize.binary.DataReader* classes. This class implements the features to select parts of an image and deletes them. The *straditize.widgets.selection_toolbar.SelectionToolbar* interfaces with instances of this class.

Disclaimer

This program is free software: you can redistribute it and/or modify it under the terms of the GNU General Public License as published by the Free Software Foundation, either version 3 of the License, or (at your option) any later version.

This program is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU General Public License for more details.

You should have received a copy of the GNU General Public License along with this program. If not, see <https://www.gnu.org/licenses/>.

Classes

LabelSelection	Class to provide selection functionalities for an image

class straditize.label_selection.**LabelSelection**

Bases: object

Class to provide selection functionalities for an image

This class provides functionalities to select features in an image. A new selection can be started through *enable_label_selection()* method and selected parts can be removed through the *remove_selected_labels()* method.

A 2D boolean mask of the selected pixels can be accessed through the *selected_part* attribute.

This class generally assumes that the array for the selection is a 2D integer array, e.g. obtained from the `skimage.morphology.label()` function.

The selection of labels is handled through the colormap. The selection is displayed as a matplotlib image on the `ax` attribute of this instance. If the color for one label is equal to the *cselect* color, it is considered as selected. Additionally every cell that has a value greater than the original number of labels is considered to be selected.

Cells with a value of -1 are not selected and cells with a value of 0 cannot be selected. **Attributes**

cselect	The RGBA color for selected polygons
cunselect	The RGBA color for unselected polygons
label_arrs	List of attribute names of arrays that should be modified if the labels are about to be removed.
remove_callbacks	Functions that shall be called before the labels are removed.
selected_labeled_part	The selected part as a 2D boolean mask
selected_labels	A list of selected labels in the selection array
selected_part	The selected part as a 2D boolean mask

Methods

copy_cmap(cmap_src, colors)	Copy a colormap with replaced colors
disable_label_selection([remove])	Disable the label selection
enable_label_selection(arr, ncolors[, img, ...])	Start the selection of labels
get_default_cmap(ncolors)	The default colormap for binary images
highlight_small_selections([n])	Highlight labels that cover only a small portion of cells
pick_label(event)	Pick the label selected by the given mouseevent

Continued on next page

Table 175 – continued from previous page

remove_selected_labels([disable])	Remove the selected parts of the diagram
remove_small_selection_ellipses()	Remove the ellipes for small features
select_all_labels()	Select the entire array
select_all_other_labels()	Invert the selection
select_labels(selected)	Select a list of labels
unselect_all_labels()	Clear the selection

cid_select = None

static copy_cmap(*cmap_src*, *colors*)
Copy a colormap with replaced colors

This function creates a method that has the same name and the same *under*, *over* and *bad* values as the given *cmap_src* but with replaced colors

Parameters

- **cmap_src** (*matplotlib.colors.Colormap*) – The source colormap
- **colors** (*np.ndarray*) – The colors for the colormap

Returns The new colormap

Return type matplotlib.colors.Colormap

cselect = [1.0, 0.0, 0.0, 1.0]
The RGBA color for selected polygons

cunselect = [0.0, 0.0, 0.0, 0.0]
The RGBA color for unselected polygons

disable_label_selection(*remove=None*)
Disable the label selection

This will disconnect the *pick_event* and remove the selection images

Parameters **remove** (*bool*) – Whether to remove the selection image from the plot. If None, the `_remove` attribute is used

See also:

enable_label_selection(), *remove_selected_labels()*

enable_label_selection(*arr*, *ncolors*, *img=None*, *set_picker=False*, ***kwargs*)
Start the selection of labels

Parameters

- **arr** (*2D np.ndarray of dtype int*) – The labeled array that contains the features to select.
- **ncolors** (*int*) – The maximum of the labels in *arr*
- **img** (*matplotlib image*) – The image for the selection. If not provided, a new image is created
- **set_picker** (*bool*) – If True, connect the matplotlib pick_event to the *pick_label()* method

See also:

disable_label_selection(), *remove_selected_labels()*

get_default_cmap(*ncolors*)
The default colormap for binary images

highlight_small_selections(*n=20*)
Highlight labels that cover only a small portion of cells

This method uses the `skimage.morphology.remove_small_objects()` to detect and highlight small features in the diagram. Each feature will be highlighted through an ellipsis around it.

See also:

remove_small_selection_ellipses()

label_arrs = []
List of attribute names of arrays that should be modified if the labels are about to be removed. The attributes might be callable and should then provide the array

pick_label(*event*)
Pick the label selected by the given mouseevent

remove_callbacks = None
Functions that shall be called before the labels are removed. The keys must be the attributes in the `label_attrs` list, values must be list of function that accept too arguments, the array and the boolean mask highlighting the cells that will be set to 0

remove_selected_labels(*disable=False*)
Remove the selected parts of the diagram

This method will call the callbackes in the *remove_callbacks* attribute for all the attributes in the *label_arrs* list.

