pyrotoolbox

Version 1.5b

Table of Contents

Welcome to pyrotoolbox's documentation!

This is a collection of different tools which are useful for data processing of PyroScience data. This module contains functions to parse data and to re-calculate it.

Installation

Please install a suitable python-distribution first. We recommend Anaconda-Python.

Afterwards you can install *pyrotoolbox* in the “Anaconda Command Prompt” by typing

```
pip install pyrotoolbox
```

Parser

This module contains parser to load data from different logfile formats.

The “parse” function is able to detect all possible input formats.

The return is for all functions a dataframe containing the data and a dictionary containing the parsed metadata. Independent of the input format the columns and metadata-names should be identical. Other functions in this module expect these naming conventions.

pyrotoolbox.parsers. parse (*fname : str*) → tuple [DataFrame , dict]

Reads any pyroscience textfile. Not .pyr files! Returns a dataframe and a dict with metadata.

Parameters :

fname – path to the textfile

pyrotoolbox.parsers. read_workbench (*fname : str*) → tuple [DataFrame , dict]

Loads and parses a Workbench file and returns a pandas DataFrame and a dictionary with metadata

Parameters :

fname – file name of the logfile

Returns :

(DataFrame, metadata-dict)

pyrotoolbox.parsers. read_fireplate_workbench (*fname : str*) → tuple [DataFrame , dict]

Loads and parses a Workbench file of a fireplate and returns a pandas DataFrame and a dictionary with metadata

Parameters :

fname – path to the lofile

Returns :

DataFrame, metadata-dict

pyrotoolbox.parsers. read_developertool (*fname : str*) → tuple [DataFrame , dict]

Loads and parses a logfile from the PyroDeveloperTool

Parameters :

fname – path to the logfile

Returns :

(DataFrame, metadata-dict)

pyrotoolbox.parsers. read_developertool_directory (*pattern : str = '*.txt'*)

parses all files matching the pattern (default * .txt) and returns 3 dictionaries

first dictionary is UID/Name-ChX -> List of Dataframes

second dictionary is UID/Name-ChX -> List of metadata-dicts

third dictionary is UID/Name-ChX -> List of filenames

Parameters :

pattern – files to load. Default: * .txt

pyrotoolbox.parsers. read_aquaphoxlogger (*fname : str*) → tuple [DataFrame , dict]

Loads and parses a logfile from an AquapHOx-Logger

Parameters :

fname – path to the logfile

Returns :

(DataFrame, metadata-dict)

pyrotoolbox.parsers. read_fsgo2 (*fname : str*) → tuple [DataFrame , dict]

Loads and parses a logfile from a FSGO2

Parameters :

fname – path to the logfile

Returns :

(DataFrame, metadata-dict)

pyrotoolbox.parsers. read_fdo2_logger (*fname : str*) → tuple [DataFrame , dict]

Loads and parses a logfile from the FDO2 Logger

Parameters :

fname – path to the logfile

Returns :

(DataFrame, metadata-dict)

Examples

```
[1]:
```

```
from pyrotoolbox import parse
%matplotlib inline
```

Workbench

```
[2]:
```

```
df, m = parse('ChannelData/A_Firesting Pro (4 Channels)_(A Ch.
1)_pH.txt')
```

[3]:

```
df
```

[3]:

date_time	time_s	pH	dphi	signal_intensity	ambient_light	R	status	sample_temperature
2024-12-11 08:31:03.546								
2024-12-11 08:31:03.647								
2024-12-11 08:31:03.992								
2024-12-11 08:31:04.147								
2024-12-11 08:31:04.246								
...								
2024-12-11 08:55:36.025								
2024-12-11 08:55:36.126								
2024-12-11 08:55:36.226								
2024-12-11 08:55:36.326								
2024-12-11 08:55:36.427								

