flux-data-qaqc Documentation

John Volk

Oct 15, 2021

Contents:

View on GitHub.

`flux-data-qaqc` provides a framework to create reproducible workflows for validation and analysis of eddy covariance data. The package is intended for those who need to post-process flux data, particularly for generating daily and monthly evapotransipration (ET) timeseries estimates with energy balance closure corrections applied. Applications where this software may be useful include analysis involving eddy covariance flux tower data, hydrologic or atmospehric model validation, and irrigation and water consumption studies.

Key functionalities and tools include:

- data validation with methods for quality-based filtering
- time series data tools, e.g. temporal aggregation and resampling
- management of site metadata, data provenance, and file structure
- energy balance closure algorithms and other meterological calculations
- downloading and management of gridMET meterological data
- customizable and interactive data visualizations
- batch processing and unit conversions

CHAPTER 1

Installation

Using PIP:

```
pip install fluxdataqaqc
```

PIP should install the necessary dependencies however it is recommended to use conda and first install the provided virtual environment. This is useful to avoid changing your local Python environment. Note, `flux-data-qaqc` has been tested for Python 3.7+, although it may work with versions greater than or equal to 3.4.

First make sure you have the `fluxdataqaqc` environment file, you can download it here. Next to install run,

```
conda env create -f environment.yml
```

To activate the environment before using the `flux-data-qaqc` package run,

```
conda activate fluxdataqaqc
```

Now install using PIP:

```
pip install fluxdataqaqc
```

Now all package modules and tools should be available in your Python environment PATH and able to be imported. Note if you did not install the Conda virtual environment above, PIP should install dependencies automatically but be sure to be using a version of Python above or equal to 3.4. To test that everything has installed correctly by opening a Python interpretor or IDE and run the following:

```python
import fluxdataqaqc
```

and

```python
from fluxdataqaqc import Data, QaQc, Plot
```

If everything has been installed correctly you should get no errors.

1.1 Developer mode

If you plan on contributing to `flux-data-qaqc` you can install it in *developer mode* which allows you to easily modify the source code and test changes within a Python environment.

Clone or download from GitHub.

```
git clone https://github.com/Open-ET/flux-data-qaqc.git
```

Next move to the root directory and install and activate the conda environment:

```
conda env create -f environment.yml
```

Run the following to install `flux-data-qaqc` in developer mode into your environment,

```
pip install -e .
```

CHAPTER 2

Configuration Options and Caveats

This tutorial shows how to use `flux-data-qaqc` with climate data of various formats and generally covers formatting rules of input data and extra options that can be set in a config file. The major differences when using `flux-data-qaqc` for different input data lie in the config file declarations therefore the entire workflow from *Tutorial* will work just the same once your config file is set up correctly.

2.1 Example data description

The data used in *Configuration Options and Caveats* is provided with `flux-data-qaqc` and can be downloaded here. There are two datasets used in the following examples, the data used for showing quality based filtering of data based on QC flags is from the FLUXNET 2015 dataset for site "ARM USDA UNL OSU Woodward Switchgrass 1" which contains switchgrass, more information on this site can be found here. The site that contains multiple soil heat flux measurements for weighted averaging and soil moisture measurements for plotting is a subset (shortened for reduced disk space) of the "ARM Southern Great Plains site- Lamont" AmeriFlux site dataset, more information on this site can be found here.

A reproducible Jupyter Notebook with minor differences of this tutorial can be found here.

2.2 Setting up a config file

`flux-data-qaqc` starts with the creation of a configuration file, a text file with extension ".ini" or ".INI" that follows the rules set here. A config file for `flux-data-qaqc` requires the sections: 1. **METADATA** 2. **DATA** although you may provide additional for custom uses.

In **METADATA** you may enter any metadata that you wish so long as the entrie's *key* is followed by an equal sign and the assigned *value*, i.e.

```
key = value
```

Here are the *mandatory* metadata entries unique to `flux-data-qaqc`:

- climate_file_path

- station_elevation
- station_longitude
- station_latitude
- site_id

The "climate_file_path" is the full or relative file path of the input climate file (excel or CSV, more on formatting this below) containing the climatic data to be analyzed. The "station_elevation" (meters) and latitude/longitude (decimal degrees) fields are used to calculate clear sky potential solar radiation using an ASCE formulation and to locate the nearest gridMET cell centroid when downloading reference ET. "site_id" is used for saving output files.

Optional metadata entries that are used by `flux-data-qaqc` include "missing_data_value", "qc_threshold", "qc_flag", "var_name_delim", "skiprows", "date_parser", and "gridmet_file_path". "missing_data_value" is used to correctly parse missing values in the input climate time series. All other optional metadata that can be used by `flux-data-qaqc` except "gridmet_file_path" (which is simply the path to a file that is downloaded by *`QaQc.download_gridMET`*) are explained within this page.

The **DATA** section of the config file is where you specify climate variables and their units following the same approach explained above.

Here is a list of all the "expected" climate variable names in the **DATA** section of a config file, where keys are the keys in the config file and values are the internal names used by `flux-data-qaqc`. This list can be accessed at any time from *`Data.variable_names_dict`*:

```
>>> from fluxdataqaqc import Data
>>> Data.variable_names_dict
    {'datestring_col' : 'date' ,
    'net_radiation_col' : 'Rn' ,
    'ground_flux_col' : 'G' ,
    'latent_heat_flux_col' : 'LE' ,
    'latent_heat_flux_corrected_col' : 'LE_user_corr' ,
    'sensible_heat_flux_col' : 'H' ,
    'sensible_heat_flux_corrected_col' : 'H_user_corr' ,
    'shortwave_in_col' : 'sw_in' ,
    'shortwave_out_col' : 'sw_out' ,
    'shortwave_pot_col' : 'sw_pot' ,
    'longwave_in_col' : 'lw_in' ,
    'longwave_out_col' : 'lw_out' ,
    'vap_press_col' : 'vp' ,
    'vap_press_def_col' : 'vpd' ,
    'avg_temp_col' : 't_avg' ,
    'precip_col' : 'ppt' ,
    'wind_spd_col' : 'ws'}
```

You may view these climate entry keys and values (as found in the config file) from within Python using the *`Data.config`* property which contains all information listed in the config file as a `configparser.ConfigParser` instance.

```
>>> config_path = 'config_for_QC_flag_filtering.ini'
>>> d = Data(config_path)
>>> # loop through a list of tuples with keys and values from DATA section
>>> for each in d.config.items('DATA'):
>>>     print(each)
    ('datestring_col', 'date')
    ('net_radiation_col', 'Rn')
    ('net_radiation_units', 'w/m2')
    ('ground_flux_col', 'G')
```

(continues on next page)

(continued from previous page)

```
    ('ground_flux_units', 'w/m2')
    ('latent_heat_flux_col', 'LE')
    ('latent_heat_flux_qc', 'a_qc_value')
    ('latent_heat_flux_units', 'w/m2')
    ('latent_heat_flux_corrected_col', 'LE_corrected')
    ('latent_heat_flux_corrected_units', 'w/m2')
    ('sensible_heat_flux_col', 'H')
    ('sensible_heat_flux_qc', 'a_qc_value')
    ('sensible_heat_flux_units', 'w/m2')
    ('sensible_heat_flux_corrected_col', 'H_corrected')
    ('sensible_heat_flux_corrected_units', 'w/m2')
    ('shortwave_in_col', 'sw_in')
    ('shortwave_in_qc', 'swrad_flag')
    ('shortwave_in_units', 'w/m2')
    ('shortwave_out_col', 'sw_out')
    ('shortwave_out_units', 'w/m2')
    ('shortwave_pot_col', 'sw_pot')
    ('shortwave_pot_units', 'w/m2')
    ('longwave_in_col', 'lw_in')
    ('longwave_in_units', 'w/m2')
    ('longwave_out_col', 'lw_out')
    ('longwave_out_units', 'w/m2')
    ('vap_press_col', 'na')
    ('vap_press_units', 'na')
    ('vap_press_def_col', 'vpd')
    ('vap_press_def_units', 'hPa')
    ('avg_temp_col', 't_avg')
    ('avg_temp_units', 'C')
    ('precip_col', 'ppt')
    ('precip_units', 'mm')
    ('wind_spd_col', 'ws')
    ('wind_spd_units', 'm/s')
```

You can also access the data from the `Data.config` as a dictionary, for example if your **METADATA** section has an entry for "land_cover", e.g.

```
[METADATA]
land_cover = CROP
...
```

then access this value with `configparser.ConfigParser.get` which returns the value of "land_cover" in the config file's **METADATA** section

```
>>> d.config.get('METADATA', 'land_cover')
    CROP
```

Tip: If you are unsure if your config file's metadata contains a specific entry you can pass the `fallback` keyword-only argument to the `configparser.ConfigParser.get` method similar to a Python dictionary.

Here is an example,

```
>>> d.config.get('METADATA', 'land_cov', fallback='not given')
    "not given"
```

2.3 Input formatting and caveats

2.3.1 Missing data

For parsing data gaps in input time series assign the "missing_data_value" to the **METADATA** section of the config file. The value should be numeric, e.g.

```
missing_data_value = -999
```

If the input time series file does not contain all climate variables that are expeced by `flux-data-qaqc`, then specify them as missing ('na') in the config file or simply do not list them in the config. Missing variables will be ignored for the most part and will not be present in output files/plots, however if key variables for the energy balance are not present (LE, H, G, and Rn) then you will not be able to run energy balance closure correction routines.

2.3.2 Data file format

`flux-data-qaqc` accepts Microsoft Excel files (.xlx and .xlsx) and comma separated value (CSV) text files containing time series input. The input file should have a column with combined date and time. Currently there is no restriction on the temporal frequency of input data however it is automatically resampled to daily frequency before running correction routines. Lastly, there should be a single header row containing all variable names followed by the first entry of climatic variables.

Here is an example of a valid input file's first 5 rows and 8 columns:

date	t_avg	sw_pot	sw_in	lw_in	vpd	ppt	ws
2009-01-01	2.803	186.71	123.108	261.302	1.919	0	3.143
2009-01-02	2.518	187.329	121.842	268.946	0.992	0	2.093
2009-01-03	5.518	188.008	124.241	268.004	2.795	0	4.403
2009-01-04	-3.753	188.742	113.793	246.675	0.892	0	4.336

Note: If the the input datas temporal frequency is not recognized `flux-data-qaqc` will attempt to resample it to daily frequency when it is used to create a *QaQc* object. Also, if a value is not recognized a numeric in any data column it will be forced to a null value.

2.3.3 Data header formatting

A common format of some time series data is that the header row may not start on the first line of the file. If this is the case you must add an entry to the **METADATA** section of the config file "skiprows" which states the number of rows to skip before finding the header row. A caveat is that if using CSV data files you may have any number of comment lines before the header so long as they start with a hashtag symbol "#" (comment), in this case you should not add "skiprows" to **METADATA**.

2.3.4 Optimize data load time

`flux-data-qaqc` utilizes the `pandas` for most time series data management, specifically the usage of `datetime.datetime` objects for advanced temporal analysis tools. If your file is large you can specify the datetime format in the **METADATA** section of the config file to potentially greatly speedup the loading of data. For example if your date column contains strings in the format year month day hour minute with no delimiters, e.g. 201401010000 for 2014 January 1st at midnight, then in the `flux-data-qaqc` config file you would enter:

```
date_parser = %Y%m%d%H%M
```

For more information of the correct date parser string for your date format see the directives of the `datetime.datetime.strptime` here.

2.4 Quality-based data filtering

Currently `flux-data-qaqc` supports filtering out poor quality data based on user-provided quality control (QC) values (numeric) or flags (characters) using the *`Data.apply_qc_flags`* method. This feature helps to facilitate manual or semi-manual data filtering which is sometimes necessary during data preprocessing.

2.4.1 Flag-based filtering

Let's say that you have a column in your input data named 'QC_flag' that contains character strings signifying the assigned data quality for a climate time series. The flag is either 'g' meaning a data point is 'good' or if the flag is 'b' the data point is bad quality and you would like to filter it. Further let's say that you want the filter to apply to your latent energy and and sensible heat variables, then in your config file you would need to declare the flag for 'bad' data ('b') to be filtered out in the **METADATA** section:

```
qc_flag = b
```

and in the **DATA** section of your config you will state that the 'QC_flag' column should be applied to your LE and H variables:

```
latent_heat_flux_qc = QC_flag
sensible_heat_flux_qc = QC_flag
```

Now, when the *`Data.apply_qc_flags`* method is used the all date entries of LE and H that have a "QC_flag" value of 'b' will be forced to null in the *`Data.df`* property of a *`Data`* instance.

2.4.2 Threshold-based filtering

Another option is to use a numeric quality control *value* that exists in your input data along with a threshold value which means that when the quality control value falls below this threshold you would like to exclude it from the analysis. Let's assume the column containing the quality control values is named 'QC_values' and it contains values between 0 and 1 with 0 meaning the poorest quality data and 1 being the highest and that you would like to remove all data for select variables with a quality control value below 0.5. Let's further assume that you would like this to apply to your incoming solar radiation variable. Then you would declare the threshold in the **METADATA** section of your config file:

```
qc_threshold = 0.5
```

and in the **DATA** section of your config you will state that the 'QC_value' column should be applied to your incoming shortwave radiation variable:

```
shortwave_in_qc = QC_value
```

Now you are all set to use the functionality, note that you may apply the same quality control value or flag column to multiple climate variables (as shown in the first example). You may also use both numeric qualtiy control values and character string flags for the same input dataset although they cannot both be applied to the same variable. In other wordsf, if you have a column of quality control numeric values it cannot also have character strings mixed in. Another option that is used in the example below is to declare multiple quality control flags that should be filtered out using a comma separated list. For example in the provided example config the flags 'x' and 'b' are used to remove select days from incoming shorwave radiation,

```
qc_flag = x, b
```

There is another option for specifying variables quality control values/flags. Name the column containing the qualtiy control value/flag in your input climate file the same as the variable it corresponds to with the suffix "_QC". For example if your sensible heat column was named **sens_h** then your qualtiy control column should be named **sens_h_QC**. If you use this option you do not need to specify the names in your config file.

2.4.3 Example with flags and thresholds

This example uses the provided time series and config files for QC flag filtering on GitHub. The data is from the FLUXNET 2015 site "ARM USDA UNL OSU Woodward Switchgrass 1". This location exhibits switchgrass fields, more information on this site can be found here.

Because this dataset did not originally contain character Qa/Qc flags they were added for demonstration and applied to shortwave radiation. To view the list of string flags specified in the config file,

```
>>> config_path = 'config_for_QC_flag_filtering.ini'
>>> d = Data(config_path)
>>> # view or reassign the numeric threshold specified in the config file
>>> d.qc_threshold
    0.5
```

And to view the character QC flags if assigned in the config,

```
>>> d.qc_flag
    ['x', 'b']
```

The *Data.qc_var_pairs* attribute shows you which variables were found in your input file that have quality control values assigned, it uses the names as found in the input file,

```
>>> d.qc_var_pairs
    {'LE': 'a_qc_value', 'H': 'a_qc_value', 'sw_in': 'swrad_flag'}
```

Tip: The *Data.qc_var_pairs* dictionary can be updated in Python to assign different columns of QC values to different time series variables.

Now let's apply the QC values.

Note that in this example we mixed both numeric values and threshold with character flags, the numeric values are being applied to LE and H whereas the flags ('x' and 'b') are applied to incoming shortwave radiation.

```
>>> # make copys of before and after the QC filter is applied
>>> no_qc = d.df.input_LE.copy()
>>> no_qc_swrad = d.df.input_sw_in.copy()
>>> # apply QC flags/values
>>> d.apply_qc_flags()
>>> qc_def = d.df.input_LE.copy()
>>> qc_flag_swrad = d.df.input_sw_in.copy()
    WARNING: renaming column Rn to input_Rn
```

(continues on next page)

(continued from previous page)

```
    WARNING: renaming column G to input_G
    WARNING: renaming column LE to input_LE
    WARNING: renaming column H to input_H
    WARNING: renaming column sw_in to input_sw_in
    WARNING: renaming column sw_out to input_sw_out
    WARNING: renaming column sw_pot to input_sw_pot
    WARNING: renaming column lw_in to input_lw_in
    WARNING: renaming column lw_out to input_lw_out
    WARNING: renaming column vpd to input_vpd
    WARNING: renaming column t_avg to input_t_avg
    WARNING: renaming column ppt to input_ppt
    WARNING: renaming column ws to input_ws
```

Note: This is a good time to point out that `flux-data-qaqc` may change the names of your input variables if they exactly match the internal names used by the software (see *`Data.variable_names_dict`*, if this is the case (as is above) a warning message is printed when reading in the data (accessing the `df` or `monthly_df` properties of *`Data`* or *`QaQc`* for the first time) and the names will be modified with a prefix of "_input" as shown above.