Parameters **disable** (*bool*) – If True, call the *disable_label_selection()* method at the end

See also:

enable_label_selection(), *disable_label_selection()*

remove_small_selection_ellipses()
Remove the ellipes for small features

Removes the ellipses plotted by the *highlight_small_selections()* method

select_all_labels()
Select the entire array

select_all_other_labels()
Invert the selection

select_labels(*selected*)
Select a list of labels

Parameters **selected** (*np.ndarray*) – The numpy array of labels that should be selected

property selected_labeled_part
The selected part as a 2D boolean mask

property selected_labels
A list of selected labels in the selection array

property selected_part
The selected part as a 2D boolean mask

unselect_all_labels()
Clear the selection

straditize.magnifier module

Magnifier class for an image

Disclaimer

Classes

Magnifier(ax_src[, ax])	A magnification of a matplotlib axes

class straditize.magnifier.**Magnifier**(*ax_src*, *ax=None*, **args*, ***kwargs*)

Bases: object

A magnification of a matplotlib axes

It zooms into the region where the mouse pointer is, when it enters the source axes. The appearance of the plot is defined by the *make_plot()* method. **Methods**

adjust_limits(zoom_val)	
close()	Close the magnifier and the associated plots
disconnect()	
enable_zoom()	
make_plot(image, *args, **kwargs)	
onenter(event)	
onleave(event)	
onmotion(event)	

Attributes

dx
dy

adjust_limits(*zoom_val*)

ax = None

cid_enter = None

cid_leave = None

cid_motion = None

close()

Close the magnifier and the associated plots

disconnect()

property dx

property dy

enable_zoom()

make_plot(*image*, **args*, ***kwargs*)

onenter(*event*)

onleave(*event*)

onmotion(*event*)

straditize.straditizer module

Core module of the Straditizer class

This module defines the *Straditizer* class, the main object to digitize a stratigraphic diagram

Disclaimer

Classes

Straditizer(image[, ax, plot, attrs])	An object to digitize a stratigraphic diagram

Functions

format_coord_func(ax, ref)	Create a function that can replace the

class straditize.straditizer.**Straditizer**(*image*, *ax=None*, *plot=True*, *attrs=None*)

Bases: *straditize.label_selection.LabelSelection*

An object to digitize a stratigraphic diagram

One instance of a straditizer manages the digitization of a `PIL.Image.Image` hold in the `image` attribute.

To create a new Straditizer instance, you can either provide a `PIL.Image.Image` to the class constructor (i.e. `stradi = Straditizer(image)`) or you use the *from_dataset()* method.

The reader for the diagram part can be accessed through the *data_reader* attribute, the reader for the column names through the *colnames_reader* attribute.

Parameters

- **image** (*PIL.Image.Image or np.ndarray*) – The image file to process. A numpy array should be 3D with shape `(Y, X, 4)`, where the last channel [..., -1] should represent the alpha channel. A PIL.Image.Image will be converted to a RGBA image (if not already)

- **ax** (*matplotlib.axes.Axes*) – The matplotlib axes. If None, a new one will be created
- **attrs** (dict or pandas.DataFrame) – The attributes for this straditizer

Methods

adjust_lims()	
adjust_lims_after_resize([event])	
adjust_lims_after_zoom(ax)	
align_columns()	Shift the columns after the marks have been moved
close()	
create_magni_marks(marks)	Copy the created marks to the magnifiers axes
create_variable(ds, vname, data, **kwargs)	Insert the data into a variable in an xr.Dataset
data2px_y(coord)	Transform the data coordinates into pixel coordinates
digitize_diagram()	
draw_data_box()	
draw_figure()	
from_dataset(ds[, ax, plot])	Create a new *Straditizer* from a dataset
get_attr(key)	
get_labels([categorize])	
get_reader_for_column(col)	
guess_data_lims([fraction])	Guess the limits of the diagram part
image_array()	
init_reader([reader_type, ax])	
load(fname[, ax, plot])	
marks_for_column_ends([threshold])	
marks_for_column_starts([threshold])	
marks_for_data_selection([nums, fraction, ...])	
marks_for_occurences()	Create marks for editing the occurences
marks_for_samples()	
marks_for_samples_sep([nrows])	
marks_for_vertical_alignment()	Create marks for vertical alignment of the columns
marks_for_x_values([at_col_start])	Create two marks for selecting the x-values
marks_for_y_values()	Create two marks for selecting the x-values
plot_image([ax])	
px2data_y(coord)	Transform the pixel coordinates into data coordinates
remove_data_box()	Remove the data_box
remove_marks()	Remove any drawn marks
reset_image(image[, reader])	Reset the straditizer image
save(fname)	Dump the *Straditizer* instance to a file
set_attr(key, value)	Update an attribute in the attrs
show_data_diagram()	
show_full_image()	
to_dataset([ds])	All the necessary data as a xarray.Dataset
update_column_ends()	
update_column_starts()	
update_data_part()	
update_image(arr, mask)	Update the image from the given 3D-array
update_occurences([remove])	Set the occurences from the given marks
update_samples([remove])	