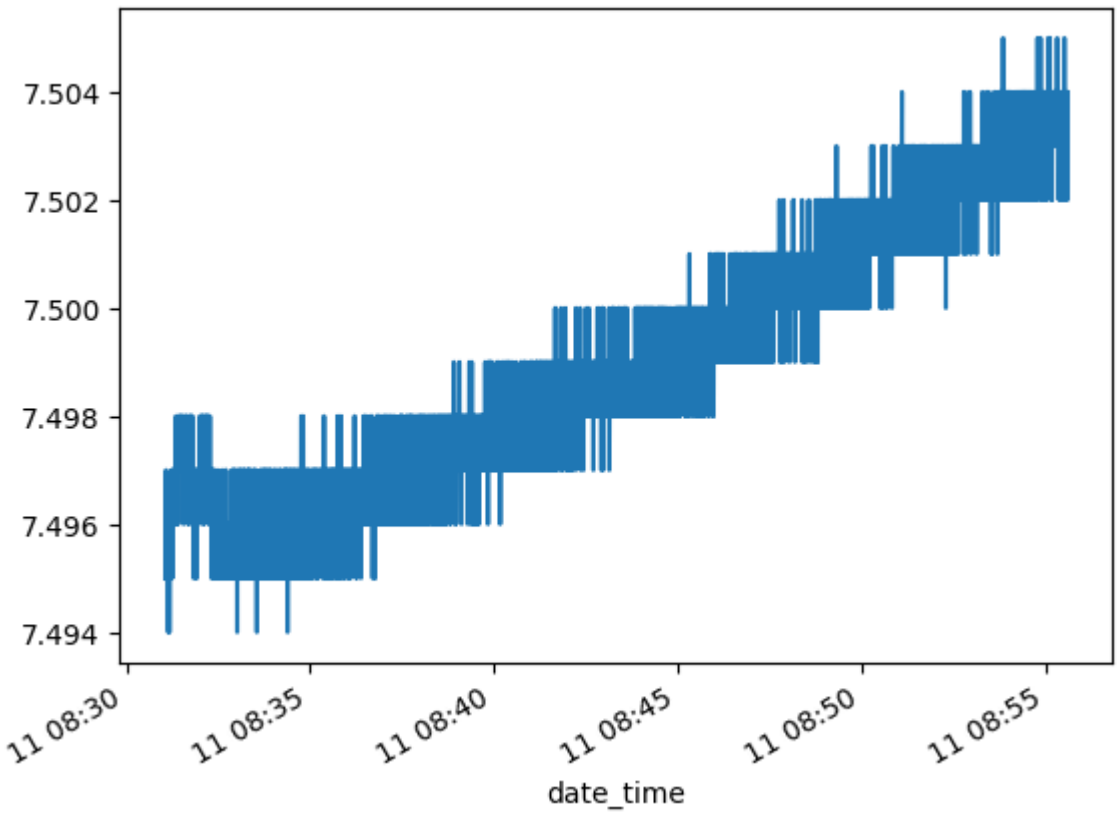
9982 rows × 8 columns

[4]:

```
df['pH'].plot()
```

[4]:

```
<Axes: xlabel='date_time'>
```




```
[5]:
```

```
m
```

```
[5]:
```

```
{'experiment_name': ' comment',
 'experiment_description': '\n',
 'software_version': 'Workbench V1.5.3.2466',
 'device': 'FSP19 [A] FSPRO-4',
 'device_serial': '21450119',
 'uid': '2466C2055D64A687',
 'firmware': '4.11:001',
 'channel': 1,
 'sensor_code': 'SHG7-597-623',
 'settings': {'duration': '16 ms',
  'intensity': '80%',
  'amp': '400x',
  'frequency': 3000,
```

```
  'crc_enable': False,
  'write_lock': False,
  'auto_flash_duration': False,
  'auto_amp': True,
  'analyte': 'pH',
  'fiber_type': '1 mm',
  'temperature': 'external sensor',
  'pressure': 'internal sensor',
  'salinity': 7.5,
  'fiber_length_mm': 1000},
 'calibration': {'date_calibration_acid': datetime.datetime(2024, 12,
9, 0, 0),
  'date_calibration_base': None,
  'date_calibration_offset': None,
  'R1': 1.420695,
  'pH1': 2.266,
  'temp1': 24.89,
  'salinity1': 2.0,
  'R2': 0.046,
  'pH2': 14.0,
  'temp2': 20.0,
  'salinity2': 7.5,
  'offset': 0.0,
  'dphi_ref': 57.8,
  'attenuation_coefficient': 0.0339,
  'bkgdAmpl': 0.584,
  'bkgdDphi': 0.0,
  'dsf_dye': 0.9047,
  'dtf_dye': -0.00567,
  'pka': 8.319,
  'slope': 1.087,
  'bottom_t': -0.0159,
  'top_t': -0.002465,
  'slope_t': 0.0,
  'pka_t': -0.01147,
  'pka_is1': 2.54,
  'pka_is2': 0.25}}
```

```
[6]:
```

```
m['settings']
```

```
[6]:
```

```
{'duration': '16 ms',
 'intensity': '80%',
```

```
 'amp': '400x',
 'frequency': 3000,
 'crc_enable': False,
 'write_lock': False,
 'auto_flash_duration': False,
 'auto_amp': True,
 'analyte': 'pH',
 'fiber_type': '1 mm',
 'temperature': 'external sensor',
 'pressure': 'internal sensor',
 'salinity': 7.5,
 'fiber_length_mm': 1000}
```

```
[7]:
```

```
m['calibration']
```

```
[7]:
```

```
{'date_calibration_acid': datetime.datetime(2024, 12, 9, 0, 0),
 'date_calibration_base': None,
 'date_calibration_offset': None,
 'R1': 1.420695,
 'pH1': 2.266,
 'temp1': 24.89,
 'salinity1': 2.0,
 'R2': 0.046,
 'pH2': 14.0,
 'temp2': 20.0,
 'salinity2': 7.5,
 'offset': 0.0,
 'dphi_ref': 57.8,
 'attenuation_coefficient': 0.0339,
 'bkgdAmpl': 0.584,
 'bkgdDphi': 0.0,
 'dsf_dye': 0.9047,
 'dtf_dye': -0.00567,
 'pka': 8.319,
 'slope': 1.087,
 'bottom_t': -0.0159,
 'top_t': -0.002465,
 'slope_t': 0.0,
 'pka_t': -0.01147,
 'pka_is1': 2.54,
 'pka_is2': 0.25}
```

FirePlate

[8]:

```
df, m = parse('ChannelData FirePlate/A_FirePlate-O2_(A Ch.
1)_Oxygen.txt')
```

[9]:

```
df
```

[9]:

date_time	time_s	A02_oxygen_%O2	A02_dphi	A02_signal_intensity	A02_ambient_light	A02_status	A03_ox
2024-09-04 09:26:12.371							
2024-09-04 09:26:15.380							
2024-09-04 09:26:18.396							
2024-09-04 09:26:21.404							
2024-09-04 09:26:24.419							
2024-09-04 09:26:27.438							
2024-09-04 09:26:30.424							
2024-09-04 09:26:33.441							
2024-09-04 09:26:36.448							
2024-09-04 09:26:39.456							

	time_s	A02_oxygen_%O2	A02_dphi	A02_signal_intensity	A02_ambient_light	A02_status	A03_ox
date_time							
2024-09-04 09:26:42.464							
2024-09-04 09:26:45.477							
2024-09-04 09:26:48.484							
2024-09-04 09:26:51.493							
2024-09-04 09:26:54.513							
2024-09-04 09:26:57.507							