Here is a plot showing the data before and after applying the filter.

```
>>> from bokeh.plotting import ColumnDataSource, figure, show
>>> from bokeh.models.formatters import DatetimeTickFormatter
>>> p = figure(x_axis_label='date', y_axis_label='swrad with data removed based on QC␣
↪value')
>>> p.line(no_qc_swrad.index, no_qc_swrad, color='red', legend="no flag", line_
↪width=2)
>>> p.line(no_qc_swrad.index, qc_flag_swrad, color='black', legend="flag = b or x",␣
↪line_width=2)
>>> p.xaxis.formatter = DatetimeTickFormatter(days="%d-%b-%Y")
>>> show(p)
```

And for LE,

```
>>> p = figure(x_axis_label='date', y_axis_label='LE with data removed based on QC␣
↪value')
>>> p.line(no_qc.index, no_qc, color='red', legend="no QC", line_width=2)
>>> p.line(no_qc.index, qc_def, color='black', legend="QC=0.5", line_width=2)
>>> p.xaxis.formatter = DatetimeTickFormatter(days="%d-%b-%Y")
>>> show(p)
```

2.4.4 Alternative naming method for QC data

In this case the climate variables QC columns are named with the same base name as the climate variables with the '_QC' suffix. For example if LE is named 'LE_F_MDS' in your input files header then the QC column is named 'LE_F_MDS_QC'. If your time series data has qualtiy control header names which follow this convention they will automatically be detected and used when you apply them using *`Data.apply_qc_flags`*, i.e. the column names and variables they should be assigned to do not need to be declared in the config.ini file.

```
>>> import os
>>> config_path = os.path.join('..','Basic_usage','fluxnet_config.ini')
>>> d = Data(config_path)
>>> # view input files header, note the QC columns
```

(continues on next page)

(continued from previous page)

```
>>> d.header
    Index(['TIMESTAMP', 'TA_F', 'TA_F_QC', 'SW_IN_POT', 'SW_IN_F', 'SW_IN_F_QC',
           'LW_IN_F', 'LW_IN_F_QC', 'VPD_F', 'VPD_F_QC', 'PA_F', 'PA_F_QC', 'P_F',
           'P_F_QC', 'WS_F', 'WS_F_QC', 'USTAR', 'USTAR_QC', 'NETRAD', 'NETRAD_QC',
           'PPFD_IN', 'PPFD_IN_QC', 'PPFD_OUT', 'PPFD_OUT_QC', 'SW_OUT',
           'SW_OUT_QC', 'LW_OUT', 'LW_OUT_QC', 'CO2_F_MDS', 'CO2_F_MDS_QC',
           'TS_F_MDS_1', 'TS_F_MDS_1_QC', 'SWC_F_MDS_1', 'SWC_F_MDS_1_QC',
           'G_F_MDS', 'G_F_MDS_QC', 'LE_F_MDS', 'LE_F_MDS_QC', 'LE_CORR',
           'LE_CORR_25', 'LE_CORR_75', 'LE_RANDUNC', 'H_F_MDS', 'H_F_MDS_QC',
           'H_CORR', 'H_CORR_25', 'H_CORR_75', 'H_RANDUNC', 'NEE_VUT_REF',
           'NEE_VUT_REF_QC', 'NEE_VUT_REF_RANDUNC', 'NEE_VUT_25', 'NEE_VUT_50',
           'NEE_VUT_75', 'NEE_VUT_25_QC', 'NEE_VUT_50_QC', 'NEE_VUT_75_QC',
           'RECO_NT_VUT_REF', 'RECO_NT_VUT_25', 'RECO_NT_VUT_50', 'RECO_NT_VUT_75',
           'GPP_NT_VUT_REF', 'GPP_NT_VUT_25', 'GPP_NT_VUT_50', 'GPP_NT_VUT_75',
           'RECO_DT_VUT_REF', 'RECO_DT_VUT_25', 'RECO_DT_VUT_50', 'RECO_DT_VUT_75',
           'GPP_DT_VUT_REF', 'GPP_DT_VUT_25', 'GPP_DT_VUT_50', 'GPP_DT_VUT_75',
           'RECO_SR', 'RECO_SR_N'],
          dtype='object')
```

Verify that the QC columns have been paired with corresponding climate variables

```
>>> d.qc_var_pairs
    {'NETRAD': 'NETRAD_QC',
     'G_F_MDS': 'G_F_MDS_QC',
     'LE_F_MDS': 'LE_F_MDS_QC',
     'H_F_MDS': 'H_F_MDS_QC',
     'SW_IN_F': 'SW_IN_F_QC',
     'SW_OUT': 'SW_OUT_QC',
     'LW_IN_F': 'LW_IN_F_QC',
     'LW_OUT': 'LW_OUT_QC',
     'VPD_F': 'VPD_F_QC',
     'TA_F': 'TA_F_QC',
     'P_F': 'P_F_QC',
     'WS_F': 'WS_F_QC'}
```

Note: FLUXNET files include their own qualtiy control flags for sensible heat and other variables where quality threshold columns are named the same as the climate variable they correspond to with the "_QC" suffix. Therefore they do not need to be defined in a config file before applying them.

For the dataset defined in the example "FLUXNET_config.ini" we did not specify a QC threshold or flag(s) in the config file, therefore we must assign it when calling the `Data.apply_qc_flags` method (shown in *Example of threshold filtering*).

```
>>> # view the QC threshold specified in the config file
>>> print(d.qc_threshold, type(d.qc_threshold))
    None <class 'NoneType'>
```

Alternatively, you may assign the threshold of flag values at any time directly to a `Data` instance:

```
>>> d.qc_threshold = .75
```

2.4.5 Example of threshold filtering

Be sure to validate QC thresholds or flags before applying them to make sure everything seems correct. Below we see that the lowest QC values correspond with poor quality gap-fill data near the begining of the time series of sensible heat (H).

```
>>> from bokeh.models import LinearAxis, Range1d
>>> p = figure(x_axis_label='date', y_axis_label='sensible heat flux (w/m2)')
>>> p.extra_y_ranges = {"sec": Range1d(start=-0.1, end=1.1)}
>>> p.line(d.df.index, d.df['H_F_MDS'], color='red', line_width=1, legend='data')
>>> p.add_layout(LinearAxis(y_range_name="sec", axis_label='QC value'), 'right')
>>> p.circle(d.df.index, d.df['H_F_MDS_QC'], line_width=2, y_range_name="sec", legend=
↪'QC')
>>> p.x_range=Range1d(d.df.index[0], d.df.index[365])
>>> p.xaxis.formatter = DatetimeTickFormatter(days="%d-%b-%Y")
>>> p.legend.location = "top_left"
>>> show(p)
    WARNING: Insufficient data to calculate mean for multiple G measurements
    WARNING: Insufficient data to calculate mean for multiple THETA measurements
```

As a reminder, the routine provided for numeric or theshold filtering removes all data entries that have been assigned to a QC column and have a QC value that falls below some threshold.

```
>>> # apply QC numeric threshold filters
>>> d.apply_qc_flags(threshold=0.5)
```

Values with QC values < 0.5 are now removed (null) for any variable listed in *`Data.qc_var_pairs`*.

Caution: The *`Data.apply_qc_flags`* method applies the filter to all variables in the climate file that have a QC column if columns are not specified in the config file.

To see all columns (variables) that may have been affected by the previous filter or to constrain them, modify the declarations in the config file or within *`Data.qc_var_pairs`*, i.e.

```
>>> d.qc_var_pairs
    {'NETRAD': 'NETRAD_QC',
    'G_F_MDS': 'G_F_MDS_QC',
    'LE_F_MDS': 'LE_F_MDS_QC',
    'H_F_MDS': 'H_F_MDS_QC',
    'SW_IN_F': 'SW_IN_F_QC',
    'SW_OUT': 'SW_OUT_QC',
    'LW_IN_F': 'LW_IN_F_QC',
    'LW_OUT': 'LW_OUT_QC',
    'VPD_F': 'VPD_F_QC',
    'TA_F': 'TA_F_QC',
    'P_F': 'P_F_QC',
    'WS_F': 'WS_F_QC'}
```

Now let's view the same sesnible heat flux time series after applying the threshold filter, notice the strange oscillating artifact near the beginning of the time series as been removed:

```
>>> p = figure(x_axis_label='date', y_axis_label='sensible heat flux (w/m2)')
>>> p.extra_y_ranges = {"sec": Range1d(start=-0.1, end=1.1)}
>>> p.line(d.df.index, d.df['H_F_MDS'], color='red', line_width=1, legend='data')
>>> p.add_layout(LinearAxis(y_range_name="sec", axis_label='QC value'), 'right')
>>> p.circle(d.df.index, d.df['H_F_MDS_QC'], line_width=2, y_range_name="sec", legend=
↪'QC')
>>> p.x_range=Range1d(d.df.index[0], d.df.index[365])
>>> p.xaxis.formatter = DatetimeTickFormatter(days="%d-%b-%Y")
>>> p.legend.location = "top_left"
>>> show(p)
```

See also:

Step 0, manual cleaning of poor quality data for an example that shows how to filter poor quality data after loading data into a `QaQc` object.

2.5 Averaging data from multiple sensors

2.5.1 Non-weighted averaging

If the climate station being analyzed has multiple sensors for the same variable (e.g. sensible heat flux) you can easily tell `flux-data-qaqc` to use their non-weighted average of for `flux-data-qaqc` routines including energy balance closure corrections or interactive visualizations. To do so simply list the variable names (as found in the file header) with a delimiter of your choice and then list the delimiter in the **METADATA** section. Example, if you have three sensible heat variables named "h_1", "sens_h_2", and "sensible heat, (w/m2)" then in your config file's **METADATA** you would write:

```
var_name_delim = ;
```

and the sensible heat assignment in the **DATA** section would read:

```
sensible_heat_flux_col = h_1;sens_h_2;sensible heat, (w/m2)
```

Caution: Because there is a comma in the last variable name we cannot use a comma as the name delimiter. Also, if you do not state the delimiter of variable names in the **METADATA** section of the config file, `flux-data-qaqc` will look for the single variable name "h_1;sens_h_2;sensible heat, (w/m2)" in the header which will not be found.

`flux-data-qaqc` will name the average in this case as H_mean, in general it will add the suffix "_mean" to the internal name of the variable used by `flux-data-qaqc` which can be found in the keys of the `Data.variable_names_dict` dictionary.

Hint: If you use any averaging option for an energy balance component, i.e. latent energy, sensible heat, net radiation, or soil heat flux, the average will also be used in energy balance closure corrections.

2.5.2 Weighted averaging

`flux-data-qaqc` provides the ability to read in multiple soil heat flux/moisture variables for a given station location, calculate their weighted or non weighted average, and write/plot their daily and monthly time series. *Currently weighted averaging is only provided for soil heat flux and soil moisture variables*, using this config option is also the only way to automatically produce time series plots of these variables when using `QaQc.plot`. This may be useful for comparing/validating multiple soil heat/moisture probes at varying locations or depths or varying instrumentation.

Here is what you need to do to use this functionality:

1. List the multiple soil variable names in your config file's **DATA** section following the convention:

- For multiple soil heat flux variables config names should begin with "G_" or "g_" followed by an integer starting with 1,2,3,… i.e. g_[number]. For example:

```
g_1 = name_of_my_soil_heat_flux_variable
```

For soil moisture variables the name of the config variable should follow "theta_[number]" for example:

```
theta_1 = name_of_my_soil_moisture_variable
```

2. List the units of each variable. To specify the units of your soil flux/moisture variables add "_units" to the config name you assigned:

```
g_1_units = w/m2
theta_1_units = cm
```

3. To set weights for multiple variables to compute weighted averages assign the "_weight" suffix to their names in the config file. For example, to set weights for multiple soil heat flux variables:

```
g_1_weight = 0.25
g_2_weight = 0.25
g_3_weight = 0.5
```

Hint: If weights are not given the arithmetic mean will be calculated. Or if the weights do not sum to 1, they will be automatically normalized so that they do.

As in the case for non-weighted averaging for any energy balance component, if you use this option for soil heat flux (G), the weighted average will also be used in energy balance closure corrections.

2.5.3 Weighted average example

This example uses time series data recorded from the "ARM Southern Great Plains site- Lamont" AmeriFlux eddy covariance tower, more information on this site can be found here.

Here is the **DATA** section of the config file that defines the multiple G variables in the input data file used for the example below, we put a much higher weight on the G sensors "G_2_1_1" and "G_3_1_1",

```
[DATA]
g_1 = G_1_1_1
g_1_units = w/m2
g_1_weight = 1
g_2 = G_2_1_1
g_2_units = w/m2
g_2_weight = 10
```

(continues on next page)

(continued from previous page)

```
g_3 = G_3_1_1
g_3_units = w/m2
g_3_weight = 10
g_4 = G_4_1_1
g_4_units = w/m2
g_4_weight = 1
...
```

Note, the naming system of these variables (from AmeriFlux conventions) indicates that the multiple G sensors are spaced in differing horizontal locations from one another.

There are many soil moisture sensors at this site, because we are not using these variables within any calculations and simply want them to be loaded in and later plotted we will not assign weights to them and therefore the arithmetic mean will be calculated and added to output plots and time series files. Here is what is listed in the **DATA** section of the config file for multiple soil moisture recordings in this case:

```
[DATA]
theta_1 = SWC_1_1_1
theta_1_units = (%): Soil water content (volumetric), range 0-100
theta_2 = SWC_2_1_1
theta_2_units = (%): Soil water content (volumetric), range 0-100
theta_3 = SWC_1_2_1
theta_3_units = (%): Soil water content (volumetric), range 0-100
theta_4 = SWC_2_2_1
theta_4_units = (%): Soil water content (volumetric), range 0-100
theta_5 = SWC_3_1_1
theta_5_units = (%): Soil water content (volumetric), range 0-100
theta_6 = SWC_4_1_1
theta_6_units = (%): Soil water content (volumetric), range 0-100
theta_7 = SWC_3_2_1
theta_7_units = (%): Soil water content (volumetric), range 0-100
theta_8 = SWC_4_2_1
theta_8_units = (%): Soil water content (volumetric), range 0-100
theta_9 = SWC_1_3_1
theta_9_units = (%): Soil water content (volumetric), range 0-100
theta_10 = SWC_1_4_1
theta_10_units = (%): Soil water content (volumetric), range 0-100
theta_11 = SWC_1_5_1
theta_11_units = (%): Soil water content (volumetric), range 0-100
theta_12 = SWC_1_6_1
theta_12_units = (%): Soil water content (volumetric), range 0-100
theta_13 = SWC_2_3_1
theta_13_units = (%): Soil water content (volumetric), range 0-100
theta_14 = SWC_2_3_2
theta_14_units = (%): Soil water content (volumetric), range 0-100
theta_15 = SWC_2_2_2
theta_15_units = (%): Soil water content (volumetric), range 0-100
theta_16 = SWC_2_1_2
theta_16_units = (%): Soil water content (volumetric), range 0-100
...
```

Hint: The units for soil moisture variables will be used in the y-axis daily and monthly time series plots when they are created by `QaQc.plot`.

Now that the config file has been setup, let's verify that everything was read in correctly,

```
>>> # read in the data
>>> config_path = 'config_for_multiple_soil_vars.ini'
>>> d = Data(config_path)
>>> # note the newly added multiple g and theta variables
>>> d.variables
    {'date': 'TIMESTAMP_START',
    'Rn': 'NETRAD_1_1_1',
    'LE': 'LE_1_1_1',
    'H': 'H_1_1_1',
    'sw_in': 'SW_IN_1_1_1;SW_IN_1_1_2',
    'sw_out': 'SW_OUT_1_1_1',
    'lw_in': 'LW_IN_1_1_1',
    'lw_out': 'LW_OUT_1_1_1',
    'vpd': 'VPD_PI_1_1_1',
    't_avg': 'T_SONIC_1_1_1',
    'ws': 'WS_1_1_1;WS_1_2_1',
    'g_1': 'G_1_1_1',
    'g_2': 'G_2_1_1',
    'g_3': 'G_3_1_1',
    'g_4': 'G_4_1_1',
    'theta_1': 'SWC_1_1_1',
    'theta_2': 'SWC_2_1_1',
    'theta_3': 'SWC_1_2_1',
    'theta_4': 'SWC_2_2_1',
    'theta_5': 'SWC_3_1_1',
    'theta_6': 'SWC_4_1_1',
    'theta_7': 'SWC_3_2_1',
    'theta_8': 'SWC_4_2_1',
    'theta_9': 'SWC_1_3_1',
    'theta_10': 'SWC_1_4_1',
    'theta_11': 'SWC_1_5_1',
    'theta_12': 'SWC_1_6_1',
    'theta_13': 'SWC_2_3_1',
    'theta_14': 'SWC_2_3_2',
    'theta_15': 'SWC_2_2_2',
    'theta_16': 'SWC_2_1_2'}
```

Note, the windspeed and shortwave incoming radtiation columns were assigned multiple variables as well, these will be used to calculate the non-weighted mean as described in *Non-weighted averaging*.

Check the units assignment:

```
>>> d.units
    {'Rn': 'w/m2',
     'LE': 'w/m2',
     'H': 'w/m2',
     'sw_in': 'w/m2',
     'sw_out': 'w/m2',
     'lw_in': 'w/m2',
     'lw_out': 'w/m2',
     'vpd': 'hPa',
     't_avg': 'C',
     'ws': 'm/s',
     'g_1': 'w/m2',
     'g_2': 'w/m2',
     'g_3': 'w/m2',
     'g_4': 'w/m2',
     'theta_1': '(%): Soil water content (volumetric), range 0-100',
```

(continues on next page)

(continued from previous page)

```
     'theta_2': '(%): Soil water content (volumetric), range 0-100',
     'theta_3': '(%): Soil water content (volumetric), range 0-100',
     'theta_4': '(%): Soil water content (volumetric), range 0-100',
     'theta_5': '(%): Soil water content (volumetric), range 0-100',
     'theta_6': '(%): Soil water content (volumetric), range 0-100',
     'theta_7': '(%): Soil water content (volumetric), range 0-100',
     'theta_8': '(%): Soil water content (volumetric), range 0-100',
     'theta_9': '(%): Soil water content (volumetric), range 0-100',
     'theta_10': '(%): Soil water content (volumetric), range 0-100',
     'theta_11': '(%): Soil water content (volumetric), range 0-100',
     'theta_12': '(%): Soil water content (volumetric), range 0-100',
     'theta_13': '(%): Soil water content (volumetric), range 0-100',
     'theta_14': '(%): Soil water content (volumetric), range 0-100',
     'theta_15': '(%): Soil water content (volumetric), range 0-100',
     'theta_16': '(%): Soil water content (volumetric), range 0-100'}
```

View these variables and their weights as written in the config file:

```
>>> d.soil_var_weight_pairs
    {'g_1': {'name': 'added_G_col', 'weight': '6'},
     'g_2': {'name': 'another_G_var', 'weight': '2'},
     'g_3': {'name': 'G', 'weight': '0.5'},
     'g_4': {'name': 'final_G_var', 'weight': '0.25'},
     'g_5': {'name': 'yet_another_G', 'weight': '0.25'},
     'theta_1': {'name': 'soil_moisture_z1', 'weight': '0.25'},
     'theta_2': {'name': 'soil_moisture_z10', 'weight': '0.75'}}
```

When the data is first loaded into memory the weighted (and non-weighted) averages are calculated. At this stage weights will be automatically normalized so that they sum to one and the new weights will be printed if this occurs.

```
>>> # load daily or monthly dataframe to calculate the weighted averages if they exist
>>> d.df.head();
    g weights not given or don't sum to one, normalizing
    Here are the new weights:
     G_1_1_1:0.05, G_2_1_1:0.45, G_3_1_1:0.45, G_4_1_1:0.05
    Calculating mean for var: THETA from columns: ['SWC_1_1_1', 'SWC_2_1_1', 'SWC_1_2_
↪1', 'SWC_2_2_1', 'SWC_3_1_1', 'SWC_4_1_1', 'SWC_3_2_1', 'SWC_4_2_1', 'SWC_1_3_1',
↪'SWC_1_4_1', 'SWC_1_5_1', 'SWC_1_6_1', 'SWC_2_3_1', 'SWC_2_3_2', 'SWC_2_2_2', 'SWC_
↪2_1_2']
    Calculating mean for var: sw_in
     from columns: ['SW_IN_1_1_1', 'SW_IN_1_1_2']
    Calculating mean for var: ws
     from columns: ['WS_1_1_1', 'WS_1_2_1']
```

In this example, shortwave incoming radiation and windspeed were also averaged (non-weighted) from multiple recordings as described in *Non-weighted averaging*.