Continued on next page

Table 181 – continued from previous page

update_samples_sep([remove])	
update_xvalues()	
update_yvalues()	

Attributes

adjusting	True if xlim and ylim are adjusted
attrs	pandas.DataFrame. The attributes of this straditizer
attrs_dict	
ax	The matplotlib axes
block_signals	Block the emitting of signals of this instance
colnames_reader	The *straditize.colnames.ColNamesReader* for reading the column
column_indexes	The horizontal indexes for each column
data_reader	The *straditize.binary.DataReader* instance to digitize the
data_xlim	
data_ylim	
fig	
final_df	
full_df	
indexes	
label_arrs	Built-in mutable sequence.
magni	The *straditize.magnifier.Magnifier* for the diagram image
mark_added	A signal that is emitted if a mark has been added.
mark_cids	set() -> new empty set object
mark_removed	A signal that is emitted, if a mark has been removed.
nc_meta	dict() -> new empty dictionary
valid_attrs	
yaxis_px	

adjust_lims()

adjust_lims_after_resize(*event=None*)

adjust_lims_after_zoom(*ax*)

property adjusting
True if xlim and ylim are adjusted

align_columns()
Shift the columns after the marks have been moved

This method should be called after the *marks_for_vertical_alignment()* method to align the columns

attrs = None
pandas.DataFrame. The attributes of this straditizer

property attrs_dict

ax = None
The matplotlib axes

property block_signals
Block the emitting of signals of this instance

close()

property colnames_reader
The *straditize.colnames.ColNamesReader* for reading the column names

property column_indexes
The horizontal indexes for each column

create_magni_marks(*marks*)
Copy the created marks to the magnifiers axes

create_variable(*ds*, *vname*, *data*, ***kwargs*)
Insert the data into a variable in an `xr.Dataset`

data2px_y(*coord*)
Transform the data coordinates into pixel coordinates

Parameters **coord** (*1D np.ndarray*) – The coordinate values

Returns The numpy array with transformed coordinates

Return type np.ndarray

data_reader = None
The *straditize.binary.DataReader* instance to digitize the data

property data_xlim

property data_ylim

digitize_diagram()

draw_data_box()

draw_figure()

property fig

fig_h = None

fig_w = None

property final_df

classmethod from_dataset(*ds*, *ax=None*, *plot=True*)
Create a new *Straditizer* from a dataset

This method uses a dataset that has been exported with the *to_dataset()* method to intialize a new reader

property full_df

get_attr(*key*)

get_labels(*categorize=1*)

get_reader_for_column(*col*)

guess_data_lims(*fraction=0.7*)
Guess the limits of the diagram part

Parameters **fraction** (*float*) – The smallest fraction that has to be covered for a cell to be considered as a corner

Returns

- *np.array* – xmin and xmax of the diagram part (see *data_xlim*)
- *np.array* – ymin and ymax of the diagram part (see *data_ylim*)

image_array()

property indexes

init_reader(*reader_type='area'*, *ax=None*, ***kwargs*)

label_arrs = ['image_array']

classmethod load(*fname*, *ax=None*, *plot=True*)

magni = None
The *straditize.magnifier.Magnifier* for the diagram image

mark_added
A signal that is emitted if a mark has been added. Functions are expected to accept one argument, the newly created *straditize.cross_mark.CrossMarks* instance

mark_cids = {}

mark_removed
A signal that is emitted, if a mark has been removed. Functions are expected to accept one argument, the removed *straditize.cross_mark.CrossMarks* instance

marks_for_column_ends(*threshold=None*)

marks_for_column_starts(*threshold=None*)

marks_for_data_selection(*nums=2*, *fraction=0.7*, *guess_lims=True*)

marks_for_occurences()
Create marks for editing the occurences

marks_for_samples()

marks_for_samples_sep(*nrows=3*)

marks_for_vertical_alignment()
Create marks for vertical alignment of the columns

This method creates one mark for each column. These marks should then be moved to positions that should be on the same vertical level. After that, the *align_columns()* method has to be called

marks_for_x_values(*at_col_start=True*)
Create two marks for selecting the x-values