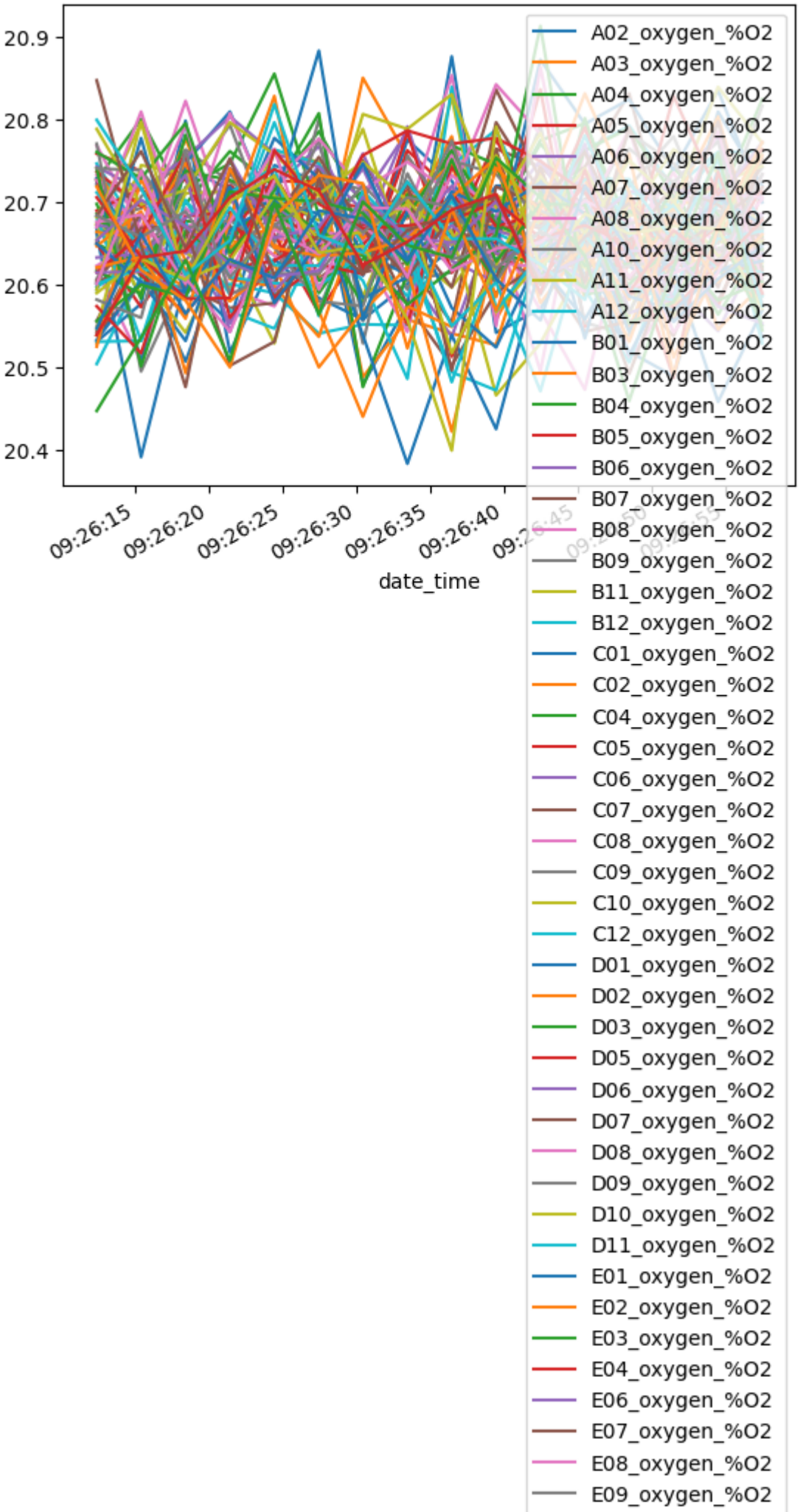
16 rows × 423 columns

```
[10]:
```

```
df.filter(regex='%O2').plot()
```

```
[10]:
```

```
<Axes: xlabel='date_time'>
```


20.9
20.8
20.7
20.6
20.5
20.4
09:26:15
09:26:20
09:26:25
09:26:30
09:26:35
09:26:40
date_time
A02_oxygen_%O2
A03_oxygen_%O2
A04_oxygen_%O2
A05_oxygen_%O2
A06_oxygen_%O2
A07_oxygen_%O2
A08_oxygen_%O2
A10_oxygen_%O2
A11_oxygen_%O2
A12_oxygen_%O2
B01_oxygen_%O2
B03_oxygen_%O2
B04_oxygen_%O2
B05_oxygen_%O2
B06_oxygen_%O2
B07_oxygen_%O2
B08_oxygen_%O2
B09_oxygen_%O2
B11_oxygen_%O2
B12_oxygen_%O2
C01_oxygen_%O2
C02_oxygen_%O2
C04_oxygen_%O2
C05_oxygen_%O2
C06_oxygen_%O2
C07_oxygen_%O2
C08_oxygen_%O2
C09_oxygen_%O2
C10_oxygen_%O2
C12_oxygen_%O2
D01_oxygen_%O2
D02_oxygen_%O2
D03_oxygen_%O2
D05_oxygen_%O2
D06_oxygen_%O2
D07_oxygen_%O2
D08_oxygen_%O2
D09_oxygen_%O2
D10_oxygen_%O2
D11_oxygen_%O2
E01_oxygen_%O2
E02_oxygen_%O2
E03_oxygen_%O2
E04_oxygen_%O2
E06_oxygen_%O2
E07_oxygen_%O2
E08_oxygen_%O2
E09_oxygen_%O2

[11]:

```
m['settings']
```

[11]:

```
{'duration': '1 ms',
 'intensity': '40%',
 'amp': '400x',
 'frequency': 4000,
 'crc_enable': False,
 'write_lock': False,
 'auto_flash_duration': True,
 'auto_amp': True,
 'analyte': 'oxygen',
 'fiber_type': '1 mm',
 'temperature': 'internal sensor',
 'pressure': 'internal sensor',
 'salinity': 7.5,
 'fiber_length_mm': 0}
```

[12]:

```
import pandas as pd
pd.DataFrame(m['calibration'])
```

[12]:

	A02	A03	A04	A05	A06	A07	A08	A10	A11	A12	...	H02	H03	H04	H05	H06	H07	H09	H10	H11
date_calibration_high																				
date_calibration_zero																				
dphi100																				
dphi0																				
f																				
m																				
freq																				
tt																				
kt																				

	A02	A03	A04	A05	A06	A07	A08	A10	A11	A12	...	H02	H03	H04	H05	H06	H07	H09	H10	H11
bkgdAmpl																				
bkgdDphi																				
mt																				
pressure																				
temp100																				
humidity																				
temp0																				
percentO2																				

17 rows × 84 columns

```
[ ]:
```

Oxygen Calculation Tools

A collection of functions to re-calculate results from measurements.

Function to calculate oxygen partial pressure from raw data

The following functions can be used to calculate oxygen partial pressures from the phase angle dphi. All other oxygen units can be calculated from the partial pressure.

Module containing functions for oxygen unit conversions.

pyrotoolbox.oxygen. calculate_pO2 (*dphi* , *temperature* , *dphi0* , *dphi100* , *temp0* , *temp100* , *pressure* , *humidity* , *percentO2* , *f* , *m* , *kt* , *tt* , *mt* , *ft = 0* , *freq = 4000* , *** kwargs*)

Calculate pO2 in hPa for a given phase-angle and temperature.

Parameters :

- **dphi** – phase angle in °
- **temperature** – in °C
- **dphi0** – phase angle at zero calibration point in °
- **dphi100** – phase angle at the upper calibration point in !
- **temp0** – temperature at zero calibration point in °C
- **temp100** – temperature at the upper calibration point in °C
- **pressure** – pressure at the upper calibration point in mbar/hPa
- **humidity** – relative humidity at the upper calibration point in %
- **percentO2** – oxygen content in the dry calibration gas (upper calibration point) in %
- **f** – sensor constant

- **m** – sensor constant
- **kt** – sensor constant (in 1/K)
- **tt** – sensor constant (in 1/K)
- **mt** – sensor constant (in 1/K)
- **ft** – sensor constant (in 1/K)
- **freq** – modulation frequency in Hz
- **kwargs** – to accept additional unused parameters. (allow passing **calibration without an error)

Returns :

pO2 in hPa

pyrotoolbox.oxygen. calculate_pO2_from_calibration (*dphi* , *temperature* , *calibration : dict*)

Calculate pO2 in hPa for a given dphi, temperature and calibration.