The weights have been changed and updated as we would expect for G, you may ignore the weights for soil moisture in this case- because they were not assigned the arithmetic mean is calculated and the weights are not used.

```
>>> d.soil_var_weight_pairs
    {'g_1': {'name': 'G_1_1_1', 'weight': 0.045454545454545456},
     'g_2': {'name': 'G_2_1_1', 'weight': 0.45454545454545453},
     'g_3': {'name': 'G_3_1_1', 'weight': 0.45454545454545453},
     'g_4': {'name': 'G_4_1_1', 'weight': 0.045454545454545456},
     'theta_1': {'name': 'SWC_1_1_1', 'weight': 1},
     'theta_2': {'name': 'SWC_2_1_1', 'weight': 1},
```

(continues on next page)

(continued from previous page)

```
      'theta_3': {'name': 'SWC_1_2_1', 'weight': 1},
      'theta_4': {'name': 'SWC_2_2_1', 'weight': 1},
      'theta_5': {'name': 'SWC_3_1_1', 'weight': 1},
      'theta_6': {'name': 'SWC_4_1_1', 'weight': 1},
      'theta_7': {'name': 'SWC_3_2_1', 'weight': 1},
      'theta_8': {'name': 'SWC_4_2_1', 'weight': 1},
      'theta_9': {'name': 'SWC_1_3_1', 'weight': 1},
      'theta_10': {'name': 'SWC_1_4_1', 'weight': 1},
      'theta_11': {'name': 'SWC_1_5_1', 'weight': 1},
      'theta_12': {'name': 'SWC_1_6_1', 'weight': 1},
      'theta_13': {'name': 'SWC_2_3_1', 'weight': 1},
      'theta_14': {'name': 'SWC_2_3_2', 'weight': 1},
      'theta_15': {'name': 'SWC_2_2_2', 'weight': 1},
      'theta_16': {'name': 'SWC_2_1_2', 'weight': 1}}
```

Now the dataframe also has the weighted means that will be named g_mean and theta_mean,

```
>>> d.df.columns
    Index(['input_t_avg', 'input_sw_pot', 'input_sw_in', 'input_lw_in',
           'input_vpd', 'input_ppt', 'input_ws', 'input_Rn', 'input_sw_out',
           'input_lw_out', 'input_G', 'input_LE', 'LE_corrected', 'input_H',
           'H_corrected', 'added_G_col', 'another_G_var', 'final_G_var',
           'yet_another_G', 'soil_moisture_z1', 'soil_moisture_z10', 'a_qc_value',
           'swrad_flag', 'g_mean', 'theta_mean'],
          dtype='object')
```

Note: Even though we did not specify "ground_flux_col" in the config file, the weighted average value has now been used to update this variable. Therefore the weighted mean will be used in energy balance closure correction routines if they are subsequently run.

Check which variable will be used as G later if closure corrections are used:

```
>>> d.variables.get('G')
    'g_mean'
```

Now, let's visualize the resulting weighted average of multiple G measurements and their individual daily time series,

```
>>> # get just G columns for plot arguments
>>> G_cols = [c for c in d.df.columns if c.startswith(('g_','G_'))]
>>> G_cols
    ['G_1_1_1', 'G_2_1_1', 'G_3_1_1', 'G_4_1_1', 'g_mean']
```

The example below creates the time series plot with a short span of data for easier visibility of weighted mean, it also used the plot routines provided by *Data* and *QaQc* which are inhereted from the *Plot* class within `flux-data-qaqc`. Specifically this example utilizes *Plot.add_lines* which makes the time series plotting of multiple variables more efficient and automatically handles the hover tooltips.

```
>>> from fluxdataqaqc.plot import ColumnDataSource # for hover tooltips
>>> # shorter period for visualization
>>> df = d.df.loc['01/01/2008':'05/01/2008', G_cols]
>>> plt_vars = G_cols
>>> colors = ['blue', 'red', 'orange', 'green', 'black']
>>> x_name = 'date'
>>> source = ColumnDataSource(df)
```

(continues on next page)

(continued from previous page)

```
>>> fig = figure(x_axis_label='date', y_axis_label='Soil heat flux (w/m2)')
>>> Data.add_lines(fig, df, plt_vars, colors, x_name, source, labels=G_cols)
>>> show(fig)
```

Note, the weighted mean is closer to 'G_2_1_1' and 'G_3_1_1' as we gave them weights of 10 versus 1 to 'G_1_1_1' and 'G_4_1_1'.

Lastly, the code snippets below run the Energy Balance Ratio closure correction and creating the default plots in order to view the daily and monthly time series of multiple soil moisture variables. It also shows how to upload the output plot file into a Jupyter Notebook for viewing.

```
>>> # in order to correctly view the output in a Jupyter notebook
>>> from bokeh.io import output_notebook
>>> output_notebook()
```

Within the set of default plots created by the `QaQc.plot` method will include interactive daily and monthly time series of multiple *G* and soil moisture variables if they were assigned in the input config file (as in this example), scroll down to view them.

```
>>> from fluxdataqaqc import QaQc
>>> q = QaQc(d)
>>> q.correct_data()
>>> # this will NOT save the plot file, use output_type='save'
>>> q.plot(output_type='show')
```

CHAPTER 3

Tutorial

This tutorial demonstrates the most important features of the `flux-data-qaqc` Python package for management, analysis, and visualization of eddy covariance time series data. It is recommended to read the *Installation* and *Configuration Options and Caveats* tutorials before this one.

A Jupyter Notebook of this tutorial is available here.

Tip: Currently, the software does not include a command line interface therefore to use the software you must use Python, e.g. make your own scripts or use an interactive shell. However, you will see that common workflows can be accomplished with a few (5-10) lines of code and you can simply follow the templates given here to make custom scripts.

3.1 Description of example datasets

The data for this example comes from the "Twitchell Alfalfa" AmeriFlux eddy covariance flux tower site in California. The site is located in alfalfa fields and exhibits a mild Mediterranean climate with dry and hot summers, for more information on this site or to download data click here.

3.2 Loading input

The loading and management of input climatic data and metadata from a config.ini file is done using the `fluxdataqaqc.Data` object. In a nutshell, a `Data` object is created from a properly formatted config file (see *Setting up a config file*) and has tools for parsing input climate data, averaging input climate time series, accessing/managing metadata, flag-based data filtering, and creating interactive visualizations of input data.

There is only one argument to create a Data object, the path to the config.ini file:

```
>>> # imports for code snippets within tutorial
>>> import pandas as pd
>>> from fluxdataqaqc import Data, QaQc, Plot
>>> from bokeh.plotting import figure, show, ColumnDataSource
>>> from bokeh.models.formatters import DatetimeTickFormatter
>>> from bokeh.models import LinearAxis, Range1d
>>> # create a Data object from the config.ini file
>>> config_path = 'US-Tw3_config.ini'
>>> d = Data(config_path)
```

3.2.1 Attributes of a Data object

Below are some of the useful attributes of the `Data` object and how they may be used.

The full path to the config.ini file that was used to create the `Data` instance can be accessed, note that it will return a system-depenedent `pathlib.Path` object. E.g. on my Linux machine the path is:

```
>>> d.config_file
    PosixPath('/home/john/flux-data-qaqc/examples/Basic_usage/US-Tw3_config.ini')
```

On a Windows machine the path will have the appropriate backslashes.

Similarly to access the climate time series file:

```
>>> d.climate_file
    PosixPath('/home/john/flux-data-qaqc/examples/Basic_usage/AMF_US-Tw3_BASE_HH_5-5.
↪csv')
```

The `Data.config` attribute is a `configparser.ConfigParser` object, it allows you to access metadata and data in the config file in multiple ways and to modify them. In `flux-data-qaqc` it is mainly used for accessing information about the input data.

```
>>> # get a list of all entries in the METADATA section of the config.ini
>>> d.config.items('METADATA') # access the DATA section the same way
    [('climate_file_path', 'AMF_US-Tw3_BASE_HH_5-5.csv'),
     ('station_latitude', '38.1159'),
     ('station_longitude', '-121.6467'),
     ('station_elevation', '-9.0'),
     ('missing_data_value', '-9999'),
     ('skiprows', '2'),
     ('date_parser', '%Y%m%d%H%M'),
     ('site_id', 'US-Tw3'),
     ('country', 'USA'),
     ('doi_contributor_name', 'Dennis Baldocchi'),
     ('doi_contributor_role', 'Author'),
     ('doi_contributor_email', 'baldocchi@berkeley.edu'),
     ('doi_contributor_institution', 'University of California, Berkeley'),
     ('doi_organization', 'California Department of Water Resources'),
     ('doi_organization_role', 'Sponsor'),
     ('flux_measurements_method', 'Eddy Covariance'),
     ('flux_measurements_variable', 'CO2'),
     ('flux_measurements_operations', 'Continuous operation'),
     ('site_name', 'Twitchell Alfalfa'),
     ('igbp', 'CRO'),
     ('igbp_comment',
      'alfalfa is a fast growing leguminous crop raised for animal feed of low␣
↪stature.  It is planted in rows and typically reaches 60-70 cm in height prior to
↪harvest.'),
```

(continues on next page)

(continued from previous page)

```
    ('land_ownership', 'public'),
    ('network', 'AmeriFlux'),
    ('reference_paper',
     'Baldocchi, D., Penuelas, J. (2018) The Physics And Ecology Of Mining Carbon␣
↪Dioxide From The Atmosphere By Ecosystems, Global Change Biology, 45(), 9275-9287'),
    ('reference_doi', '10.1111/gcb.14559'),
    ('reference_usage', 'Reference'),
    ('research_topic',
     'The research approach of the University of California, Berkeley Biometeorology␣
↪Laboratory involves the coordinated use of experimental measurements and␣
↪theoretical models to understand the physical, biological, and chemical processes␣
↪that control trace gas fluxes between the biosphere and atmosphere and to quantify␣
↪their temporal and spatial variations. The research objectives of the Mayberry␣
↪Wetland, Twitchell Wetland, Sherman Island, Twitchell Island, Twitchell Alfalfa, ␣
↪and Twitchell Corn sites are as follows: 1) Describe differences in the fluxes of␣
↪CO2, CH4, H2O, and energy between different land uses, 2) Understand the mechanisms␣
↪controlling these fluxes, 3) Use ecosystem modeling to understand controls on these␣
↪mechanisms under different environmental scenarios. These six sites were selected␣
↪to capture a wide range of inundated conditions within the Sacramento-San Joaquin␣
↪River Delta. The research focuses on the eddy covariance technique to measure CH4,␣
↪CO2, H2O, and energy fluxes and works to combine measurements of both net fluxes␣
↪and partitioned fluxes in order to achieve a mechanistic understanding of the␣
↪ecological controls on current and future carbon flux in the Delta.'),
    ('terrain', 'Flat'),
    ('aspect', 'FLAT'),
    ('wind_direction', 'W'),
    ('surface_homogeneity', '370.0'),
    ('site_desc',
     "The Twitchell Alfalfa site is an alfalfa field owned by the state of␣
↪California and leased to third parties for farming. The tower was installed on May␣
↪24, 2013. This site and the surrounding region are part of the San Joaquin -␣
↪Sacramento River Delta drained beginning in the 1850's and subsequently used for␣
↪agriculture. The field has been alfalfa for X years...., Crop rotation occurs every␣
↪5-6 years.  The site is harvested by mowing and bailing several times per year. ␣
↪The field is fallow typically between November and February. The site is irrigated␣
↪by periodically-flooded ditches surrounding the field. The site is irrigated by␣
↪raising, and subsequently lowering the water table??"),
    ('site_funding', 'California Department of Water Resources'),
    ('team_member_name', 'Joe Verfaillie'),
    ('team_member_role', 'Technician'),
    ('team_member_email', 'jverfail@berkeley.edu'),
    ('team_member_institution', 'University of California, Berkeley'),
    ('url_ameriflux', 'http://ameriflux.lbl.gov/sites/siteinfo/US-Tw3'),
    ('utc_offset', '-8'),
    ('mat', '15.6'),
    ('map', '421.0'),
    ('land_owner', 'California Department of Water Resources'),
    ('climate_koeppen', 'Csa'),
    ('doi', '10.17190/AMF/1246149'),
    ('doi_citation',
     'Dennis Baldocchi (2013-) AmeriFlux US-Tw3 Twitchell Alfalfa, 10.17190/AMF/
↪1246149'),
    ('doi_dataproduct', 'AmeriFlux'),
    ('team_member_address',
     'Department of Environmental Science, Policy and Management, 137 Mulford Hall,␣
↪345 Hilgard Hall,Berkeley, CA USA 94720-3110'),
    ('url', 'http://nature.berkeley.edu/biometlab/sites.php?tab=US-Tw3'),
```

(continues on next page)

(continued from previous page)

```
     ('dom_dist_mgmt', 'Agriculture'),
     ('site_snow_cover_days', '0.0'),
     ('state', 'CA'),
     ('location_date_start', '20130524.0'),
     ('acknowledgement',
      'Biometeorology Lab, University of California, Berkeley, PI:  Dennis Baldocchi
↪')]
```

A useful method is the `configparser.ConfigParser.get` which takes the section of the config file and the "option" and returns the value:

```
>>> d.config.get(section='METADATA', option='site_name')
    'Twitchell Alfalfa'
```

```
>>> # section and option are optional keywords
>>> d.config.get('METADATA', 'site_name')
    'Twitchell Alfalfa'
```

Tip: If you are unsure if an entry or option exists in the config file, use the `fallback` keyword argument

```
>>> # section and option are optional keywords
>>> d.config.get('METADATA', 'site name', fallback='na')
    'na'
```

Some metadata entries are added as *`Data`* attributes for easier access as they are used in multiple ways later, these include:

- site_id*
- elevation*
- latitude*
- longitude*
- na_val
- qc_threshold
- qc_flag

*mandatory **METADATA** entries in the config file, see *Setting up a config file* for further explanation.

View all the columns as found in the header row of the input time series climate file.

```
>>> d.header
    array(['TIMESTAMP_START', 'TIMESTAMP_END', 'CO2', 'H2O', 'CH4', 'FC',
           'FCH4', 'FC_SSITC_TEST', 'FCH4_SSITC_TEST', 'G', 'H', 'LE',
           'H_SSITC_TEST', 'LE_SSITC_TEST', 'WD', 'WS', 'USTAR', 'ZL', 'TAU',
           'MO_LENGTH', 'V_SIGMA', 'W_SIGMA', 'TAU_SSITC_TEST', 'PA', 'RH',
           'TA', 'VPD_PI', 'T_SONIC', 'T_SONIC_SIGMA', 'SWC_1_1_1',
           'SWC_1_2_1', 'TS_1_1_1', 'TS_1_2_1', 'TS_1_3_1', 'TS_1_4_1',
           'TS_1_5_1', 'NETRAD', 'PPFD_DIF', 'PPFD_IN', 'PPFD_OUT', 'SW_IN',
           'SW_OUT', 'LW_IN', 'LW_OUT', 'P', 'FC_PI_F', 'RECO_PI_F',
           'GPP_PI_F', 'H_PI_F', 'LE_PI_F'], dtype='<U15')
```

Note: All of the header columns will not necessarily be loaded, only those specified in the config file. Also, no data other than the header line is loaded into memory when creating a `Data` object, the time series data is only loaded when calling `Data.df` for increased efficiency for some workflows involving only metadata.

3.2.2 Variable names and units

In `flux-data-qaqc` there are two naming schemes for climate variables, the names as defined by the column headers in the input time series file and the internal names for some variables and calculated variables created by the package. We will refer to these two sets as "user-defined" and "internal" names hereforth.

The `Data.variables` attribute maps the internal to user-defined variable names:

```
>>> d.variables
    {'date': 'TIMESTAMP_START',
     'Rn': 'NETRAD',
     'G': 'G',
     'LE': 'LE_PI_F',
     'H': 'H_PI_F',
     'sw_in': 'SW_IN',
     'sw_out': 'SW_OUT',
     'lw_in': 'LW_IN',
     'lw_out': 'LW_OUT',
     'vpd': 'VPD_PI',
     't_avg': 'T_SONIC',
     'ws': 'WS',
     'theta_1': 'SWC_1_1_1',
     'theta_2': 'SWC_1_2_1'}
```

And, the `Data.inv_map` maps the internal to user-defined names if they differ, however this is only created once the data is loaded by calling `Data.df`.

```
>>> # a similar dictionary attribute for input units
>>> d.units
    {'Rn': 'w/m2',
     'G': 'w/m2',
     'LE': 'w/m2',
     'H': 'w/m2',
     'sw_in': 'w/m2',
     'sw_out': 'w/m2',
     'lw_in': 'w/m2',
     'lw_out': 'w/m2',
     'vpd': 'hPa',
     't_avg': 'C',
     'ws': 'm/s',
     'theta_1': '(%): Soil water content (volumetric), range 0-100',
     'theta_2': '(%): Soil water content (volumetric), range 0-100'}
```

3.2.3 Accessing input data

The `Data.df` property gves access to the time series input climate data for columns specified in the config file as a datetime-indexed `pandas.DataFrame` object. This object has numerous powerful built in tools for time series analysis and visualization.

```
>>> # first 5 datetimes that are not gaps
>>> d.df.dropna().head()
```

Tip: There are *many* tutorials on how to use the `pandas.DataFrame` and its powerful data analysis tools for multiple purposes online, to get started you may want to visit Panda's own list of tutorials here.