Parameters **at_col_start** (*bool*) – If True, and no translation has yet been performed, create a mark at the column start and ask for the corresponding value

marks_for_y_values()
Create two marks for selecting the x-values

nc_meta = {'axis': {'long_name': 'Axis coordinate'}, 'data_lims': {'dims': ('axis'

plot_image(*ax=None*, ***kwargs*)

px2data_y(*coord*)
Transform the pixel coordinates into data coordinates

Parameters **coord** (*1D np.ndarray*) – The coordinate values in pixels

Returns The numpy array with transformed coordinates

Return type np.ndarray

remove_data_box()
Remove the data_box

remove_marks()
Remove any drawn marks

reset_image(*image*, *reader=False*)
Reset the straditizer image

Parameters

- **image** (*PIL.Image.Image*) – The new image to use
- **reader** (*bool*) – If True, the image of the data reader will be replaced, too

save(*fname*)
Dump the *Straditizer* instance to a file

Parameters **fname** (*str*) – The file name where to save the instance

set_attr(*key*, *value*)
Update an attribute in the *attrs*

show_data_diagram()

show_full_image()

to_dataset(*ds=None*)
All the necessary data as a xarray.Dataset

Parameters **ds** (*xarray.Dataset*) – The dataset in which to insert the data. If None, a new one will be created

Returns Either the given *ds* or a new xarray.Dataset instance

Return type xarray.Dataset

update_column_ends()

update_column_starts()

update_data_part()

update_image(*arr*, *mask*)
Update the image from the given 3D-array

Parameters

- **arr** (*3D np.ndarray of dtype float*) – The image array
- **mask** (boolean mask of the same shape as *arr*) – The mask of features that shall be set to 0 in *arr*

update_occurences(*remove=True*)
Set the occurences from the given marks

update_samples(*remove=True*)

update_samples_sep(*remove=True*)

update_xvalues()

update_yvalues()

property valid_attrs

yaxis_data = None

property yaxis_px

straditize.straditizer.**format_coord_func**(*ax*, *ref*)

Create a function that can replace the `matplotlib.axes.Axes.format_coord()`

Parameters

- **ax** (*matplotlib.axes.Axes*) – The axes instance
- **ref** (*weakref.weakref*) – The reference to the *Straditizer* instance

Returns The function that can be used to replace *ax.format_coord*

Return type function

straditize.version module

Version string of straditize

Disclaimer

1.8 Changelog

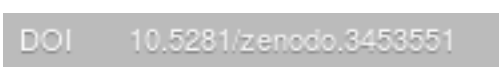
1.8.1 v0.1.3

Patch that forces the diagram limits to be integers.

1.8.2 v0.1.2

This release contains several small bug fixes, mainly for the bar data reader and for duplicated samples and occurences.

1.8.3 v0.1.1



This release has been approved by the Journal of Open Source Software (JOSS) in https://github.com/openjournals/joss-reviews/issues/1216

Added

- Changelog

Changed

- Thanks to the feedbacks from @ixjlyons and @sgrieve the installation instructions and some other documentation instructions have been improved (see the issues https://github.com/Chilipp/straditize/issues/1, https://github.com/Chilipp/straditize/issues/2, https://github.com/Chilipp/straditize/issues/3, https://github.com/Chilipp/straditize/issues/4 and https://github.com/Chilipp/straditize/issues/8)

CHAPTER

TWO

HOW TO CITE STRADITIZE

When using straditize, you should at least cite the publication in the Journal of Open Source Software:

Sommer, Philipp, Dilan Rech, Manuel Chevalier, and Basil A. S. Davis. Straditize: Digitizing Stratigraphic Diagrams. *Journal of Open Source Software* , vol. 4, no. 34, 34, The Open Journal, Feb. 2019, p. 1216, doi:10.21105/joss.01216, https://doi.org/10.21105/joss.00363.

`BibTex` - `EndNote`

Furthermore, each release of straditize is associated with a DOI using zenodo.org. If you want to cite a specific version or plugin, please refer to the releases page of straditize.

CHAPTER

THREE

INDICES AND TABLES

- genindex
- modindex
- search

BIBLIOGRAPHY

[Basil2007a] Davis, Basil A. S., and A. C. Stevenson. “The 8.2ka Event and Early-Mid Holocene Forests, Fires and Flooding in the Central Ebro Desert, NE Spain.” Quat. Sci. Rev. , vol. 26, no. 13-14, 2007, pp. 1695-712, doi:10.1016/j.quascirev.2007.04.007.

[Basil2007] Davis, Basil A. S., and A. C. Stevenson. “The 8.2ka Event and Early-Mid Holocene Forests, Fires and Flooding in the Central Ebro Desert, NE Spain.” Quat. Sci. Rev. , vol. 26, no. 13-14, 2007, pp. 1695-712, doi:10.1016/j.quascirev.2007.04.007.

PYTHON MODULE INDEX

S

INDEX

A

B

C

D

E

F

G

H

I

K

L

M

N

O

P

R

S

T

U

V

W

X

Y

Z