Params dphi :

phase angle in °

Parameters :

- **temperature** – in °C
- **calibration** – dictionary as generated by the parser functions (metadata['calibration']).

Expected entries:

- dphi0 [°]
- dphi100 [°]
- temp0 [°C]
- temp100 [°C]
- pressure [mbar]
- humidity [%]
- percentO2 [%]

- f
- m
- kt [1/K]
- tt [1/K]
- mt [1/K]
- ft [1/K]

Functions for unit conversions

The following functions are used to convert between oxygen units. The partial pressure of oxygen (in hPa) is the central unit. All units can be converted toward pO2 and all units can be calculated from pO2. It is also the measured unit on all devices.

Module containing functions for oxygen unit conversions.

pyrotoolbox.oxygen. **i_only_think_in_hpa** (*df : DataFrame* , *m : dict*) → Series

Convert any oxygen data to hPa.

Useful for people like me who prefer to look at oxygen data in hPa. This function takes the dataframe and metadata dict of the parser functions and returns the oxygen data in hPa. If you want to change the conditions of the conversions (e.g. assuming the experiment setup was wrong) use the convert_to_hPa function.

Parameters :

- **df** – DataFrame as generated by the parser functions.
- **m** – metadata-dict as generated by the parser functions.

Returns :

Series with oxygen data in hPa.

pyrotoolbox.oxygen. **i_have_a_fireplate_and_still_only_think_in_hPa** (*df : DataFrame* , *m : dict*) → DataFrame

A copy of the popular “i_only_think_in_hpa” function for fireplate data.

This was separated due to different format of fireplate data, but works very similar.

Parameters :

- **df** – DataFrame as generated by the read_fireplate_workbench function.
- **m** – metadata-dict as generated by the read_fireplate_workbench function.

Returns :

DataFrame with oxygen data in hPa.

pyrotoolbox.oxygen. convert_to_hPa (*data : Series* , *unit : str* , *temperature = None* , *pressure = None* , *salinity = None*) → Series

Convert any oxygen unit to hPa.

Parameters :

- **data** – any oxygen data produced by a pyroscience device.
- **unit** – unit of the data. Has to be one of ('oxygen_hPa', 'oxygen_torr', 'oxygen_%O2', 'oxygen_%airsat', 'oxygen_µM', 'oxygen_µg/L', 'oxygen_mg/L', 'oxygen_mL/L').
- **temperature** – sample temperature. Used for all units except 'oxygen_hPa', 'oxygen_torr' and 'oxygen_%O2'.
- **pressure** – gas-pressure of or above the sample. Used for all units except 'oxygen_hPa' and 'oxygen_torr'
- **salinity** – salinity of the sample in g/L. Used for all dissolved oxygen units.

pyrotoolbox.oxygen. hPa_to_torr (*data*)

Convert hPa to torr.

Parameters :

data – hPa data to convert.

pyrotoolbox.oxygen. hPa_to_percentO2 (*data* , *pressure*)

Convert hPa to %O2.

Parameters :

- **data** – hPa data to convert.
- **pressure** – gas pressure of the sample in hPa.

pyrotoolbox.oxygen. hPa_to_percent_airsat (*data* , *pressure* , *temperature*)

Convert hPa to %airsat. This is a unit for water samples and assumes 100% relative humidity.

Parameters :

- **data** – hPa data to convert.
- **pressure** – gas pressure above the sample in hPa.
- **temperature** – temperature of the sample in °C.

pyrotoolbox.oxygen. hPa_to_uM (*data* , *temperature* , *salinity*)

Convert hPa to µM. This is a unit for water sample.

Parameters :

- **data** – hPa data to convert.
- **temperature** – temperature of the sample in °C.
- **salinity** – salinity of the sample in g/L.

pyrotoolbox.oxygen. hPa_to_mgL (*data* , *temperature* , *salinity*)

Convert hPa to mg/L. This is a unit for water sample.

Parameters :

- **data** – hPa data to convert.
- **temperature** – temperature of the sample in °C.
- **salinity** – salinity of the sample in g/L.

Helper functions for oxygen calculations

The following functions are implemented do to the above documented calculations. They might still be useful for your own unit conversions

Module containing functions for oxygen unit conversions.

pyrotoolbox.oxygen. vapour_pressure_water (*temperature*)

Calculates the vapour pressure of water at given Temperature (°C), return in hPa

#This is Equation (6) from Murray, F.W. 1967. On the computation of saturation vapour pressure. J. Applied Meteorology 6: 203-204

Parameters :

temperature – Temperature in °C

Returns :

vapour pressure in hPa

pyrotoolbox.oxygen. calc_oxygen_solubility (*temperature* , *salinity*)

Calculate the oxygen solubility in µM for water with given temperature and salinity.

The formula and constants from “From Garcia 1992: Oxygen solubility in seawater: better fitting equations” are used.

Parameters :

- **temperature** – water temperature (°C)
- **salinity** – salinity of water (g/L)

Returns :

oxygen solubility in µM

pyrotoolbox.oxygen. calc_pressure_and_water_corrected_pO2 (*pressure* , *T* , *water_sat* , *percentO2 = 20.95*)

calculate the partial pressure of oxygen in air with a given water saturation and total pressure

Parameters :

- **pressure** – ambient pressure in hPa
- **T** – Temperature in °C
- **water_sat** – water saturation of the air in % (0-100)
- **percentO2** – percentage of oxygen in air in % (0-100). Default 20.95

Returns :

partial pressure of oxygen in hPa

Examples

Load AquapHOx-L data and recalculate dissolved oxygen with different salinity

[ ]:

```
from pyrotoolbox import parse
%matplotlib inline
from pyrotoolbox.oxygen import hPa_to_mgL
```

[3]:

```
# Load AquapHOx-L data
df, m = parse('AphoxData/21270013.txt')
```

[8]:

```
# Convert hPa to mg/L for a salinity of 32 g/L
```

[9]:

```
hPa_to_mgL(df['oxygen_hPa'], df['sample_temperature'], 32)
```

[9]:

```
date_time
2024-07-04 11:21:14    7.796528
2024-07-04 11:21:44    7.804866
2024-07-04 11:22:14    7.803978
2024-07-04 11:22:44    7.860132
2024-07-04 11:23:14    7.687174
                           ...
2024-07-04 19:33:44    7.388505
2024-07-04 19:34:14    7.383316
2024-07-04 19:34:44    7.377246
2024-07-04 19:35:14    7.366197
2024-07-04 19:35:44    7.359155
Length: 990, dtype: float64
```