By default the column names in `Data.df` are retained from user-defined names unless they were named exactly the same as an internal name. For example the input ground heat flux column in this dataset is named "G", therefore it was renamed as "input_g"

```
>>> d.df.columns
    Index(['input_G', 'WS', 'VPD_PI', 'T_SONIC', 'SWC_1_1_1', 'SWC_1_2_1',
           'NETRAD', 'SW_IN', 'SW_OUT', 'LW_IN', 'LW_OUT', 'H_PI_F', 'LE_PI_F',
           'theta_mean'],
          dtype='object')
```

```
>>> # the new name was also updated in Data.variables
>>> d.variables.get('G')
    'input_G'
```

As stated earlier, `Data.inv_map` maps the user-defined names to internal `flux-data-qaqc` names only after loading `Data.df`:

```
>>> d.inv_map
    {'TIMESTAMP_START': 'date',
     'NETRAD': 'Rn',
     'input_G': 'G',
     'LE_PI_F': 'LE',
     'H_PI_F': 'H',
     'SW_IN': 'sw_in',
     'SW_OUT': 'sw_out',
     'LW_IN': 'lw_in',
     'LW_OUT': 'lw_out',
     'VPD_PI': 'vpd',
     'T_SONIC': 't_avg',
     'WS': 'ws',
     'SWC_1_1_1': 'theta_1',
     'SWC_1_2_1': 'theta_2'}
```

Tip: The `Data.inv_map` is mainly used to rename the dataframe to internal names, this can be very useful if you are creating your own custom workflows using the `flux-data-qaqc` API because it allows you to only know the internal names of variables therefore they can be hard coded into your workflow and applied to different eddy covariance datasets. For example, let's say we wanted to make HTML tables of basic statistics of just the energy balance components for many datasets (that may have different names for the same variables) and save the file using the user-defined names:

```
>>> d = Data('US-Tw3_config.ini')
>>> df = d.df.rename(columns=d.inv_map)
>>> # get some metadata for saving
>>> site_id = d.site_id
>>> vars_we_want = ['H', 'LE', 'Rn', 'G']
>>> # rename variables, calculate basice statistics table and save to HTML
>>> df[vars_we_want].rename(columns=d.variables).describe().to_html('{}.html'.
↪format(site_id))
```

(continues on next page)

(continued from previous page)

```
    Calculating mean for var: THETA from columns: ['SWC_1_1_1', 'SWC_1_2_1']
    WARNING: renaming column G to input_G
>>> # which produces the following HTML table with user-defined names:
>>> from IPython.display import HTML
>>> HTML(filename='{}.html'.format(site_id))
```

Another powerful feature of the `Data.df` property is that it is datetime-indexed using the input data's temporal frequency, view the date index like so:

```
>>> d.df.index
    DatetimeIndex(['2013-01-01 00:00:00', '2013-01-01 00:30:00',
                   '2013-01-01 01:00:00', '2013-01-01 01:30:00',
                   '2013-01-01 02:00:00', '2013-01-01 02:30:00',
                   '2013-01-01 03:00:00', '2013-01-01 03:30:00',
                   '2013-01-01 04:00:00', '2013-01-01 04:30:00',
                   ...
                   '2018-06-04 19:00:00', '2018-06-04 19:30:00',
                   '2018-06-04 20:00:00', '2018-06-04 20:30:00',
                   '2018-06-04 21:00:00', '2018-06-04 21:30:00',
                   '2018-06-04 22:00:00', '2018-06-04 22:30:00',
                   '2018-06-04 23:00:00', '2018-06-04 23:30:00'],
                  dtype='datetime64[ns]', name='date', length=95088, freq=None)
```

Datetime-indexed `pandas.DataFrame` objects have useful features for time series analysis like grouping and calculating statistics by time aggregates. The example below shows how to calculate the day of year mean for energy balance components, it also demonstrates how to use the `add_lines` plotting method available to `Data`, `QaQc`, and `Plot` objects.

```
>>> # convert to internal names, copy dataframe
>>> df = d.df.rename(columns=d.inv_map)
>>> # day of year mean of input energy balance components
>>> vars_we_want = ['H', 'LE', 'Rn', 'G']
>>> doy_means = df[vars_we_want].groupby(d.df.index.dayofyear).mean()
>>> # create a Bokeh figure
>>> fig = figure(x_axis_label='day of year', y_axis_label='day of year mean (w/m2)')
>>> # arguements needed for creating interactive plots
>>> plt_vars = vars_we_want
>>> colors = ['red', 'blue', 'black', 'green']
>>> x_name = 'date'
>>> source = ColumnDataSource(doy_means)
>>> Plot.add_lines(fig, doy_means, plt_vars, colors, x_name, source, labels=vars_we_
↪want,
>>>     x_axis_type=None)
>>> show(fig)
```

Note: The `x_axis_type=None` is a unique argument to `Plot.add_lines` and `Plot.line_plot` that in this case means to not try to force the x-axis format to a datetime representation, default is `x_axis_type='date'`.

See also:

Some routines occur automatically when creating a `Data` object, including calcuation of weighted and non-weighted averages of soil heat flux and soil moisture which is described in *Averaging data from multiple sensors*.

3.2.4 Modifying input data

A last note on the *Data* object (same goes for the *QaQc* object) is that `Data.df` is a class property, in this case that means that it can be reassigned with a different `pandas.DataFrame`. This is critical for manual pre-filtering and validation of data before proceeding with energy balance closure routines. A simple example is shown here:

```
>>> # add 5 to air temperature, this would effect ET calculations later
>>> x = d.df
>>> x['T_SONIC'] += 5
>>> d.df = x
```

A realistic use of the reassignability of the *Data.df* and *QaQc.df* properties is shown in manual cleaning of poor quality data.

See also:

The *Data.apply_qc_flags* method allows for reading in quality control flags with the input data and filtering specific data out based on user-defined numeric or character flags. This routine is specific to *Data* and includes several attributes that are added to a *Data* instance, for full explanation and examples see *Quality-based data filtering*.

3.3 Visualize input data

The *Data.plot* method create a series of interactive time series plots of input data, potential plots inlcude:

- energy balance components
- radiation components
- multiple soil heat flux measurements
- air temperature
- vapor pressure and vapor pressure deficit
- wind speed
- precipitation
- latent energy
- multiple soil moisture measurements

If any of these variables are not found the plot(s) will not be added.

The most useful interactive features of plots created by `flux-data-qaqc` are:

- pan/zoom
- hover tolltips on var names, values, date
- linked x-axes on time series plots
- save plot option (can save specific subplot zoomed in)

Here is an example,

```
>>> d.plot(output_type='notebook', plot_width=700)
```

The output plot is not shown in the online documentation due to memory constraints.

Hint: The plot methods of *Data* and *QaQc* objects have the keyword argument `output_type` which by default is set to "save", the other two options are "notebook" for showing within a Jupyter Notebook and "show" which opens a temporary file in the default web browser.

If you rather save the plot, and maybe you want 2 columns of plots,

```
>>> d.plot(ncols=2, plot_width=500)
```

After saving a plot without specifying the output file path (keyword argument `out_file`), it will be saved to an "output" directory where the config file is with the file name based on *Data.site_id* with the suffix "_input_plots":

```
>>> # where the plot file was saved by default
>>> d.plot_file
    PosixPath('/home/john/flux-data-qaqc/examples/Basic_usage/output/US-Tw3_input_
↪plots.html')
```

The following plot is not shown due to excessive memory usage needed to build online documentation.

```
>>> # view outplot plots within Jupyter notebook
>>> from IPython.display import HTML
>>> HTML(filename=d.plot_file)
```

Hint: The *QaQc.plot* method shown below is similar however it may include added plots with calculated and corrected variables (if they exist) and will always plot data in daily and monthly temporal frequency because daily frequency is required before applying `flux-data-qaqc` energy balance closure corrections.

3.4 Temporal resampling

The *QaQc* object holds several tools for managing data and eddy covariance data analysis, but one of it's primary features is temporal resampling of input data to daily and monthly frequencies. The resampling of time series data to daily frequency occurs upon the creation of a *QaQc* instance if the frequency within the preceeding *Data* object is not already daily:

```
>>> # the frequency of the input data is 30 minute
>>> d.df.index[0:5]
    DatetimeIndex(['2013-01-01 00:00:00', '2013-01-01 00:30:00',
                   '2013-01-01 01:00:00', '2013-01-01 01:30:00',
                   '2013-01-01 02:00:00'],
                  dtype='datetime64[ns]', name='date', freq=None)
```

```
>>> # creating a QaQc instance will automatically convert to daily
>>> q = QaQc(d)
    The input data temporal frequency appears to be less than daily.
    Data is being resampled to daily temporal frequency.
    Filtering days with less then 100.0% or 48/48 sub-daily measurements
    Converting vpd from hpa to kpa
```

```
>>> # first 5 datetime indices are dates now
>>> q.df.index[0:5]
    DatetimeIndex(['2013-01-01', '2013-01-02', '2013-01-03', '2013-01-04',
```

(continues on next page)

(continued from previous page)

```
                    '2013-01-05'],
                  dtype='datetime64[ns]', name='date', freq=None)
```

The method used for aggregating different variables, e.g. mean or sum, when resampling to daily or monthly frequency is defined in the `QaQc.agg_dict` class attribute:

```
>>> # these are the internal names as keys and temporal aggregation method as values
>>> QaQc.agg_dict
    {'energy': 'mean',
     'flux': 'mean',
     'flux_corr': 'mean',
     'br': 'mean',
     'ET': 'sum',
     'ET_corr': 'sum',
     'ET_gap': 'sum',
     'ET_fill': 'sum',
     'ET_fill_val': 'sum',
     'ET_user_corr': 'sum',
     'ebr': 'mean',
     'ebr_corr': 'mean',
     'ebr_user_corr': 'mean',
     'ebr_5day_clim': 'mean',
     'gridMET_ETr': 'sum',
     'gridMET_prcp': 'sum',
     'lw_in': 'mean',
     't_avg': 'mean',
     'rso': 'mean',
     'sw_pot': 'mean',
     'sw_in': 'mean',
     'vp': 'mean',
     'vpd': 'mean',
     'ppt': 'sum',
     'ws': 'mean',
     'Rn': 'mean',
     'sw_out': 'mean',
     'lw_out': 'mean',
     'G': 'mean',
     'LE': 'mean',
     'LE_corr': 'mean',
     'LE_user_corr': 'mean',
     'H': 'mean',
     'H_corr': 'mean',
     'H_user_corr': 'mean'}
```

Note: There are several calculated variables above that may not look familiar, many are calculated by the energy balance closure correction routines and described in *Closure Methodologies*. Also, any other variables (not found in *`QaQc.agg_dict`* that exist in a *`QaQc.df`* before accessing *`QaQc.monthly_df`* the first time will be averaged in the monthly time series dataframe (*`QaQc.monthly_df`*).

The *`QaQc`* constructor tries to infer the temporal frequency of the input time series data, however the method is not always accurate, to access the inferred initial temporal frequency of the data view the *`QaQc.temporal_freq`* attribute:

```
>>> q.temporal_freq
    '30T'
```

If the inferred input frequency was accurate you will see a Pandas datetime alias, in this case '30T' is thirty minutes. If the temporal frequency is not automatically detected you should be able to rely on the `n_samples_per_day` instance attribute that is manually estimated by the `QaQc` constructor:

```
>>> q.n_samples_per_day
    48
```

3.4.1 Filter days with sub-daily gaps

The `drop_gaps` and `daily_frac` keyword arguments used when creating a *QaQc* instance allow you to control how days with sub-daily measurement gaps will or will not be filtered out when resampling to daily frequency.

Sub-daily gaps in energy balance variables LE, H, Rn, and G , and daily ASCE standaridized reference ET inputs, e.g. hourly ea ("vp"), rs ("sw_in"), t_min, t_max, and ws, can be linearly interpolated automatically before daily aggregations. Interpolation is performed over gap lengths measured in hours, with options to control the longest length of gap to interpolate when $Rn \geq 0$ controlled by the *QaQc* keyword argument `max_interp_hours` (default 2 hours) and the longest gap to interpolate when $Rn < 0$ set by the `max_interp_hours_night` (default 4 hours).:math:‘

Important: By default the *QaQc* constructor will first linearly interpolate energy balance and ASCE ref. ET variables (LE, H, Rn, G, ea ("vp"), rs ("sw_in"), t_min, t_max, and ws) according to the maximum gap lengths (`max_interp_hours` and `max_interp_hours_night`) and then count sub-daily gaps and drop days (set values to null) for all climate data columns (not QC flag or sub-daily gap count columns) where any of the sub-daily data are missing because by default `drop_gaps=True` and `daily_frac=1.0`. In other words, if you have hourly input data for(LE and one hour was missing on a given day, by default that hour will be linearly interpolated before calculating the daily time series and the daily mean will be calculated after. On the other hand, if other climate variables had a single hour missing on a given day, e.g. wind direction, this day would be filtered out by the *QaQc* constructor. This is important because the daily time series is what is used in all energy balance closure correction and daily ASCE standardized reference ET algorithms.

The percentage of sub-daily samples to require set by the `daily_frac` argument and the maximum length of gaps to linearly interpolate set by `max_interp_hours` and `max_interp_hours_night` complement each other and are used in tandem. For example, if the input data is half-hourly and you only want a maximum of 4 hours to be interpolated on any given day and gap lengths to interpolate should be no more than 2 hours each then you would pass the following parameters to the *QaQc* constructor:

```
>>> q = QaQc(d, daily_frac=20/24, max_interp_hours=2, max_interp_hours_night=2)
    The input data temporal frequency appears to be less than daily.
    Data is being resampled to daily temporal frequency.
    Linearly interpolating gaps in energy balance components up to 2 hours when Rn <␣
↪0 and up to 2 hours when Rn >= 0.
    Filtering days with less then 83.33333333333334% or 40/48 sub-daily measurements
```

In this case we set `daily_frac=20/24` because we are only allowing a maximum of 4 hours of total gaps in the day in other words we are requiring 40 of the 48 half hourly samples to exist before we filter out a day. Remember, because linear interpolation of gaps is done before counting sub-daily gaps, this could result in retaining days with more than 4 hours of gaps in the original time series of energy balance components. You may also pass the `daily_frac` arugment as a decimal fraction, e.g. $0.8333 \approx 20/24$.

To not drop any days and take daily means/sums based on whatever data exists in a given day *without* any interpolation of energy balance variables,

```
>>> q = QaQc(d, drop_gaps=False, max_interp_hours=None)
    The input data temporal frequency appears to be less than daily.
    Data is being resampled to daily temporal frequency.
```

Let's view a comparison of Rn using different options of filtering days with sub-daily gaps in the working dataset, because it has several periods of systematic gaps which cause upwards skewing of daily mean Rn if not filtered carefully:

```
>>> # make an empty pandas dataframe for Rn series
>>> Rn_df = pd.DataFrame()
>>> # recreate multiplt QaQc instances using different sub-day gap filters
>>> q = QaQc(d, drop_gaps=False, max_interp_hours=None)
>>> Rn_df['sub_day_gaps'] = q.df.Rn_subday_gaps
>>> Rn_df['no_filter_no_interp'] = q.df.rename(columns=q.inv_map).Rn
>>> q = QaQc(d, drop_gaps=False)
>>> Rn_df['no_filter_with_interp'] = q.df.rename(columns=q.inv_map).Rn
>>> q = QaQc(d, daily_frac=0.5) # filter days with less than 50% data
>>> Rn_df['require_50'] = q.df.rename(columns=q.inv_map).Rn
>>> q = QaQc(d, daily_frac=0.75)
>>> Rn_df['require_75'] = q.df.rename(columns=q.inv_map).Rn
>>> q = QaQc(d, daily_frac=1, max_interp_hours=24, max_interp_hours_night=24)
>>> Rn_df['require_100_with_interp'] = q.df.rename(columns=q.inv_map).Rn
>>> q = QaQc(d, daily_frac=1, max_interp_hours=None)
>>> Rn_df['require_100_no_interp'] = q.df.rename(columns=q.inv_map).Rn
>>> # plot to compare results of day-gap filter
>>> fig = figure(x_axis_label='date', y_axis_label='mean daily net radiation (w/m2),␣
↪filtered based on sub-daily gaps')
>>> # arguments needed for creating interactive line plots
>>> colors = ['red', 'darkred','orange', 'blue', 'black', 'tan']
>>> plt_vars = ['no_filter_no_interp', 'no_filter_with_interp', 'require_50',
↪'require_75', 'require_100_with_interp', 'require_100_no_interp']
>>> labels = ['no filter wout/interp.', 'no filter w/interp.', 'require > 50% w/
↪interp.', 'require > 75% w/interp.', 'require 100% w/interp.', 'require 100% wout/
↪interp.']
>>> x_name = 'date'
>>> source = ColumnDataSource(Rn_df)
>>> Plot.add_lines(fig, Rn_df, plt_vars, colors, x_name, source, labels=labels)
>>> # add daily gap counts to secondary y
>>> fig.extra_y_ranges['gap_counts'] = Range1d(start=0, end=48)
>>> fig.add_layout(LinearAxis(y_range_name='gap_counts', axis_label='number of sub-
↪daily gaps'), 'right')
>>> fig.circle('date', 'sub_day_gaps', legend='n sub-day gaps', y_range_name='gap_
↪counts',
>>>     color='silver', source=source
>>> )
>>> fig.hover[0].tooltips.append(('sub_day_gaps','@{}'.format('sub_day_gaps')))
>>> fig.legend.location = 'top_right'
>>> show(fig)
```

Try zooming in on the gaps filled by the "no filter wout/interp." line to compare which days are retained/filtered by different options, also remove lines by clicking on them in the legend to compare subsets of options.

Tip: For a more fine-grained approach to filtering out days where perhaps multiple 2 hour gaps were filled use the newly created daily gap count columns: "LE_subday_gaps", "H_subday_gaps", "Rn_subday_gaps", and "G_subday_gaps":

```
>>> q = QaQc(d)
>>> df = q.df.rename(columns=q.inv_map)
```

For example, you could post-filter out days in any given energy balance variable, in this case Rn where sub-daily gaps exceed a threshold:

```
>>> df.loc[(df.Rn_subday_gaps > 4) & (df.Rn.notna()), ['Rn','Rn_subday_gaps
↪']]
```

3.4.2 Monthly time series

The `QaQc.monthly_df` property allows for creating the monthly time series of input anc calculated variables provided by `QaQc.correct_data`. It uses the same temporal aggregation methods as the daily time series i.e. from `QaQc.agg_dict`. Although there are many similarities there are important differences between `QaQc.df` and `QaQc.monthly_df` other than the obvious: when accessing the `QaQc.monthly_df` it will automatically run the default energy balance closure correction routine provided by `QaQc.correct_data` *if* it has not yet been run. You can check if it has been run at anytime by:

```
>>> q.corrected
    False
```

To show how this works let's access the monthly data and show the monthly statistics of the "corrected" evapotranspiration (ET_corr):

```
>>> # first note, ET_corr is not in the dataset yet
>>> 'ET_corr' in q.df.columns
    False
```

Now access the monthly time series,

```
>>> q.monthly_df;
>>> 'ET_corr' in q.df.columns
    True
```

By calling the monthly dataframe, the energy balance closure was applied automatically

```
>>> q.monthly_df.ET_corr.describe()
    count     61.000000
    mean      87.858135
    std       49.938287
    min       11.370062
    25%       41.418994
    50%       84.383190
    75%      127.500125
    max      192.033481
    Name: ET_corr, dtype: float64
```

```
>>> q.corrected
    True
```

Note: The `QaQc.monthly_df` also filters out months with less than 30% of days of the month missing by default. To calculate monthly time series with other threshold fractions of days required use the `util.monthly_resample` function and adjust the keyword argument `thresh` which is the fraction (0-1) of days of the month required to not be gaps otherwise the month's value will be forced to null, e.g. if you wanted to caclulate the monthly mean air temperature requiring 30 and 90 percent of the days in the month to not be gaps:

```
>>> from fluxdataqaqc.util import monthly_resample
>>> # select just t_avg for example
>>> cols = ['t_avg']
>>> df = q.df.rename(columns=q.inv_map)
>>> # create temporary df with different monthly resample results
>>> tmp_df = monthly_resample(df, cols, 'mean', thresh=0.9).rename(
>>>     columns={'t_avg': 'thresh_90'}
>>> )
>>> # join temp dataframe with monthly resample results using different thresh
>>> monthly_gap_comp = tmp_df.join(monthly_resample(df, cols, 'mean', thresh=0.3).
↪rename(
>>>     columns={'t_avg': 'thresh_30'})
>>> )
>>> # plot to compare results of day-gap filter
>>> fig = figure(x_axis_label='date', y_axis_label='monthy mean air temperature (C),␣
↪filtered based on daily gaps')
>>> # arguments needed for creating interactive line plots
>>> x = 'date'
>>> source = ColumnDataSource(monthly_gap_comp)
>>> # this example also shows how to use other Bokeh plot arguments
>>> Plot.line_plot(fig,'date','thresh_30',source,'red',label='require > 30%', line_
↪alpha=0.5)
>>> Plot.line_plot(fig,'date','thresh_90',source,'black',label='require > 90%',line_
↪dash='dotted', line_width=2)
>>> fig.legend.location = 'top_right'
>>> show(fig)
```

3.5 Energy balance corrections

`flux-data-qaqc` provides routines that adjust surface energy balance fluxes to improve energy balance closure of eddy covariance flux station data. These routines ultimately result in a corrected daily and monthly time series of latent energy, sensible heat, and evapotranspiration with the option to gap-fill days in corrected ET with ET calculated from gridMET reference ET and fraction of reference ET.

There are two methods currently implemented:

- Energy Balance Ratio method (default), modified from the FLUXNET method
- Bowen Ratio approach (forces closure)
- Multiple least squares regression - user defines LHS and RHS from LE, H, Rn, and G,

Detailed descriptions of methods including daily ET gap-filling methods can be found in the online documentation *Closure Methodologies* page. A few important notes on the API of these methods and other hydro-climatic statistical variables that are calculated are shown below.

The *`QaQc.correct_data`* method is used to run energy balance closure corrections. Here are a few tips on using them,

```
>>> # note above the monthly_df automatically applied the 'ebr' Energy Balance Ratio␣
↪correction
>>> q.corr_meth
```

(continues on next page)

(continued from previous page)

```
    'ebr'
```

```
>>> # potential correction options
>>> q.corr_methods
    ('ebr', 'br', 'lin_regress')
```

```
>>> # to specify the Bowen Raito method:
>>> q.correct_data(meth='br')
```

```
>>> # the most recently used correction method is now shown
>>> q.corr_meth
    'br'
```

Tip: After applying any energy balance closure correction routine all previous corrected variables will be overwritten or dropped in *QaQc.df*, *QaQc.monthly_df*, and *QaQc.variables*, therefore to make a comparison of different methods on the same data make a copy of the `df` or `monthly_df` properties before running the next correction, e.g.

```
>>> # make copies of daily results of different correction options
>>> q.correct_data(meth='ebr')
>>> ebr_gapfilled = q.df
>>> q.correct_data(meth='ebr', etr_gap_fill=False)
>>> ebr_notgapfilled = q.df
>>> q.correct_data(meth='br')
>>> br_gapfilled = q.df
>>> q.correct_data(meth='br', etr_gap_fill=False)
>>> br_notgapfilled = q.df
```

3.5.1 ET gap-filling

A few notes on the option that uses reference ET and fraction of daily reference ET to fill in large gaps in corrected ET, i.e. the keyword argument `QaQc.correct_data(etr_gap_fill = True)`.

- The nearest gridMET cell's time series data for precipitation and alfalfa reference ET is attempted to be downloaded if it is not found in the `gridmet_file_path` entry of the config.ini file.
- If the path to a gridMET file is not found it is re-downloaded, the config file will be updated with the new path and resaved.
- Only the overlapping time period that matches the eddy covariance time series data is attempted to be downloaded, i.e. the period in `QaQc.df.index`.
- When a gridMET file is downloaded it will always be saved in a subdirectory where the config file is located called "gridMET_data" and named using the `QaQc.site_id` and gridMET cell centroid latitude and longitude.
- Corrected latent energy (LE_{corr}) gaps are also backwards filled from gap-filled ET.

Caution: gridMET only exists within the contiguous United States and from 1979 to present, therefore if your station lies outside of this region or you are analyzing eddy flux data recorded before 1979 this option will not be ususable and you should always run corrections with `etr_gap_fill=False` to avoid potential errors.

The Bowen Ratio correction method will produce the 'br' variable which is the Bowen Ratio.

3.6 Other calculations

By default, *QaQc.correct_data* also calculates ET from input latent energy (LE) and air temperature, corrected ET from corrected LE and air temperature, potential clear sky radiation (ASCE formulation), and the *Data* object attempts to calculate vapor pressure deficit from vapor pressure and air temperature or vapor pressure from vapor pressure deficit and air temperature if they exist at hourly or shorter temporal frequency.

3.6.1 Evapotranspiration

The evapotranspiration (ET) calculations are described in *Steps 7 and 8 correct turbulent fluxes, EBR, and ET* of the Energy Balance Ratio correction explanation.

3.6.2 ASCE clear sky radiation

Daily ASCE potential clear sky radiation (R_{so}) is calculated using equation 19 in the "ASCE Standardized Reference Evapotranspiration Equation" final report by the Task Committee on Standardization of Reference Evapotranspiration Environmental and Water Resources Institute of the American Society of Civil Engineers January, 2005 here. This calculation is a simple method based primarily on elevation and latitude which results in a theoretical envelope of R_{so} as a function of the day of year,

$$R_{so} = \left(5 + 2 \times 10^{-5} z\right) R_a$$

where z is elevation in meters and R_a is daily extraterrestrial radiation (radiation with in the absence of an atmosphere), which itself is a well-behaved function of solar declination, the day of the year and the solar constant (see equations 21-29 in the ASCE report).

3.6.3 Vapor pressure/deficit

The *Data* object will attempt to calculate vapor pressure or vapor pressure deficit if one exists but not the other and average air temperature time series also exists with the input data at hourly or shorter temporal frequency. The Magnus equation (eqn. 37 in the ASCE report) states that the saturation vapor pressure (es) in kPa relates to air temperature,

$$es = 0.6108e^{\left(\frac{17.27 \cdot T}{(T+237.3)}\right)}$$

where T is average hourly air temperature in degrees celcius. Vapor pressure deficit (vpd) is,

$$vpd = es - ea,$$

where ea is actual vapor pressure in kPa. **Note,** The equations above are defined for hourly measurements however they are used for hourly or shorter mean variables (T, ea, or vpd) within `flux-data-qaqc` and then converted to daily means, if they are not present in the input data at hourly or shorter frequencies then they are not calculated.

These equations can be rearanged to solve for either es or vpd given the other variable and air temperature. For example, if given T and vpd, then to get actual vapor pressure

$$es = 0.6108e^{\left(\frac{17.27 \cdot T}{(T+237.3)}\right)}$$

$$ea = es - vpd.$$

In `flux-data-qaqc` actual vapor pressure is named "vp" not "ea". Also, during these calculations, if relative humidity is not found in the input dataset then it will subsequently be estimated as

$$rh = 100 \times \frac{ea}{es}.$$

Hint: The same calculations are available at the daily timestep but are not automatically applied as the hourly or higher temporal frequency calculation is preffered. To apply the estimates of vapor pressure or vapor pressure deficit, and saturation vapor pressure, and relative humidity with daily data one must call the `QaQc._calc_vpd_from_vp` method from a *`QaQc`* instance.

3.6.4 Calculated variable reference

Although variables created by energy balance closure corrections are described in *Closure Methodologies* and others are below, here is a reference list of all possibly calculated variables created by `flux-data-qaqc`:

Variable	Description
rso	potential clear sky radiation (ASCE formulation)
flux	input LE + H
energy	input Rn - G
ebr_5day_clim	5 day climatology of the filtered Energy Balance Ratio
LE_corr	corrected latent energy
ebc_cf	energy balance closure correction factor (inverse of ebr_corr)
ebr_corr	corrected energy balance ratio
flux_corr	LE_corr + H_corr
ebr	input energy balance ratio
br	bowen ratio
H_corr	corrected sensible heat
ET	ET calculated from input LE and average air temperature
ET_corr	ET calculated from LE_corr and avg. air temp.
gridMET_ETr	gridMET alfalfa reference ET (nearest cell)
gridMET_prcp	gridMET precipitation
ETrF	fraction of reference ET for ET_corr, i.e. ET_corr / gridMET_ETr
ETrF_filtered	filtered ETrF
ET_fill	gridMET_ETr * ETrF_filtered (fills gaps in ET_corr)
ET_gap	True on gap days in ET_corr, False otherwise
ET_fill_val	value of ET_fill on gap days

3.7 A note on units

Upon creation of a *`QaQc`* object, variables are checked for valid input units and converted to required units needed for internal calculations when running *`QaQc.correct_data`* and for certain default plots (see below). For a list of valid input units for different variables refer to the `QaQc.allowable_units` attribute:

```
>>> q.allowable_units
    {'LE': ['w/m2'],
     'H': ['w/m2'],
     'Rn': ['w/m2'],
     'G': ['w/m2'],
```

(continues on next page)

(continued from previous page)

```
     'lw_in': ['w/m2'],
     'lw_out': ['w/m2'],
     'sw_in': ['w/m2'],
     'sw_out': ['w/m2'],
     'ppt': ['mm', 'in'],
     'vp': ['kpa', 'hpa'],
     'vpd': ['kpa', 'hpa'],
     't_avg': ['c', 'f']}
```

For each variable above, if given one of the units allowable the units will automatically be converted to the required units.

To know which variables are required to be in particular units view `Qc.required_units`:

```
>>> q.required_units
    {'LE': 'w/m2',
     'H': 'w/m2',
     'Rn': 'w/m2',
     'G': 'w/m2',
     'lw_in': 'w/m2',
     'lw_out': 'w/m2',
     'sw_in': 'w/m2',
     'sw_out': 'w/m2',
     'ppt': 'mm',
     'vp': 'kpa',
     'vpd': 'kpa',
     't_avg': 'c'}
```

Note: The list of allowable units is a work in progress, if your input units are not available consider raising an issue on GitHub or providing the conversion directly with a pull request. Automatic unit conversions are handled within the *`util.Convert`* class using the *`util.Convert.convert`* class method.

3.8 Save resampled and corrected data

The *`QaQc.write`* method conveniently writes daily and monthly time series of input and calculated variables to comma separated value (CSV) files. If the *`QaQc.correct_data`* method has not yet been run it will be and the monthly time series will also be created using the default parameters for the correction routine (Energy Balance Ratio method with ETr-based gap filling).

The default output directory for time series files can be accessed/changed by the `out_dir` attribute, if not changed it will be located in the same directory of the config.ini file. The daily and monthly time series file names will begin with the *`QaQc.site_id`* followed by "daily_data" or "monthly_data" resepctively. For example,

```
>>> # new QaQc instance
>>> q = QaQc(d)
>>> # a platform dependent pathlib.Path object
>>> q.out_dir
    PosixPath('/home/john/flux-data-qaqc/examples/Basic_usage/output')
```

The line below shows that no output files have been written to *`QaQc.out_dir`* yet,

```
>>> # print files in output directory that begin with the site_id
>>> [f.name for f in q.out_dir.glob('{}*'.format(q.site_id))]
    ['US-Tw3_input_plots.html']
```

```
>>> q.corrected
    False
```

```
>>> # writing files also ran corrections since they were not yet run
>>> q.write()
>>> q.corrected
    True
```

Now the respective daily and monthly time series have been written to `QaQc.out_dir`,

```
>>> [f.name for f in q.out_dir.glob('{}*'.format(q.site_id))]
    ['US-Tw3_daily_data.csv', 'US-Tw3_monthly_data.csv', 'US-Tw3_input_plots.html']
```

Hint: You can overwrite the default name of the output directory to save the daily and monthly time series using the `out_dir` keyword argument to `QaQc.write`, this option keeps the location within the directory of the config file but just changes the name, whereas to change the entire output directory path adjust the `QaQc.out_dir` attribute directly. Also, the naming scheme of output files created will use user-defined names for all input variables.

3.9 Visualize resampled and corrected data

Similar to the `Data.plot`, the `QaQc.plot` method creates a series of default time series and scatter plots of input and in this case calculated variables. The temporal frequency of plots from `QaQc.plot` will always be daily and monthly and additional plots are created for validation of energy balance closure corrections, otherwise the same options such as number of subplot columns, super title, subplot dimensions, output type, output file path, etc. are available. Similar to `QaQc.write` and `QaQc.monthly_df`, if the data has not yet been corrected the `plot` method will correct it using the default parameters before creating the plots.

Here is an example of the default daily and monthly time series plots produced after running the Energy Balance Ratio closure correction:

```
>>> q = QaQc(d)
>>> q.plot(output_type='notebook', plot_width=700)
```

CHAPTER 4

Closure Methodologies

flux-data-qaqc currently provides two routines which ultimately adjust turbulent fluxes in order to improve energy balance closure of eddy covariance tower data, the Energy Balance Ratio and the Bowen Ratio method.

Closure methods are assigned as keyword arguments to the *QaQc.correct_data* method, and for a list of provided closure options see *QaQc.corr_methods*. For example, if you would like to run the Bowen Ratio correction routine assuming you have succesfully created a *QaQc* object,

```
# q is a QaQc instance
q.correct_data(meth='br')
```

The other keyword argument for *QaQc.correct_data* allows for gap filling corrected evapotranspiration (ET) which is calculated from corrected latent energy (LE). By default the gap filling option is set to True, more details on this below in *Step 9, optionally gap fill corrected ET using gridMET reference ET and reference ET fraction*.

Tip: All interactive visualizations in this page were created using *Plot.line_plot*, *Plot.add_lines*, and *Plot.scatter_plot* which automatically handle issues with utilizing the mouse hover tooltips and other bokeh.plotting.figure.Figure features.

4.1 Data description

The data for this example comes from the "Twitchell Alfalfa" AmeriFlux eddy covariance flux tower site in California. The site is located in alfalfa fields and exhibits a mild Mediterranean climate with dry and hot summers, for more information on this site or to download data click here.

4.2 Energy Balance Ratio method

The Energy Balance Ratio method (default) is modified from the FLUXNET methodology (step 3 daily heat processing). The method involves filtering out of extreme values of the daily Energy Balance Ratio time series, smoothing, and gap filling. Then the inverse of the filtered and smoothed time series is used as a series of correction factors for the initial time series of latent energy (LE) and sensible heat (H) flux time series.

4.2.1 All steps, abbreviated

Below is a step-by-step description of the Energy Balance Ratio correction routine used by `flux-data-qaqc`. More details and visual demonstration of steps are shown below.

Step 0 (optional): optionally filter out poor quality data first if quality control (QC) values or flags are provided with the dataset or other means. For example, FLUXNET data includes QC values for H and LE, e.g. H_F_MDS_QC and LE_F_MDS_QC are QC values for gap filled H and LE. This allows for manual pre-QaQc of data.

Step 1: calculate the Energy Balance Ratio (EBR = $\frac{H+LE}{Rn\breve{}G}$) daily time series from raw data.

Step 2: filter EBR values that are outside 1.5 times the interquartile range.

Step 3: for each day in the daily time series of filtered EBR, a sliding window of +/- 7 days (15 days) is used to select up to 15 values.

Step 4: for each day take a percentile (default 50) of the 15 EBR values. Check if the inverse of the EBR value is $> |2|$ or if the the inverse of the ratio multiplied by the measured LE would result in a flux greater than 850 or less than -100 w/m^2, if so leave a gap for filling later.

Step 5: if less than +/- 5 days exist in the sliding 15 day window, use the mean EBR for all days in a +/- 5 day (11 day) sliding window. Apply same criteria for an extreme EBR value as in step 4.

Step 6: if no EBR data exist in the +/- 5 sliding window to average, fill remaining gaps of EBR with the mean from a +/- 5 day sliding window over the day of year mean for all years on record, i.e. 5 day climatology. Calculate the 5 day climatology from the filtered and smoothed EBR as produced from step 5. Apply same criteria for an extreme EBR value as in steps 4 and 5.

Step 7: use the filtered EBR time series from previous steps to correct LE and H by multiplying by the energy balance closure correction factor $EBC_{CF} = \frac{1}{EBR}$, where EBR has been filtered by the previous steps. Use the corrected LE and H to calculate the corrected EBR.

Step 8: calculate corrected ET from corrected LE using average air temperature to adjust the latent heat of vaporization.

Step 9 (optional): if desired, fill remaining gaps in the corrected ET time series with ET that is calculated by gridMET reference ET (ETr or ETo) multiplied by the filtered and smoothed fraction of reference ET ($ETrF$ or $EToF$).

4.2.2 Step 0, manual cleaning of poor quality data

Below we can see that the daily time series of net radiation (Rn) has some periods of poor quality data. This is a common issue due, e.g. to instrumentation problems, that cannot always be avoided. In this case the sensor did not record values at night (or they were not provided with the data) when Rn values are lower for several days (e.g. around 8/26/2014) which resulted in overestimates of daily mean Rn during these periods. Although these days can automatically be filtered out by the `QaQc` class, the example below shows a way of manually filtering them because in other cases outliers in the daily data may not be caused by resampling of sub-daily data with systematic measurement gaps. The main point is that manual inspection and potentially pre-filtering of poor quality data before proceeding with energy balance closure corrections is often necessary.

There are several ways to conduct manual pre-filtering of poor quality meterological time series data, to filter data based on input quality flags or numeric quality values see *Quality-based data filtering*.

`flux-data-qaqc` also allows for filtering of poor quality data on the fly as shown in this example. In other words, we simply filter out the periods we think have bad data for Rn within Python before running the closure correction.

After manually determing the date periods with poor quality Rn, here is how they were filtered oiut before running the correction:

```
>>> import pandas as pd
>>> import numpy as np
>>> from fluxdataqaqc import Data, QaQc
>>> d = Data('Path/to/config.ini')
>>> # days with sub daily gaps can be filtered out automatically here,
>>> # see "Tip" below the following plot
>>> q = QaQc(d, drop_gaps=False)
>>> # rename dataframe columns for ease of variable access, adjust
>>> df = q.df.rename(columns=q.inv_map)
```

Here were the dates chosen and one way to filter them,

```
>>> # make a QC flag column for Rn
>>> df['Rn_qc'] = 'good'
>>> df.loc[pd.date_range('2/10/2014','2/10/2014'), 'Rn_qc'] = 'bad'
>>> df.loc[pd.date_range('8/25/2014','9/18/2014'), 'Rn_qc'] = 'bad'
>>> df.loc[pd.date_range('10/21/2015','10/26/2015'), 'Rn_qc'] = 'bad'
>>> df.loc[pd.date_range('10/28/2015','11/1/2015'), 'Rn_qc'] = 'bad'
>>> df.loc[pd.date_range('7/23/2016','7/23/2016'), 'Rn_qc'] = 'bad'
>>> df.loc[pd.date_range('9/22/2016','9/22/2016'), 'Rn_qc'] = 'bad'
>>> df.loc[pd.date_range('3/3/2017','3/3/2017'), 'Rn_qc'] = 'bad'
>>> # filter (make null) based on our QC flag column for Rn
>>> df.loc[df.Rn_qc == 'bad', 'Rn'] = np.nan
>>> # reassign to use pre-filtered data for corrections
>>> q.df = df
```

The resulting energy balance component plot with Rn filtered:

Tip: In this case, the issues with Rn were caused by resampling 30 minute data with systematic night-time gaps. These sort of issues can be automatically handled when creating a *QaQc* object; the keyword arguments `drop_gaps` and `daily_frac` to the *QaQc* class are used to automatically filter out days with measurement gaps of varying size, i.e.,

```
>>> d = Data('path/to/config.ini')
>>> q = QaQc(d, drop_gaps=True, daily_frac=0.8)
>>> q.correct_data()
```

This would produce very similar energy balance closure results as the manual filter above. Another more fine-grained option would have been to flag the days with gaps in the sub-daily input time series that you would like to filter by *Data.apply_qc_flags*.