Oxygen unit conversions

[1]:

```
from pyrotoolbox.parsers import parse
from pyrotoolbox.oxygen import *
```

Load data

[2]:

```
df, m = parse('datademo4/A_Firesting Pro (4 Channels)_(A Ch.
1)_Oxygen.txt')
```

[3]:

```
df = df.iloc[80:].copy()
```

convert input unit (%O2) to hPa

```
[4]:
```

```
df['oxygen_hPa'] = i_only_think_in_hpa(df, m)
```

alternative:

```
[5]:
```

```
df['oxygen_hPa_2'] = convert_to_hPa(df['oxygen_%O2'],
unit='oxygen_%O2',
                                    temperature=m['settings']
['temperature'],
                                    pressure=df['pressure'])
```

convert hPa to all other oxygen units

```
[6]:
```

```
df['oxygen_torr'] = hPa_to_torr(df['oxygen_hPa'])
```

```
[7]:
```

```
df['oxygen_%O2'] = hPa_to_percentO2(df['oxygen_hPa'], df['pressure'])
```

```
[8]:
```

```
df['oxygen_%airsat'] = hPa_to_percent_airsat(df['oxygen_hPa'],
df['pressure'], temperature=20)
```

[9]:

```
df['oxygen_mgL'] = hPa_to_mgL(df['oxygen_hPa'], temperature=20,
salinity=7.5)
```

[10]:

```
df['oxygen_uM'] = hPa_to_uM(df['oxygen_hPa'], temperature=20,
salinity=7.5)
```

unit conversion mg/L for different environmental conditions

Example: user has logged data with a fixed-T setting of 37°C and salinity of 7.5g/L, but the real temperature was 32°C and real salinity was 11g/L.

[11]:

```
po2 = i_only_think_in_hpa(df, m)
hPa_to_mgL(po2, temperature=32, salinity=11)
```

[11]:

```
date_time
2023-04-03 13:16:28.485    0.247168
2023-04-03 13:16:29.486    1.704633
2023-04-03 13:16:30.486    2.777244
2023-04-03 13:16:31.486    3.508465
2023-04-03 13:16:32.486    3.947728
2023-04-03 13:16:33.485    4.229732
2023-04-03 13:16:34.485    4.379028
2023-04-03 13:16:35.485    4.378365
2023-04-03 13:16:36.485    4.379360
2023-04-03 13:16:37.485    4.419836
2023-04-03 13:16:38.485    4.491167
2023-04-03 13:16:39.486    4.563492
2023-04-03 13:16:40.487    4.635486
Name: oxygen_hPa, dtype: float64
```

```
[ ]:
```

```

```

pH Calculation Tools

Module to calculate pH for devices from Pyroscience.

The following functions are only valid for devices with Firmware >= 4.10 (released 2023).

pyrotoolbox.pH. calculate_pH (*R* , *temperature* , *salinity* , *top* , *bottom* , *pka* , *slope* , *pka_t* , *bottom_t* , *top_t* , *slope_t* , *pka_is1* , *pka_is2* , *offset* , *** kwargs*)

Calculate pH from R, temperature, salinity and sensor constants. See also function calculate_pH_from_calibration.

Parameters :

- **R** – R-value from the sensor
- **temperature** – temperature of the sample in °C
- **salinity** – salinity of the sample in g/L
- **top** – material constant
- **bottom** – material constant
- **pka** – material constant
- **slope** – material constant
- **pka_t** – material constant
- **bottom_t** – material constant
- **top_t** – material constant
- **slope_t** – material constant
- **pka_is1** – material constant
- **pka_is2** – material constant
- **offset** – offset in pH-units (from 3rd calibration point)
- **kwargs** – kwargs are ignored

Returns :

pyrotoolbox.pH. calculate_pH_from_calibration (*R* , *temperature* , *salinity* , *calibration : dict* , ** *kwargs*)

apply a pH calibration to measurement data for FW >= 410

The calibration parameters can be passed from the calibration metadata.

Example usage: apply_pH_calibration(data['R'], data['temp'], data['salinity'], m['calibration'])

Parameters :

- **R** – R-value from the sensor
- **temperature** – temperature of the sample in °C

- **salinity** – salinity of the sample in g/L
- **calibration** – calibration dictionary as created by the parsers
- **kwargs** – kwargs are inserted into the calibration and override the values

Returns :

calculated pH values

pyrotoolbox.pH. calculate_pH_from_interpolated_calibration (*R* , *temperature* , *salinity* , *calibrations* , *return_fits = False*)

Apply an interpolated pH calibration to measurement data

The top and bottom value is calculated for every passed calibration and fitted linear over time. For every measurement point an individual value of top and bottom is calculated.

Parameters :

- **R** – data for “R”
- **temperature** – overrides temp values from df. Single value or iterable with same length as df is required
- **salinity** – Single value or iterable with same length as df is required
- **calibrations** – dict in format {timestring: calib_data} e.g. {‘2019-05-05 13:10:00’: {pH1: 4, temp1: 22.08, ...}}, or a list of calibration-dicts
- **R** – overrides R values from df. Single value or iterable with same length as df is required
- **return_fits** – return the fits of top, bottom and offset instead

Returns :

calculated pH-values

pyrotoolbox.pH. calculate_pH_with_prospective_drift_compensation (*R* , *temperature* , *salinity* , *calibration : dict* , *d0 : float* , *d1 : float* , *d2 : float* , *d3 : float* , *start_timestamp = None*)

apply prospective pH drift compensation to a pH measurement.

The effect on the sensors is assumed to be only a decrease in Top.

Parameters :

- **R** – R-values, pandas Series or DataFrame with time index
- **temperature** – temperature data
- **salinity** – salinity data
- **calibration** – calibration dictionary as created by the parser functions
- **start_timestamp** – optional, time of the calibration. By default, the date_calibration_acid value is used
- **d0** – first coefficient of the temperature dependent drift
- **d1** – second coefficient of the temperature dependent drift
- **d2** – third coefficient of the temperature dependent drift
- **d3** – fourth coefficient of the temperature dependent drift
- **start_timestamp** – optional, start of the drift. If omitted the date of the last acid calibration is used