Note: The remaining step-by-step explanation in this page uses the pre-filtered input time series, however results of the energy balance closure correction without pre-filtering outliers of Rn are also shown in plots for the final steps (8 and 9) for comparison. If you now ran:

```
>>> q.df = df
>>> q.correct_data()
>>> q.plot(output_type='show')
```

This will directly produce the same output of step 9 using the pre-filtered data.

4.2.3 Steps 1 and 2, filtering outliers of EBR

Calculate daily EBR = $\frac{H+LE}{Rn-G}$ time series and filter out extreme values that are outside 1.5 the interquartile range. Note, in `flux-data-qaqc` this is named as "ebr".

4.2.4 Steps 3, 4, and 5, further filtering of EBR using moving window statistics

Filter the EBR time series using statistics performed over multiple moving windows. Specifically, take the median EBR from a +/- 7 day moving window, if less than 11 days exist in this window take the mean from a +/- 5 day moving window. In both of these cases check the resulting value before retaining based on the following criteria:

- the inverse of the EBR value must be $> |2|$
- the the inverse of the ratio multiplied by the measured LE should result in a flux less than 850 and greater than -100 w/m^2

If either of these criteria are not met leave a gap for the day for filling in later steps.

4.2.5 Step 6, calculate the 5 day climatology of EBR

Compute the 5 day climatology of daily EBR (as adjusted from previous steps) to fill in remaining gaps of 11 or more days. Specifically, calculate the the day of year mean of the EBR for all years in record and then extract the day of year mean using a moving +/- 5 day (11 day) moving window. The resulting value is also checked against the same criteria described in steps 3-5:

- the inverse of the EBR value must be $> |2|$
- the the inverse of the ratio multiplied by the measured LE should result in a flux less than 850 and greater than -100 w/m^2

Note, this step is only used for remaining gaps which should be larger than 11 days in the EBR time series following step 5. This example has a few time periods that were filled with the 5 day climatology of EBR which can be seen as the thin blue line in the plot below.

`flux-data-qaqc` also keeps a record of the 5 day climatology of the Energy Balance Ratio as calculated at this step (shown below), it is named by `flux-data-qaqc` as ebr_5day_clim.

4.2.6 Steps 7 and 8 correct turbulent fluxes, EBR, and ET

Calculate corrected LE and H by multiplying by $\frac{1}{EBR}$ where EBR is the filtered EBR time series from previous steps:

$$LE_{corr} = LE \times \frac{1}{EBR}$$

and

$$H_{corr} = H \times \frac{1}{EBR}.$$

Use corrected LE and H to calculate the corrected EBR,

$$EBR_{corr} = \frac{H_{corr} + LE_{corr}}{Rn - G}.$$

Calculate ET from LE using average air temperature to adjust the latent heat of vaporization following the method of Harrison, L.P. 1963,

$$ET_{mm \cdot day^{-1}} = 86400_{sec \cdot day^{-1}} \times \frac{LE_{w \cdot m^{-2}}}{2501000_{MJ \cdot kg^{-1}} - (2361 \cdot T_C)},$$

where evapotransipiration (ET) in $mm \cdot day^{-1}$, LE is latent energy flux in $w \cdot m^{-2}$, and T is air temperature in degrees celcius. The same approach is used to calculate corrected ET (ET_{corr}) using LE_{corr}.

The plot below shows the time series of the initial and corrected ET (ET and ET_{corr}).

There were not significant gaps in the energy balance components for this dataset and therefore step 9 was not used, although it is still demonstrated with an artificial gap in the next step.

The following plot shows the energy balance closure of the initial and corrected data after applying the steps above, including the manual pre-filtering of Rn,

Notice the mean daily corrected energy balance ratio (slope of corrected) is 1 or near perfect closure. However, the same plot below shows the results if we skipped the manual pre-filtering of outlier Rn values. In this case the resulting corrected mean closure is only 0.93:

Tip: These and other interactive visualizations of energy balance closure results are provided by default via the `QaQc.plot` method.

In `flux-data-qaqc` new variable names from these steps are: LE_corr, H_corr, ebr, ebr_corr, ebc_cf, ET, ET_corr, ebr_corr, and ebr_5day_clim. The inverse of the corrected EBR (filtered from previous steps) is named ebc_cf which is short for energy balance closure correction factor as described by the FLUXNET methodology (step 3 daily heat processing).

4.2.7 Step 9, optionally gap fill corrected ET using gridMET reference ET and reference ET fraction

This is done by downloading ETr or ETo (default is ETr) for the overlapping gridMET cell (site must be in CONUS) and then calculating,

$$ET_{fill} = ETrF \times ET_r,$$

where

$$ETrF = \frac{ET_{corr}}{ET_r}$$

ET_{corr} is the corrected ET produced by step 8 and $ETrF$ is the fraction of reference ET. $ETrF$ if first filtered to remove outliers outside of 1.5 times the interquartile range, it is then smoothed with a 7 day moving average (minimum of 2 days must exist in window) and lastly it is linearly interpolated to fill any remaining gaps.

The same gap filling procedure can easily be done using gridMET grass reference ET (ETo) as opposed to alfalfa reference ET (ETr).

Tip: The filtered and raw versions of $ETrF$/$EToF$, gridMET ETr, gridMET ETo, gap days, and monthly total number of gap filled days are tracked for post-processing and visualized by the `QaQc.plot` and `QaQc.write` methods.

Since the data used in this example does not have gaps, for illustration we have created the following large gap in the measured energy balance components from May through August, 2014:

The resulting time series of ET_{corr} using the optional gap filling method described is shown below.

Note, the gap filled values of ET (green line) do not accurately catch the harvesting cycles of alfalfa however the ET_{corr} values (blue line) do, this is because the gap filled values are based from gridMET reference ET which is not locally representative. If this is hard to see, try using the box zoom tool on the right of the plot to zoom in on the gap-filled period.

This ET gap-filling step is used by default when running `flux-data-qaqc` energy balance closure correction routines, to disable it set the `etr_gap_fill` argument of `QaQc.correct_data` to False, e.g.

```
# q is a QaQc instance
q.correct_data(meth='ebr', etr_gap_fill=False)
```

In `flux-data-qaqc` new variable names from this step are: ETrF, ETrF_filtered, gridMET_ETr, ET_gap, ET_fill, and ET_fill_val. The difference between ET_fill and ET_fill_val is that the latter is masked (null) on days that the fill value was not used to fill gaps in ET_{corr}. Also, ET_gap is a daily series of True and False values indicating which days (from step 8) of ET_{corr} were gaps that were subsequently filled.

Note: When using the ETr-based gap-filling option, any gap filled days will also be used to fill in gaps of LE_{corr}, therefore the mean closure as found in the daily and monthly closure scatter plot outputs (from `QaQc.plot`) will be updated to reflect the influence of the gap-filled days.

4.3 Bowen Ratio method

The Bowen Ratio energy balance closure correction method implemented here follows the typical approach where the corrected latent energy (LE) and sensible heat (H) fluxes are adjusted the following way

$$LE_{corr} = \frac{(Rn - G)}{(1 + \beta)},$$

and

$$H_{corr} = LE_{corr} \times \beta$$

where β is the Bowen Ratio, the ratio of sensible heat flux to latent energy flux,

$$\beta = \frac{H}{LE}.$$

This routine forces energy balance closure for each day in the time series.

Here is the resulting ET_{corr} time series using the pre-filtered (Rn) energy balance time series and the Bowen Ratio method:

And here is the energy balance closure scatter plot which shows the forced closure of the method:

New variables produced by `flux-data-qaqc` by this method include: br (Bowen Ratio), ebr, ebr_corr, LE_corr, H_corr, ET, ET_corr, energy, flux, and flux_corr.

CHAPTER 5

API Reference

This page documents objects and functions provided by `flux-data-qaqc`.

5.1 Data

class `fluxdataqaqc.`**`Data`**`(`*config*`)`

Bases: `fluxdataqaqc.plot.Plot`, *`fluxdataqaqc.util.Convert`*

An object for interfacing `flux-data-qaqc` with input metadata (config) and time series input, it provides methods and attributes for parsing, temporal analysis, visualization, and filtering data.

A *`Data`* object is initialized from a config file (see *Setting up a config file*) with metadata for an eddy covariance tower or other dataset containing time series meterological data. It serves as a starting point in the Python API of the energy balance closure analysis and data validation routines that are provided by `flux-data-qaqc`.

Manual pre-filtering of data based on user-defined quality is aided with the *`Data.apply_qc_flags`* method. Weighted or non-weighted means of variables with multiple sensors/recordings is performed upon initialization if these options are declared in the config file. The `Data` class also includes the *`Data.df`* property which returns the time series data in the form of a `pandas.DataFrame` object for custom workflows. `Data` inherits line and scatter plot methods from *`Plot`* which allows for the creation of interactive visualizations of input time series data.

`climate_file`

Absolute path to climate input file.

Type pathlib.Path

`config`

Config parser instance created from the data within the config.ini file.

Type `configparser.ConfigParser`

`config_file`

Absolute path to config.ini file used for initialization of *`Data`* instance.

Type `pathlib.Path`

header
Header as found in input climate file.

Type `numpy.ndarray` or `pandas.DataFrame.index`

elevation
Site elevation in meters as set in config.ini.

Type float

inv_map
Dictionary with input climate file names as keys and internal names as values. May only include pairs when they differ.

Type dict

latitude
Site latitude in decimal degrees, set in config.

Type float

longitude
Site longitude in decimal degrees, set in config.

Type float

out_dir
Default directory to save output of *`QaQc.write`* or *`QaQc.plot`* methods.

Type pathlib.Path

plot_file
path to plot file once it is created/saved by *`Data.plot`*.

Type pathlib.Path or None

site_id
Site ID as found in site_id config.ini entry.

Type str

soil_var_weight_pairs
Dictionary with names and weights for weighted averaging of soil heat flux or soil moisture variables.

Type dict

qc_var_pairs
Dictionary with variable names as keys and QC value columns (numeric of characters) as values.

Type dict

units
Dictionary with internal variable names as keys and units as found in config as values.

Type dict

variables
Dictionary with internal names for variables as keys and names as found in the input data as values.

Type dict

variable_names_dict
Dictionary with internal variable names as keys and keys in config.ini file as values.

Type dict

xl_parser
engine for reading excel files with Pandas. If `None` use 'openpyxl'.

Type str or None

apply_qc_flags(*threshold=None*, *flag=None*)
Apply user-provided QC values or flags for climate variables to filter poor-quality data by converting them to null values, updates *Data.df*.

Specifically where the QC value is < *threshold* change the variables value for that date-time to null. The other option is to use a column of flags, e.g. 'x' for data values to be filtered out. The threshold value or flag may be specified in the config file's **METADATA** section otherwise they should be assigned as keyword arguments here.

Specification of which QC (flag or numeric threshold) columns should be applied to which variables is set in the **DATA** section of the config file. For datasets with QC value columns with names identical to the variable they correspond to with the suffix "_QC" the QC column names for each variable do not need to be specified in the config file.

Keyword Arguments

- **threshold** (*float*) – default `None`. Threshold for QC values, if flag is below threshold replace that variables value with null.
- **flag** (*str, list, or tuple*) – default `None`. Character flag signifying data to filter out. Can be list or tuple of multiple flags.

Returns `None`

Example

If the input time series file has a column with numeric quality values named "LE_QC" which signify the data quality for latent energy measurements, then in the config.ini file's **DATA** section the following must be specified:

```
[DATA]
latent_heat_flux_qc = LE_QC
...
```

Now you must specify the threshold of this column in which to filter out when using *Data.apply_qc_flags*. For example if you want to remove all data entries of latent energy where the "LE_QC" value is below 5, then the threshold value would be 5. The threshold can either be set in the config file or passed as an argument. If it is set in the config file, i.e.:

```
[METADATA]
qc_threshold = 0.5
```

Then you would cimply call the method and this threshold would be applied to all *QC* columns specified in the config file,

```
>>> from fluxdataqaqc import Data
>>> d = Data('path/to/config.ini')
>>> d.apply_qc_flags()
```

Alternatively, if the threshold is not defined in the config file or if you would like to use a different value then pass it in,

```
>>> d.apply_qc_flags(threshold=2.5)
```

Lastly, this method also can filter out based on a single or list of character flags, e.g. "x" or "bad" gievn that the column containing these is specified in the config file for whichever variable they are to be applied to. For example, if a flag column contains multiple flags signifying different data quality control info and two in particular signify poor quality data, say "b" and "a", then apply them either in the config file:

```
[METADATA]
qc_flag = b,a
```

Of within Python

```
>>> d.apply_qc_flags(flag=['b', 'a'])
```

For more explanation and examples see the "Configuration Options" section of the online documentation.

df

Pull variables out of the config and climate time series files load them into a datetime-indexed `pandas.DataFrame`.

Metadata about input time series file format: "missing_data_value", "skiprows", and "date_parser" are utilized when first loading the `df` into memory. Also, weighted and non-weighted averaging of multiple measurements of the same climatic variable occurs on the first call of *`Data.df`*, if these options are declared in the config file. For more details and example uses of these config options please see the "Configuration Options" section of the online documentation.

Returns df (`pandas.DataFrame`)

Examples

You can utilize the df property as with any `pandas.DataFrame` object. However, if you would like to make changes to the data you must first make a copy, then make the changes and then reassign it to *`Data.df`*, e.g. if you wanted to add 5 degrees to air temp.

```
>>> from fluxdataqaqc import Data
>>> d = Data('path_to_config.ini')
>>> df = d.df.copy()
>>> df['air_temp_column'] = df['air_temp_column'] + 5
>>> d.df = df
```

The functionality shown above allows for user-controlled preprocessing and modification of any time series data in the initial dataset. It also allows for adding new columns but if they are variables used by `flux-data-qaqc` e.g. Rn or other energy balance variables, be sure to also update *`Data.variables`* and *`Data.units`* with the appropriate entries. New or modified values will be used in any further analysis/ploting routines within `flux-data-qaqc`.

By default the names of variables as found within input data are retained in *`QaQc.df`*, however you can use the naming scheme as `flux-data-qaqc` which can be viewed in *`Data.variable_names_dict`* by using the the *`Data.inv_map`* dictionary which maps names from user-defined to internal names (as opposed to *`Data.variables`*) which maps from internal names to user-defined. For example if your input data had the following names for LE, H, Rn, and G set in your config:

```
[DATA]
net_radiation_col = Net radiation, W/m2
ground_flux_col = Soil-heat flux, W/m2
latent_heat_flux_col = Latent-heat flux, W/m2
sensible_heat_flux_col = Sensible-heat flux, W/m2
```

Then the *`Data.df`* will utilize the same names, e.g.

```
>>> # d is a Data instance
>>> d.df.head()
```

produces:

date	Net radiation, W/m2	Latent-heat flux, W/m2	Sensible-heat flux, W/m2	Soil-heat flux, W/m2
10/1/2009 0:00	-54.02421778	0.70761	0.95511	-40.42365926
10/1/2009 0:30	-51.07744708	0.04837	-1.24935	-33.35383253
10/1/2009 1:00	-50.99438925	0.68862	1.91101	-43.17900525
10/1/2009 1:30	-51.35032377	-1.85829	-15.4944	-40.86201497
10/1/2009 2:00	-51.06604228	-1.80485	-19.1357	-39.80936855

Here is how you could rename your dataframe using `flux-data-qaqc` internal names,

```
>>> d.df.rename(columns=q.inv_map).head()
```

date	Rn	LE	H	G
10/1/2009 0:00	-54.02421778	0.70761	0.95511	-40.42365926
10/1/2009 0:30	-51.07744708	0.04837	-1.24935	-33.35383253
10/1/2009 1:00	-50.99438925	0.68862	1.91101	-43.17900525
10/1/2009 1:30	-51.35032377	-1.85829	-15.4944	-40.86201497
10/1/2009 2:00	-51.06604228	-1.80485	-19.1357	-39.80936855

A minor note on variable naming, if your input data variables use exactly the same names used by `flux-data-qaqc`, they will be renamed by adding the prefix "input_", e.g. "G" becomes "input_G" on the first time reading the data from disk, i.e. the first time accessing `Data.df`.

Note: The temporal frequency of the input data is retained unlike the `Qaqc.df` which automatically resamples time series data to daily frequency.

`hourly_ASCE_refET`(*reference='short', anemometer_height=None*)

Calculate hourly ASCE standardized short (ETo) or tall (ETr) reference ET from input data and wind measurement height.

If input data's time frequency is < hourly the input data will be resampled to hourly and the output reference ET time series will be returned as a datetime `pandas.Series` object, if the input data is already hourly then the resulting time series will automatically be merged into the `Data.df` dataframe named "ASCE_ETo" or "ASCE_ETr" respectively.

Keyword Arguments

- **`reference`** (*`str`*) – default "short", calculate tall or short ASCE reference ET.
- **`anemometer_height`** (*`float or None`*) – wind measurement height in meters , default `None`. If `None` then look for the "anemometer_height" entry in the **METADATA** section of the config.ini, if not there then print a warning and use 2 meters.

Returns `None` or `pandas.Series`

Hint: The input variables needed to run this method are: vapor pressure, wind speed, incoming shortwave radiation, and average air temperature. If vapor pressure deficit and average air temperature exist, the actual vapor pressure will automatically be calculated.

`plot` (*ncols=1*, *output_type='save'*, *out_file=None*, *suptitle=''*, *plot_width=1000*, *plot_height=450*, *sizing_mode='scale_both'*, *merge_tools=False*, *link_x=True*, ***kwargs*)

Creates a series of interactive diagnostic line and scatter plots of input data in whichever temporal frequency it is in.

The main interactive features of the plots include: pan, selection and scrol zoom, hover tool that shows paired variable values including date, and linked x-axes that pan/zoom togehter for daily and monthly time series plots.

It is possible to change the format of the output plots including adjusting the dimensions of subplots, defining the number of columns of subplots, setting a super title that accepts HTML, and other options. If variables are not present for plots they will not be created and a warning message will be printed. There are two options for output: open a temporary file for viewing or saving a copy to `QaQc.out_dir`.

A list of all potential time series plots created:

- energy balance components
- radiation components
- multiple soil heat flux measurements
- air temperature
- vapor pressure and vapor pressure deficit
- wind speed
- precipitation
- latent energy
- multiple soil moisture measurements

Keyword Arguments

- **ncols** (*int*) – default 1. Number of columns of subplots.
- **output_type** (*str*) – default "save". How to output plots, "save", "show" in browser, "notebook" for Jupyter Notebooks, "return_figs" to return a list of Bokeh `bokeh.plotting.figure.Figure`s, or "return_grid" to return the :obj:`bokeh.layouts.gridplot`.
- **out_file** (*str or None*) – default `None`. Path to save output file, if `None` save output to `Data.out_dir` with the name [site_id]_input_plots.html where [site_id] is `Data.site_id`.
- **suptitle** (*str or None*) – default `None`. Super title to go above plots, accepts HTML/CSS syntax.
- **plot_width** (*int*) – default 1000. Width of subplots in pixels.
- **plot_height** (*int*) – default 450. Height of subplots in pixels, note for subplots the height will be forced as the same as `plot_width`.
- **sizing_mode** (*str*) – default "scale_both". Bokeh option to scale dimensions of `bokeh.layouts.gridplot`.

- **merge_tools** (*bool*) – default False. Merges all subplots toolbars into a single location if True.
- **link_x** (*bool*) – default True. If True link x axes of daily time series plots and monthly time series plots so that when zooming or panning on one plot they all zoom accordingly, the axes will also be of the same length.

Example

Starting from a correctly formatted config.ini and climate time series file, this example shows how to read in the data and produce a series of plots of input data as it is found in the input data file (unlike `QaQc.plot` which produces plots at daily and monthly temporal frequency). This example also shows how to display a title at the top of plot with the site's location and site ID.

```
>>> from fluxdataqaqc import Data
>>> d = Data('path/to/config.ini')
>>> # create plot title from site ID and location in N. America
>>> title = "<b>Site:</b> {}; <b>Lat:</b> {}N; <b>Long:</b> {}W".format(
>>>     q.site_id, q.latitude, q.longitude
>>> )
>>> q.plot(
>>>     ncols=2, output_type='show', plot_width=500, suptitle=title
>>> )
```

Note, we specified the width of plots to be smaller than default because we want both columns of subplots to be viewable on the screen.

Tip: To reset all subplots at once, refresh the page with your web browser.

Note: Additional keyword arguments that are recognized by `bokeh.layouts.gridplot` are also accepted by `Data.plot`.

See also:

`QaQc.plot`

variable_names_dict = {'G': 'ground_flux_col', 'H': 'sensible_heat_flux_col', 'H_user_

5.2 QaQc

class fluxdataqaqc.**QaQc**(*data=None, drop_gaps=True, daily_frac=1.0, max_interp_hours=2, max_interp_hours_night=4*)

Bases: `fluxdataqaqc.plot.Plot`, `fluxdataqaqc.util.Convert`

Numerical routines for correcting daily energy balance closure for eddy covariance data and other data analysis tools.

Two routines are provided for improving energy balance closure by adjusting turbulent fluxes, latent energy and sensible heat, the Energy Balance Ratio method (modified from FLUXNET) and the Bowen Ratio method.

The `QaQc` object also has multiple tools for temporal frequency aggregation and resampling, estimation of climatic and statistical variables (e.g. ET and potential shortwave radiation), downloading gridMET reference

ET, managing data and metadata, interactive validation plots, and managing a structure for input and output data files. Input data is expected to be a *Data* instance or a `pandas.DataFrame`.

Keyword Arguments

- **data** (*Data*) – *Data* instance to create *QaQc* instance.
- **drop_gaps** (*bool*) – default `True`. If `True` automatically filter variables on days with sub-daily measurement gaps less than `daily_frac`.
- **daily_frac** (*float*) – default 1.00. Fraction of sub-daily data required otherwise the daily value will be filtered out if `drop_gaps` is `True`. E.g. if `daily_frac = 0.5` and the input data is hourly, then data on days with less than 12 hours of data will be forced to null within *QaQc.df*. This is important because systematic diurnal gaps will affect the autmoatic resampling that occurs when creating a *QaQc* instance and the daily data is used in closure corrections, other calculations, and plots. If sub-daily linear interpolation is applied to energy balance variables the gaps are counted *after* the interpolation.
- **max_interp_hours** (*None or float*) – default 2. Length of largest gap to fill with linear interpolation in energy balance variables if input datas temporal frequency is less than daily. This value will be used to fill gaps when $Rn > 0$ or Rn is missing during each day.
- **max_interp_hours_night** (*None or float*) – default 4. Length of largest gap to fill with linear interpolation in energy balance variables if input datas temporal frequency is less than daily when $Rn < 0$ within 12:00PM-12:00PM daily intervals.

agg_dict

Dictionary with internal variable names as keys and method of temporal resampling (e.g. “mean” or “sum”) as values.

Type dict

config

Config parser instance created from the data within the config.ini file.

Type `configparser.ConfigParser`

config_file

Absolute path to config.ini file used for initialization of the *fluxdataqaqc.Data* instance used to create the *QaQc* instance.

Type `pathlib.Path`

corrected

False until an energy balance closure correction has been run by calling *QaQc.correct_data*.

Type bool

corr_methods

List of Energy Balance Closure correction routines usable by *QaQc.correct_data*.

Type tuple

corr_meth

Name of most recently applied energy balance closure correction.

Type str or None

elevation

Site elevation in meters.

Type float

gridMET_exists
True if path to matching gridMET time series file exists on disk and has time series for reference ET and precipitation and the dates for these fully overlap with the energy balance variables, i.e. the date index of `QaQc.df`.

Type bool

gridMET_meta
Dictionary with information for gridMET variables that may be downloaded using `QaQc.download_gridMET`.

Type dict

inv_map
Dictionary with input climate file names as keys and internal names as values. May only include pairs when they differ.

Type dict

latitude
Site latitude in decimal degrees.

Type float

longitude
Site longitude in decimal degrees.

Type float

out_dir
Default directory to save output of `QaQc.write` or `QaQc.plot` methods.

Type pathlib.Path

n_samples_per_day
If initial time series temporal frequency is less than 0 then this value will be updated to the number of samples detected per day, useful for post-processing based on the count of sub-daily gaps in energy balance variables, e.g. "LE_subday_gaps".

Type int

plot_file
path to plot file once it is created/saved by `QaQc.plot`.

Type pathlib.Path or None

site_id
Site ID.

Type str

temporal_freq
Temporal frequency of initial (as found in input climate file) data as determined by `pandas.infer_freq`.

Type str

units
Dictionary with internal variable names as keys and units as found in config as values.

Type dict

variables
Dictionary with internal variable names as keys and names as found in the input data as values.

Type dict

Note: Upon initialization of a *QaQc* instance the temporal frequency of the input data checked using `pandas.infer_freq` which does not always correctly parse datetime indices, if it is not able to correctly determine the temporal frequency the time series will be resampled to daily frequency but if it is in fact already at daily frequency the data will be unchanged. In this case the *QaQc.temporal_freq* will be set to "na".

`agg_dict = {'ASCE_ETo': 'sum', 'ASCE_ETr': 'sum', 'ET': 'sum', 'ET_corr': 'sum', 'E`

`corr_methods = ('ebr', 'br', 'lin_regress')`

`correct_data`(*meth='ebr', et_gap_fill=True, y='Rn', refET='ETr', x=['G', 'LE', 'H'], fit_intercept=False*)

Correct turblent fluxes to improve energy balance closure using an Energy Balance Ratio method modified from FLUXNET.

Currently three correction options are available: 'ebr' (Energy Balance Ratio), 'br' (Bowen Ratio), and 'lin_regress' (least squares linear regression). If you use one method followed by another corrected,the corrected versions of LE, H, ET, ebr, etc. will be overwritten with the most recently used approach.

This method also computes potential clear sky radiation (saved as "rso") using an ASCE approach based on station elevation and latitude. ET is calculated from raw and corrected LE using daily air temperature to correct the latent heat of vaporization, if air temp. is not available in the input data then air temp. is assumed at 20 degrees celcius.

Corrected or otherwise newly calculated variables are named using the following suffixes to distinguish them:

```
uncorrected LE, H, etc. from input data have no suffix
_corr uses adjusted LE, H, etc. from the correction method used
_user_corr uses corrected LE, H, etc. found in data file (if provided)
```

Parameters

- **y** (*str*) – name of dependent variable for regression, must be in *QaQc.variables* keys, or a user-added variable. Only used if `meth='lin_regress'`.
- **x** (*str or list*) – name or list of independent variables for regression, names must be in *QaQc.variables* keys, or a user-added variable. Only used if `meth='lin_regress'`.

Keyword Arguments

- **meth** (*str*) – default 'ebr'. Method to correct energy balance.
- **et_gap_fill** (*bool*) – default True. If true fill any remaining gaps in corrected ET with ETr * ETrF, where ETr is alfalfa reference ET from gridMET and ETrF is the filtered, smoothed (7 day moving avg. min 2 days) and linearly interpolated crop coefficient. The number of days in each month that corrected ET are filled will is provided in *QaQc.monthly_df* as the column "ET_gap".
- **refET** (*str*) – default "ETr". Which gridMET reference product to use for ET gap filling, "ETr" or "ETo" are valid options.
- **fit_intercept** (*bool*) – default False. Fit intercept for regression or set to zero if False. Only used if `meth='lin_regress'`.
- **apply_coefs** (*bool*) – default False. If `True` then apply fitted coefficients to their respective variables for linear regression correction method, rename the variables with the suffix "_corr".

Returns None

Example

Starting from a correctly formatted config.ini and climate time series file, this example shows how to read in the data and apply the energy balance ratio correction without gap-filling with gridMET ETr x ETrF.

```
>>> from fluxdataqaqc import Data, QaQc
>>> d = Data('path/to/config.ini')
>>> q = QaQc(d)
>>> q.corrected
    False
```

Now apply the energy balance closure correction

```
>>> q.correct_data(meth='ebr', et_gap_fill=False)
>>> q.corrected
    True
```

Note: If `et_gap_fill` is set to `True` (default) the gap filled days of corrected ET will be used to recalculate LE_corr for those days with the gap filled values, i.e. LE_corr will also be gap-filled.

Note: The *ebr_corr* variable or energy balance closure ratio is calculated from the corrected versions of LE and H independent of the method. When using the 'ebr' method the energy balance correction factor (what is applied to the raw H and LE) is left as calculated (inverse of ebr) and saved as *ebc_cf*.

See also:

For explanation of the linear regression method see the `QaQc.lin_regress` method, calling that method with the keyword argument `apply_coefs=True` and *Rn* as the y variable and the other energy balance components as the x variables will give the same result as the default inputs to this function when `meth='lin_regress`.

daily_ASCE_refET(*reference='short'*, *anemometer_height=None*)

Calculate daily ASCE standardized short (ETo) or tall (ETr) reference ET from input data and wind measurement height.

The resulting time series will automatically be merged into the `Data.df` dataframe named "ASCE_ETo" or "ASCE_ETr" respectively.

Keyword Arguments

- **reference** (*str*) – default "short", calculate tall or short ASCE reference ET.
- **anemometer_height** (*float or None*) – wind measurement height in meters , default `None`. If `None` then look for the "anemometer_height" entry in the **METADATA** section of the config.ini, if not there then print a warning and use 2 meters.

Returns None

Note: If the hourly ASCE variables were prior calculated from a `Data` instance they will be overwritten as they are saved with the same names.

df
See *fluxdataqaqc.Data.df* as the only difference is that the *QaQc.df* is first resampled to daily frequency.

download_gridMET(*variables=None*)
Download reference ET (alfalfa and grass) and precipitation from gridMET for all days in flux station time series by default.

Also has ability to download other specific gridMET variables by passing a list of gridMET variable names. Possible variables and their long form can be found in *QaQc.gridMET_meta*.

Upon download gridMET time series for the nearest gridMET cell will be merged into the instances dataframe attibute *QaQc.df* and all gridMET variable names will have the prefix "gridMET_" for identification.

The gridMET time series file will be saved to a subdirectory called "gridMET_data" within the directory that contains the config file for the current *QaQc* instance and named with the site ID and gridMET cell centroid lat and long coordinates in decimal degrees.

Parameters **variables** (*None, str, list, or tuple*) – default None. List of gridMET variable names to download, if None download ETr and precipitation. See the keys of the *QaQc.gridMET_meta* dictionary for a list of all variables that can be downloaded by this method.

Returns `None`

Note: Any previously downloaded gridMET time series will be overwritten when calling the method, however if using the the gap filling method of the "ebr" correction routine the download will not overwrite currently existing data so long as gridMET reference ET and precipitation is on disk and its path is properly set in the config file.

---

classmethod from_dataframe(*df, site_id, elev_m, lat_dec_deg, var_dict, drop_gaps=True, daily_frac=1.0, max_interp_hours=2, max_interp_hours_night=4*)
Create a *QaQc* object from a `pandas.DataFrame` object.

Parameters

- **df** (`pandas.DataFrame`) – DataFrame of climate variables with datetime index named 'date'
- **site_id** (*str*) – site identifier such as station name
- **elev_m** (*int or float*) – elevation of site in meters
- **lat_dec_deg** (*float*) – latitude of site in decimal degrees
- **var_dict** (*dict*) – dictionary that maps *flux-data-qaqc* variable names to user's columns in *df* e.g. {'Rn': 'netrad', . . . } see *fluxdataqaqc.Data.variable_names_dict* for list of *flux-data-qaqc* variable names

Returns None

Note: When using this method, any output files (CSVs, plots) will be saved to a directory named "output" in the current working directory.

---

gridMET_meta = {'ETr': {'name': 'daily_mean_reference_evapotranspiration_alfalfa', '

lin_regress(*y, x, fit_intercept=False, apply_coefs=False*)

Least squares linear regression on single or multiple independent variables.

For example, if the dependent variable (`y`) is Rn and the independent variables (`x`) are LE and H, then the linear regression will solve for the best fit coefficients of $Rn = c_0 + c_1 LE + c_2 H$. Any number of variables in the *`QaQc.variables`* can be used for `x` and one for `y`.

If the variables chosen for regression are part of the energy balance components, i.e. Rn, G, LE, H and `apply_coefs=True`, then the best fit coefficients will be applied to their respecitive variables with consideration of the energy balance equation, i.e. the signs of the coefficients will be corrected according to $Rn - G = LE + H$, for example if `y=H` and `x=['Rn', 'G','LE]`. i.e. solving $H = c_0 + c_1 Rn + c_2 G + c_3 LE$ then the coefficients c_2 and c_3 will be multiplied by -1 before applying them to correct G and LE according to the energy balance equation.

This method returns an `pandas.DataFrame` object containing results of the linear regression including the coefficient values, number of data pairs used in the, the root-mean-square-error, and coefficient of determination. This table can also be retrieved from the `QaQc.lin_regress_results` instance attribute.

Arguments:

y (str): name of dependent variable for regression, must be in *`QaQc.variables`* keys, or a user-added variable.

x (str or list): name or list of independent variables for regression, names must be in *`QaQc.variables`* keys, or a user-added variable.

Keyword Arguments:

fit_intercept (bool): default False. Fit intercept for regression or set to zero if False.

apply_coefs (bool): default False. If `True` then apply fitted coefficients to their respective variables, rename the variables with the suffix "_corr".

Returns: `pandas.DataFrame`

Example: Let's say we wanted to compute the linear relationship between net radiation to the other energy balance components which may be useful if we have strong confidence in net radiation measurements for example. The resulting coefficients of regression would give us an idea of whether the other components were "under-measured" or "over-measured". Then, starting with a *`Data`* instance:

```
>>> Q = QaQc(Data_instance)
>>> Q.lin_regress(y='Rn', x=['G','H','LE'], fit_intercept=True)
>>> Q.lin_regress_result
```

This would produce something like the following,

SITE_ID	Y (dependent var.)	c0 (intercept)	c1 (coef on G)	c2 (coef on LE)	c3 (coef on H)	RMSE (w/m2)	r2 (coef. det.)	n (sample count)
a_site	Rn	6.99350781229883	1.552	1.054	0.943	18.25	0.91	3386

In this case the intercept is telling us that there may be a systematic or constant error in the independent variables and that G is "under-measured" at the sensor by over 50 precent, etc. if we assume daily Rn is accurate as measured.

Tip: You may also use multiple linear regression to correct energy balance components using the `QaQc.correct_data` method by passing the `meth='lin_regress'` keyword argument.

`monthly_df`

Temporally resample time series data to monthly frequency based on monthly means or sums based on `QaQc.agg_dict`, provides data as `pandas.DataFrame`.

Note that monthly means or sums are forced to null values if less than 20 percent of a months days are missing in the daily data (`QaQc.df`). Also, for variables that are summed (e.g. ET or precipitation) missing days (if less than 20 percent of the month) will be filled with the month's daily mean value before summation.

If a `QaQc` instance has not yet run an energy balance correction i.e. `QaQc.corrected` = `False` before accessing `monthly_df` then the default routine of data correction (energy balance ratio method) will be conducted.

Utilize the `QaQc.monthly_df` property the same way as the `fluxdataqaqc.Data.df`, see it's API documentation for examples.

Tip: If you have additional variables in `QaQc.df` or would like to change the aggregation method for the monthly time series, adjust the instance attribute `QaQc.agg_dict` before accessing the `QaQc.monthly_df`.

`plot` (*ncols=1*, *output_type='save'*, *out_file=None*, *suptitle="*, *plot_width=1000*, *plot_height=450*, *sizing_mode='scale_both'*, *merge_tools=False*, *link_x=True*, ***kwargs*)

Creates a series of interactive diagnostic line and scatter plots of input and computed daily and monthly aggregated data.

The main interactive features of the plots include: pan, selection and scrol zoom, hover tool that shows paired variable values including date, and linked x-axes that pan/zoom togehter for daily and monthly time series plots.

It is possible to change the format of the output plots including adjusting the dimensions of subplots, defining the number of columns of subplots, setting a super title that accepts HTML, and other options. If variables are not present for plots they will not be created and a warning message will be printed. There are two options for output: open a temporary file for viewing or saving a copy to `QaQc.out_dir`.

A list of all potential time series plots created:

- energy balance components
- radiation components
- incoming shortwave radiation with ASCE potential clear sky (daily only)
- multiple soil heat flux measurements
- air temperature
- vapor pressure and vapor pressure deficit
- wind speed
- station precipitation and gridMET precipitation
- initial and corrected latent energy
- initial, corrected, gap filled, and reference evapotranspiration
- crop coefficient and smoothed and interpolated crop coefficient
- initial and corrected energy balance ratio

- multiple soil moisture measurements

A list of all potential scatter plots created:

- radiative versus turblent fluxes, initial and corrected
- initial versus corrected latent energy
- initial versus corrected evapotranspiration

Keyword Arguments

- **ncols** (*int*) – default 1. Number of columns of subplots.
- **output_type** (*str*) – default "save". How to output plots, "save", "show" in browser, "notebook" for Jupyter Notebooks, "return_figs" to return a list of Bokeh `bokeh.plotting.figure.Figure`s, or "return_grid" to return the :obj:`bokeh.layouts.gridplot`.
- **out_file** (*str or None*) – default `None`. Path to save output file, if `None` save output to *QaQc.out_dir* with the name [site_id]_plots.html where [site_id] is *QaQc.site_id*.
- **suptitle** (*str or None*) – default `None`. Super title to go above plots, accepts HTML/CSS syntax.
- **plot_width** (*int*) – default 1000. Width of subplots in pixels.
- **plot_height** (*int*) – default 450. Height of subplots in pixels, note for subplots the height will be forced as the same as `plot_width`.
- **sizing_mode** (*str*) – default "scale_both". Bokeh option to scale dimensions of `bokeh.layouts.gridplot`.
- **merge_tools** (*bool*) – default False. Merges all subplots toolbars into a single location if True.
- **link_x** (*bool*) – default True. If True link x axes of daily time series plots and monthly time series plots so that when zooming or panning on one plot they all zoom accordingly, the axes will also be of the same length.