Returns :

array of pH values

Examples

```
[1]:
```

```
import matplotlib.pyplot as plt
%matplotlib inline

from pyrotoolbox.parsers import parse
from pyrotoolbox.pH import (calculate_pH,
calculate_pH_from_calibration,
```

```
calculate_pH_from_interpolated_calibration,

calculate_pH_with_prospective_drift_compensation)
```

recalc with different calibration

```
[2]:
```

```
pH_hl_1, m_pH_hl_1 = parse('rovdata/PH2ROV1.txt')
pH_hl_2, m_pH_hl_2 = parse('rovdata/PH2ROV2.txt')
```

```
[3]:
```

```
pH_hl_1
```

```
[3]:
```

date_time	status	dphi	sample_temperature	case_temperature	signal_intensity	ambient_light	pressure	hu
2024-04-08 13:55:32								
2024-04-08 13:56:32								
2024-04-08 13:57:32								
2024-04-08 13:58:32								
2024-04-08 13:59:32								
...								
2024-07-09 17:25:32								
2024-07-09 17:26:32								
2024-07-09 17:27:32								

	status	dphi	sample_temperature	case_temperature	signal_intensity	ambient_light	pressure	hu
date_time								
2024-07-09 17:28:32								
2024-07-09 17:29:32								

132695 rows × 11 columns

[4]:

```
calculate_pH_from_calibration?
```

```
Signature:
calculate_pH_from_calibration(
    R,
    temperature,
    salinity,
    calibration: dict,
    **kwargs,
)
Docstring:
apply a pH calibration to measurement data for FW >= 410

The calibration parameters can be passed from the calibration
metadata.

Example usage:
apply_pH_calibration(data['R'], data['temp'], data['salinity'],
m['calibration'])

:param R: R-value from the sensor
:param temperature: temperature of the sample in °C
:param salinity: salinity of the sample in g/L
:param calibration: calibration dictionary as created by the parsers
:param kwargs: kwargs are inserted into the calibration and override
the values
:return: calculated pH values
File:      ~/Programmieren/pyrotoolbox/pyrotoolbox/pH.py
Type:      function
```

[5]:

```
pH_hl_1['pH_endcal'] = calculate_pH_from_calibration(pH_hl_1['R'],

pH_hl_1['sample_temperature'],

m_pH_hl_1['settings']['salinity'],

m_pH_hl_2['calibration'])
```

```
/home/christoph/.venvs/pyro/lib/python3.12/site-packages/pandas/core/
arraylike.py:399: RuntimeWarning: invalid value encountered in log10
  result = getattr(ufunc, method)(*inputs, **kwargs)
```

[6]:

```
pH_hl_1
```

[6]:

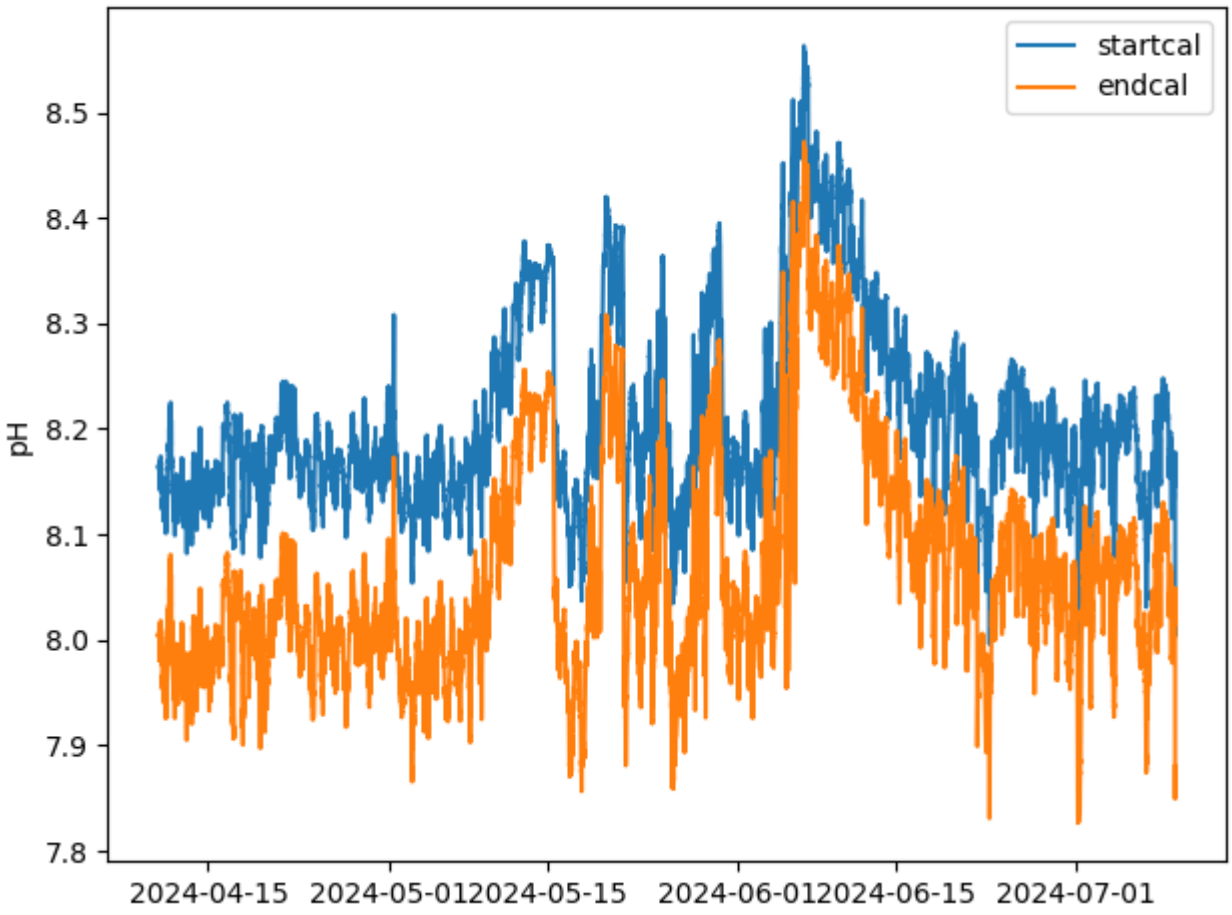
date_time	status	dphi	sample_temperature	case_temperature	signal_intensity	ambient_light	pressure	hu
2024-04-08 13:55:32								
2024-04-08 13:56:32								
2024-04-08 13:57:32								
2024-04-08 13:58:32								
2024-04-08 13:59:32								
...								
2024-07-09 17:25:32								
2024-07-09 17:26:32								
2024-07-09 17:27:32								

	status	dphi	sample_temperature	case_temperature	signal_intensity	ambient_light	pressure	hu
date_time								
2024-07-09 17:28:32								
2024-07-09 17:29:32								

132695 rows × 12 columns

[7]:

```
fig, ax = plt.subplots()
ax.plot(pH_hl_1['pH']['2024-04-10 15:00':], label='startcal')
ax.plot(pH_hl_1['pH_endcal']['2024-04-10 15:00':], label='endcal')
ax.legend()
ax.set_ylabel('pH')
fig.tight_layout()
```



calculate with different temperature

[8]:

```
pH_hl_1['pH_fixedT25'] = calculate_pH_from_calibration(pH_hl_1['R'],
                                                       25,

m_pH_hl_1['settings']['salinity'],

m_pH_hl_1['calibration'])
```

calculate with device temperature

[9]:

```
pH_hl_1['pH_devT'] = calculate_pH_from_calibration(pH_hl_1['R'],

pH_hl_1['case_temperature'],

m_pH_hl_1['settings']['salinity'],

m_pH_hl_1['calibration'])
```

```
/home/christoph/.venvs/pyro/lib/python3.12/site-packages/pandas/core/
arraylike.py:399: RuntimeWarning: invalid value encountered in log10
  result = getattr(ufunc, method)(*inputs, **kwargs)
```

calculate with 40 PSU

[10]:

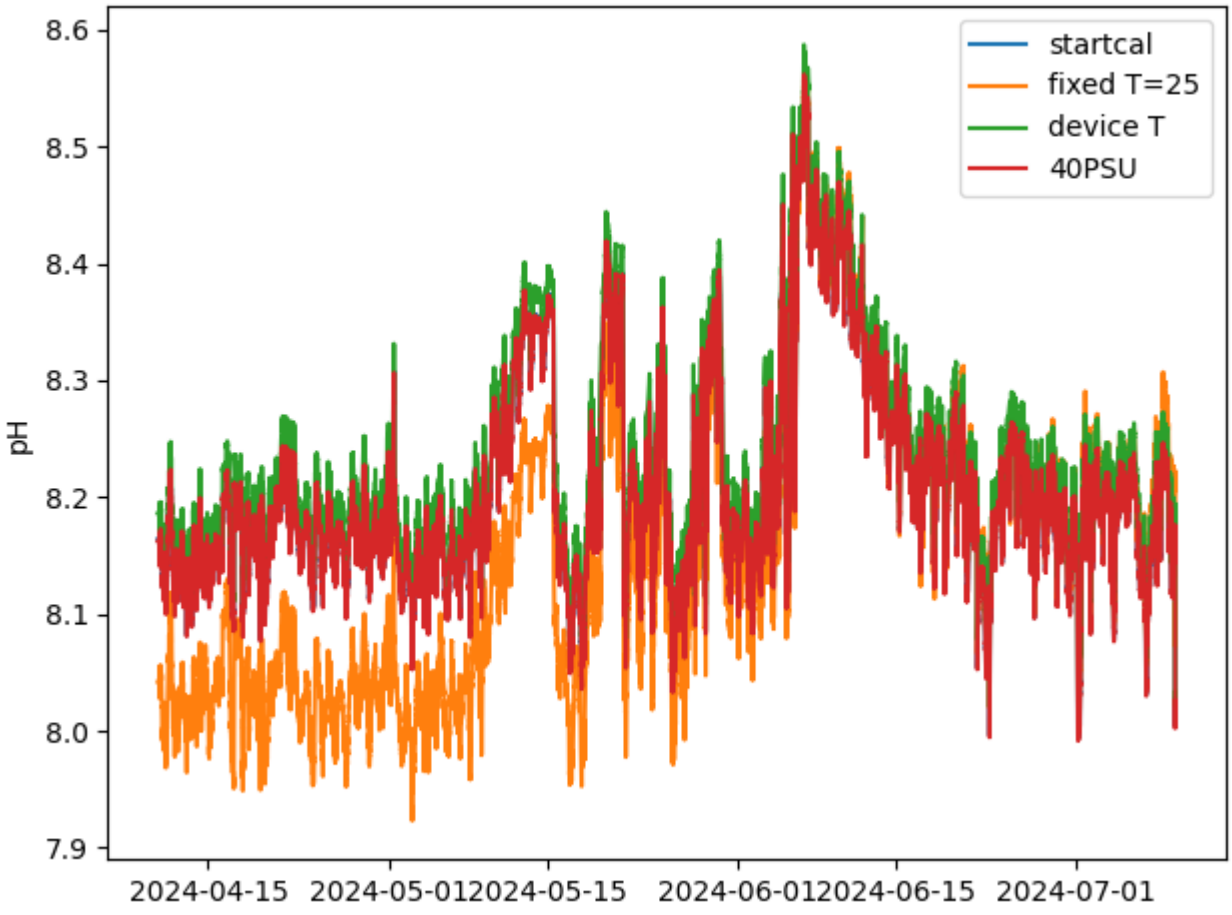
```
pH_hl_1['pH_PSU40'] = calculate_pH_from_calibration(pH_hl_1['R'],

pH_hl_1['sample_temperature'],
                                                    40,
```

```
m_pH_hl_1['calibration'])
```

```
/home/christoph/.venvs/pyro/lib/python3.12/site-packages/pandas/core/
arraylike.py:399: RuntimeWarning: invalid value encountered in log10
  result = getattr(ufunc, method)(*inputs, **kwargs)
```

```
[11]:
```

```
fig, ax = plt.subplots()
ax.plot(pH_hl_1['pH']['2024-04-10 15:00':], label='startcal')
ax.plot(pH_hl_1['pH_fixedT25']['2024-04-10 15:00':], label='fixed
T=25')
ax.plot(pH_hl_1['pH_devT']['2024-04-10 15:00':], label='device T')
ax.plot(pH_hl_1['pH_PSU40']['2024-04-10 15:00':], label='40PSU')
ax.legend()
ax.set_ylabel('pH')
fig.tight_layout()
```



Interpoliertat pH drift correction

[12]:

```
pH_hl_1['pH_interpol_corr'] =
calculate_pH_from_interpolated_calibration(
                                                pH_hl_1['R'],

pH_hl_1['sample_temperature'],
                                                35,
                                                {'2024-04-08 13:35':
m_pH_hl_1['calibration'],
                                                 '2024-07-09 13:00':
m_pH_hl_2['calibration']})
```

```
/home/christoph/.venvs/pyro/lib/python3.12/site-packages/pandas/core/
arraylike.py:399: RuntimeWarning: invalid value encountered in log10
  result = getattr(ufunc, method)(*inputs, **kwargs)
```

Prospective pH drift correction

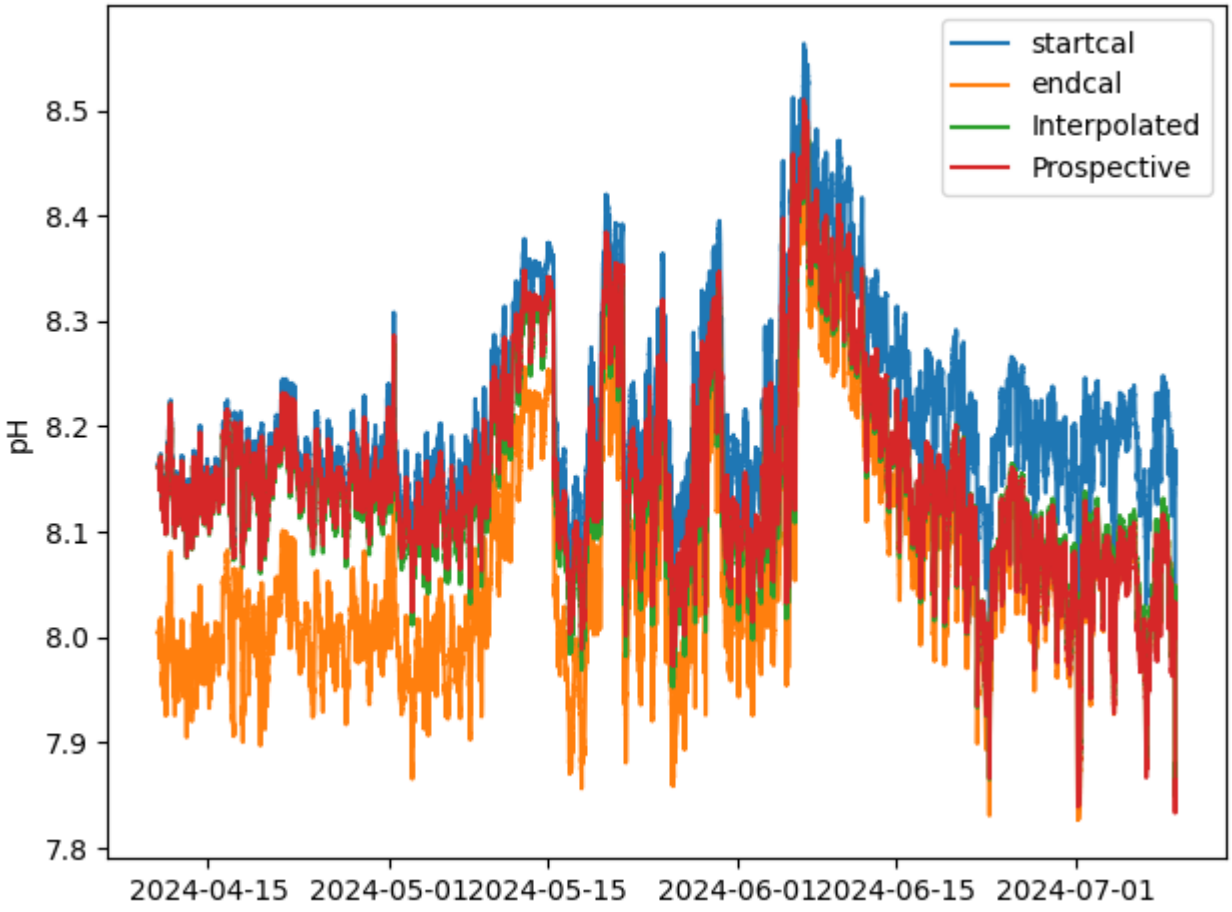
[13]:

```
pH_hl_1['pH_corr_prosp'] =
calculate_pH_with_prospective_drift_compensation(
                                                pH_hl_1['R'],

pH_hl_1['sample_temperature'],
                                                35,

m_pH_hl_1['calibration'], 4.3652*10**10, 9.0761*10**3,0,0)
```

```
/home/christoph/.venvs/pyro/lib/python3.12/site-packages/pandas/core/
arraylike.py:399: RuntimeWarning: invalid value encountered in log10
  result = getattr(ufunc, method)(*inputs, **kwargs)
```

[14]:

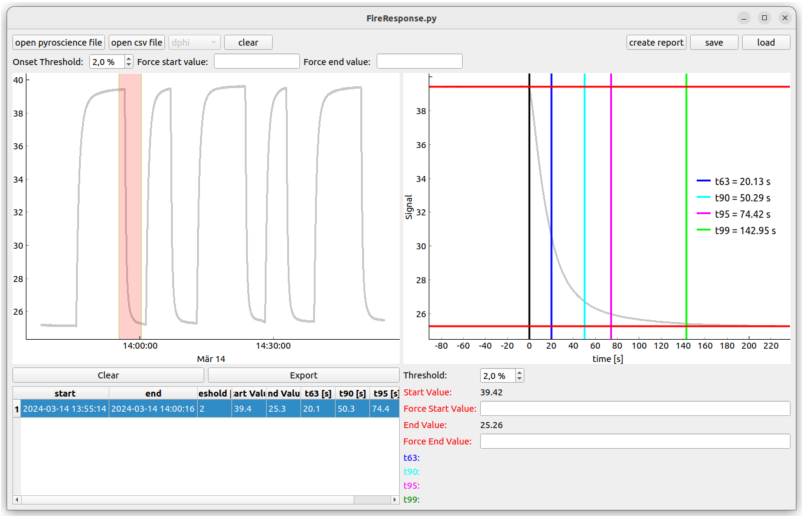
```
fig, ax = plt.subplots()
ax.plot(pH_hl_1['pH']['2024-04-10 15:00':], label='startcal')
ax.plot(pH_hl_1['pH_endcal']['2024-04-10 15:00':], label='endcal')
ax.plot(pH_hl_1['pH_interpol_corr']['2024-04-10 15:00':],
label='Interpolated')
ax.plot(pH_hl_1['pH_corr_prosp']['2024-04-10 15:00':],
label='Prospective')
ax.legend()
ax.set_ylabel('pH')
fig.tight_layout()
```




```
[ ]:
```

FireResponse

FireResponse is a graphical tool to extract response times from measurements. It can be started by running “FireResponse” in a terminal.



PyroHtmlReporter

PyroHtmlReporter is still in active development. Anything might change!

A tool to generate html reports of measurement data. The report contains a summary of the measurement, the settings and calibration registers and an interactive plot of the most important data. The reports have the filename of the input file with an additional "_report.html". If more than one file is passed at once the data is combined into a "summary.html" file.

This tool is intended to:

- create short shareable reports of measurements
- to have a quick look on measurement data
- quick debugging

The reports are standalone html files and can be viewed in any browser.

Example usage

```
PyroHtmlReporter my_measurement_data.txt
```

This results in a file "my_measurement_data_report.html".

```
PyroHtmlReporter my_measurement_data1.txt my_measurement_data2.txt
```

This results in the files "my_measurement_data1_report.html", "my_measurement_data2_report.html" and "summary.html".

Indices and tables

- Index
- Module Index
- Search Page