Example

Starting from a correctly formatted config.ini and climate time series file, this example shows how to read in the data and produce the default series of plots for viewing with the addition of text at the top of plot that states the site's location and ID.

```
>>> from fluxdataqaqc import Data, QaQc
>>> d = Data('path/to/config.ini')
>>> q = QaQc(d)
>>> q.correct_data()
>>> # create plot title from site ID and location in N. America
>>> title = "<b>Site:</b> {}; <b>Lat:</b> {}N; <b>Long:</b> {}W".format(
>>>     q.site_id, q.latitude, q.longitude
>>> )
>>> q.plot(
>>>     ncols=2, output_type='show', plot_width=500, suptitle=title
>>> )
```

Note, we specified the width of plots to be smaller than default because we want both columns of subplots to be viewable on one page.

Tip: To reset all subplots at once, refresh the page with your web browser.

Note: Additional keyword arguments that are recognized by `bokeh.layouts.gridplot` are also accepted by *`QaQc.plot`*.

`write`(*out_dir=None*, *use_input_names=False*)

Save daily and monthly time series of initial and "corrected" data in CSV format.

Note, if the energy balance closure correction (*`QaQc.correct_data`*) has not been run, this method will run it with default options before saving time series files to disk.

The default location for saving output time series files is within an "output" subdifrectory of the parent directory containing the config.ini file that was used to create the *`fluxdataqaqc.Data`* and *`QaQc`* objects, the names of the files will start with the site_id and have either the "daily_data" or "monthly_data" suffix.

Keyword Arguments

- **`out_dir`** (str or `None`) – default `None`. Directory to save CSVs, if `None` save to *`out_dir`* instance variable (typically "output" directory where config.ini file exists).
- **`use_input_names`** (*`bool`*) – default `False`. If `False` use `flux-data-qaqc` variable names as in output file header, or if `True` use the user's input variable names where possible (for variables that were read in and not modified or calculated by `flux-data-qaqc`).

Returns `None`

Example

Starting from a config.ini file,

```
>>> from fluxdataqaqc import Data, QaQc
>>> d = Data('path/to/config.ini')
>>> q = QaQc(d)
>>> # note no energy balance closure correction has been run
>>> q.corrected
    False
>>> q.write()
>>> q.corrected
    True
```

Note: To save data created by multiple correction routines, be sure to run the correction and then save to different output directories otherwise output files will be overwritten with the most recently used correction option.

5.3 Plot

class `fluxdataqaqc.`**`Plot`**

Bases: `object`

Container of plot routines of `fluxdataqaqc` including static methods that can be used to create and update interactive line and scatter plots from an arbitrary `pandas.DataFrame` instance.

Note: The *`Data`* and *`QaQc`* objects both inherit all methods of *`Plot`* therefore allowing them to be easily used for custom interactive time series plots for data within input data (in *`fluxdataqaqc.Data.df`*) and daily and monthly data in *`fluxdataqaqc.QaQc.df`* and *`QaQc.monthly_df`*.

static add_lines (*fig*, *df*, *plt_vars*, *colors*, *x_name*, *source*, *labels=None*, ***kwargs*)

Add a multiple time series to a `bokeh.plotting.figure.Figure` object using data from a datetime indexed `pandas.DataFrame` with an interactive hover tool.

Interactive hover shows the values of all time series data and date that is added to the figure.

Parameters

- **df** (`pandas.DataFrame`) – `pandas.DataFrame` containing time series data.
- **plt_vars** (*`list`*) – list of data columns in `df` to plot.
- **colors** (*`list`*) – list of line colors for variables in `plt_vars`.
- **x_name** (*`str`*) – name of the x-axis variable, e.g. the datetime index, in the `pandas.DataFrame` (`df`) containing data to plot.
- **source** (`bokeh.models.sources.ColumnDataSource`) – column data source created from the `pandas.DataFrame` with data to plot, i.e. `df`.
- **labels** (`list` or `None`) – default `None`. Labels for each plot variable in `plt_vars`.

Returns if none of the variables in `plt_vars` are found in `df` then return `None` otherwise returns the updated figure.

Return type ret (`None` or `bokeh.plotting.figure.Figure`)

Example

Similar to *`Plot.line_plot`* we first need to create a `bokeh.models.sources.ColumnDataSource` from a `pandas.DataFrame`. This example shows how to plot two variables, daily corrected latent energy and sensible heat on the same plot.

```
>>> from fluxdataqaqc import Data, QaQc, Plot
>>> d = Data('path/to/config.ini')
>>> q = QaQc(d)
>>> q.correct_data()
```

Now the *`QaQc`* instance should have the “LE_corr” (corrected latent energy) and “H_corr” (corrected sensible heat) columns, we can now make a `bokeh.models.sources.ColumnDataSource` from *`fluxdataqaqc.QaQc.df`* or *`fluxdataqaqc.QaQc.monthly_df`*,

```
>>> from bokeh.plotting import ColumnDataSource, figure, show
>>> df = q.df
>>> plt_vars = ['LE_corr', 'H_corr']
>>> colors = ['blue', 'red']
>>> labels = ['LE', 'H']
>>> source = ColumnDataSource(df)
>>> fig = figure(
>>>     x_axis_label='date', y_axis_label='Corrected Turbulent Flux'
>>> )
```

(continues on next page)

(continued from previous page)

```
>>> Plot.add_lines(
>>>     fig, df, plt_vars, colors, 'date', source, labels=labels
>>> )
>>> show(fig)
```

Note: This method is also available from the *Data* and *QaQc* objects.

static line_plot (*fig*, *x*, *y*, *source*, *color*, *label=None*, *x_axis_type='date'*, ***kwargs*)

Add a single time series to a `bokeh.plotting.figure.Figure` object using data from a datetime indexed `pandas.DataFrame` with an interactive hover tool.

Interactive hover shows the values of all time series data and date that is added to the figure.

Parameters

- **fig** (`bokeh.plotting.figure.Figure`) – a figure instance to add the line to.
- **x** (*str*) – name of the datetime index or column in the `pandas.DataFrame` containing data to plot.
- **y** (*str*) – name of the column in the `pandas.DataFrame` to plot.
- **source** (`bokeh.models.sources.ColumnDataSource`) – column data source created from the `pandas.DataFrame` with data to plot.
- **color** (*str*) – color of plot line, see Bokeh for color options.
- **label** (str or `None`) – default `None`. Label for plot legend (for `y`).
- **x_axis_type** (`str` or `None`) – default 'date'. If "date" then the x-axis will be formatted as month-day-year.

Returns `None`

Example

To use the *Plot.line_plot* function we first need to create a `bokeh.models.sources.ColumnDataSource` from a `pandas.DataFrame`. Let's say we want to plot the monthly time series of corrected latent energy, starting from a config.ini file,

```
>>> from fluxdataqaqc import Data, QaQc, Plot
>>> d = Data('path/to/config.ini')
>>> q = QaQc(d)
>>> q.correct_data()
```

Now the *QaQc* should have the "LE_corr" (corrected latent energy) column, we can now make a `bokeh.models.sources.ColumnDataSource` from *fluxdataqaqc.QaQc.df* or *fluxdataqaqc.QaQc.monthly_df*,

```
>>> from bokeh.plotting import ColumnDataSource, figure, show
>>> source = ColumnDataSource(q.monthly_df)
>>> # create the figure before using line_plot
>>> fig = figure(x_axis_label='date', y_axis_label='Corrected LE')
>>> Plot.line_plot(
>>>     fig, 'date', 'LE_corr', source, color='red', line_width=3
>>> )
>>> show(fig)
```

Notice, `line_width` is not an argument to *`Plot.line_plot`* but it is an acceptable keyword argument to `bokeh.plotting.figure.Figure` and therefore will work as expected.

Note: This method is also available from the *`Data`* and *`QaQc`* objects.

---

static scatter_plot (*fig*, *x*, *y*, *source*, *color*, *label=''*, *lsrl=True*, *date_name='date'*, ***kwargs*)

Add paired time series data to an interactive Bokeh scatter plot `bokeh.plotting.figure.Figure`.

Handles missing data points (gaps) by masking out indices in `x` and `y` where one or both are null. The `lsrl` option adds the best fit least squares linear regression line with y-intercept through zero and reports the slope of the line in the figure legend. Interactive hover shows the values of all paired (x,y) data and date that is added to the figure.

Returns minimum and maximum `x` and minimum and maximum `y` values of paired data which can be used for adding a one to one line to the figure or other uses.

Return type (tuple)

Example

Let's say that we wanted to run the energy balance ratio closure correction including gap filling with reference ET * crop coefficient and then plot corrected ET versus the calculated ET from reference ET (named "et_fill" in `flux-data-qaqc`) which is calculated on all days even those without gaps. Similar to *`Plot.line_plot`* we first need to create a `bokeh.models.sources.ColumnDataSource` from a `pandas.DataFrame`.

```
>>> from fluxdataqaqc import Data, QaQc
>>> d = Data('path/to/config.ini')
>>> q = QaQc(d)
>>> q.correct_data()
```

Now the *`QaQc`* instance should have the "et_corr" (corrected ET) and "et_fill" (et calculated from reference ET and crop coefficient) columns, we can now make a `bokeh.models.sources.ColumnDataSource` from *`fluxdataqaqc.QaQc.df`* or *`fluxdataqaqc.QaQc.monthly_df`*,

```
>>> from bokeh.plotting import ColumnDataSource, figure, show
>>> df = q.df
>>> source = ColumnDataSource(df)
>>> fig = figure(
>>>     x_axis_label='ET, corrected', y_axis_label='ET, gap fill'
>>> )
>>> # note, we are calling this plot method from a QaQc instace
>>> q.scatter_plot(
>>>     fig, 'ET_corr', 'ET_fill', source, 'red', label='lslr'
>>> )
>>> show(fig)
```

The `label` keyword argument will be used in the legend and since the least squares linear regression line between x and y is being calculated the slope of the line will also be printed in the legend. In this case, if the slope of the regression line is 0.94 then the legend will read "lslr, slope=0.94".

Note: Extra keyword arguments (accepted by `bokeh.plotting.figure.Figure`) will be passed to the scatter plot but not to the least squares regression line plot.

Note: This method is also available from the *Data* and *QaQc* objects.

5.4 utility classes and functions

Collection of utility objects and functions for the `fluxdataqaqc` module.

class `fluxdataqaqc.util.`**`Convert`**

Bases: `object`

Tools for unit conversions for `flux-data-qaqc` module.

`allowable_units = {'G': ['w/m2', 'mj/m2'], 'H': ['w/m2', 'mj/m2'], 'LE': ['w/m2', 'mj/`

`classmethod convert` (*var_name*, *initial_unit*, *desired_unit*, *df*)

Givin a valid initial and desired variable dimension for a variable within a `pandas.DataFrame`, make the conversion and return the updated `pandas.DataFrame`.

For a list of variables that require certain units within `flux-data-qaqc` see *Convert.allowable_units* (names of allowable options of input variable dimensions) and *Convert.required_units* (for the mandatory dimensions of certain variables before running QaQc calculations).

Parameters

- **`var_name`** (*str*) – name of variable to convert in `df`.
- **`initial_unit`** (*str*) – name of initial unit of variable, must be valid from *Convert.allowable_units*.
- **`desired_unit`** (*str*) – name of units to convert to, also must be valid.
- **`df`** (`pandas.DataFrame`) – `pandas.DataFrame` containing variable to be converted, i.e. with `var_name` in columns.

Returns updated dataframe with specified variable's units converted

Return type df (`pandas.DataFrame`)

Note: Many potential dimensions may not be provided for automatic conversion, if so you may need to update your variable dimensions manually, e.g. within a *Data.df* before creating a *QaQc* instance. Unit conversions are required for variables that can potentially be used in calculations within *Data* or *QaQc*.

`pretty_unit_names = {'c': 'C', 'f': 'F', 'hpa': 'hPa', 'k': 'K', 'kpa': 'kPa', 'p`

`required_units = {'G': 'w/m2', 'H': 'w/m2', 'LE': 'w/m2', 'Rn': 'w/m2', 'lw_in': 'w/`

`fluxdataqaqc.util.`**`monthly_resample`** (*df*, *cols*, *agg_str*, *thresh=0.75*)

Resample dataframe to monthly frequency while excluding months missing more than a specified percentage of days of the month.

Parameters

- **`df`** (`pandas.DataFrame`) – datetime indexed DataFrame instance
- **`cols`** (*list*) – list of columns in *df* to resample to monthy frequency
- **`agg_str`** (*str*) – resample function as string, e.g. 'mean' or 'sum'

Keyword Arguments **thresh** (*float*) – threshold (decimal fraction) of how many days in a month must exist for it to be temporally resampled, otherwise the monthly value for the month will be null.

Returns datetime indexed DataFrame that has been resampled to monthly time frequency.

Return type ret (pandas.DataFrame)

Note: If taking monthly totals (*agg_str* = 'sum') missing days will be filled with the months daily mean before summation.

fluxdataqaqc.util.**write_configs**(*meta_df*, *data_dict*, *out_dir=None*)

Write multiple config files based on collection of site metadata and a dictionary containing variable information.

Useful for creating config files for *flux-data-qaqc* for batches of flux stations that utilize the same naming conventions and formatting.

Parameters

- **meta_df** (pandas.DataFrame) – dataframe that contains the following columns (or more) that describe metadata for multiple climate stations: 'site_id', 'climate_file_path', 'station_longitude' 'station_elevation', 'station_latitude', and 'missing_data_value'. Elevation should be in meters and latitude is in decimal degrees. Additional metadata columns will be added to the config file for each site, e.g. 'QC_flag', 'anemometer_height', and any others.
- **data_dict** (*dict*) – dictionary that maps *flux-data-qaqc* config names to user's column names in input files header e.g. {'net_radiation_col': 'netrad', 'net_radiation_units' : 'w/m2'} Anything that *flux-data-qaqc* config files "DATA" section can be present here including QC flag names, multiple soil moisture names and weights.

Keyword Arguments **out_dir** (*str or None*) – default None. Directory to save config files, if None then save to currect working directory.

Returns

list of pathlib.Path objects of full paths to each config file written.

Return type configs (list)

Raises Exception – if one of the mandatory metadata columns does not exist in *meta_df*.

CHAPTER 6

Automated testing with pytest

Software tests are automatically run each time a change to `flux-data-qaqc` is made on the master branch in GitHub using this GitHub Actions workflow. Automated tests help spot potential bugs early so that they be identified and corrected efficiently resulting in an improved user experience.

6.1 Running tests manually

`pytest` is required to run software tests that are provided. You can install `pytest` with PIP:

```
pip install pytest
```

The tests utilize the example flux input data and `flux-data-qaqc` configuration files that are provided with the software whether installed from PyPI or GitHub. These files can be found here.

To run the tests, navivgate to the root directory of the source code (from the command line or shell) and run pytest:

```
pytest
```

This will print out basic test results, usage of `pytest` plugins and command line options can be used for getting more information out of the tests.

CHAPTER 7

Contributors

7.1 OpenET team

Guidance and feedback has been given by the OpenET team members,

- Dr. Ayse Kilic
- Christian Dunkerly
- Forrest Melton
- Dr. Gabriel Senay
- Dr. Joshua Fisher
- Dr. John Volk
- Dr. Justin Huntington
- Dr. Martha Anderson
- Dr. Richard G. Allen

7.2 Acknowledgements

The Energy Balance Ratio correction method was modified from the FLUXNET2015 methodology for correcting daily latent energy and sensible heat. Additionally, FLUXNET and AmeriFlux eddy covariance data was used for code development and we thank them for mantaining and providing eddy flux data.

We would also like to acknowledge the RefET python package and its developer Charles Morton. `RefET` methods are used in `flux-data-qaqc` for calculating the American Soceity of Civil Engineers Standardized reference ET using the hourly and daily formulations, as well as the calculation for potential solar radiation.

7.3 Contributing

`flux-data-qaqc` is an open-source Python package and anyone seriously interested in contributing is encouraged to do so. Look for current issues to get started or offer direct changes with a pull request.

7.4 Report issues or problems with the software

Please open a GitHub issue if you have a technical problem with the software. The issues page is here.

7.5 Seek support or give feedback

Any suggestions or feedback that are nontechnical but related to software development or usage please open a GitHub issue. Any other feedback should be sent to John.Volk@dri.edu.

CHAPTER 8

Change Log

8.1 Version 0.1.6

Add automated tests using GitHb Actions, see here and added in description of how to run tests locally on docs.

Remove `xlrd` reader as a dependency due to outdated reading ability as a `Pandas` excel reader.

Other minor bug fixes related to `Plot` class.

Ass JOSS paper and publish software on Zenodo.

8.2 Version 0.1.5

Add configuration writing function *`util.write_configs`* to `util` module to facilitate batch processing os similar formatted input files via a station metadata file and data dictionary.

Update check on energy balance ratio closure correction to also check if the inverse of the energy balance ratio is greater than 0.5, in other words $\frac{1}{EBR} > |0.5|$ to avoid closure correction factors that are too small. This check occurs both after step 3 and 6 of the energy balance closure correction routine.

8.3 Version 0.1.4

Relax default allowance for missing days threshold from 90 (~ 3 days) to 80 % (~ 6 days) in the monthly resample algorithm. In other words if a month has more than 80 % missing daily values, its monthly aggregate will not be resampled, it will be replaced with a null value. The threshold is a keyword argument to the `util.monthly_resample` function, but the default is used in any automatic resampling of variables. As a reminder, the number of missing days per month which is tabulated for some variables can be used to fine tune this filter. This change was implemented in version 0.1.4.post1.

Add daily ASCE standardized reference ET calculation option from the *`QaQc.daily_ASCE_refET`* method. Also added automatic estimation of daily maximum and minimum air temperature from input (e.g. hourly) data and added the input variables to the list of variables that are linearly interpolated before taking daily aggregates in the *QaQc* constructor. In other words, the inputs to the daily ASCE reference ET formulation: ea, tmin, tmax, rs, wind speed, are interpolated over daytime and nighttime hourly gaps (2 and 4 default) before taking daily means, mins, maxs, and subsequently used in the daily ASCE calculations.

Changed default keyword argument `reference` to "short" of the *Data.hourly_ASCE_refET* method.

Add automatic calculations for high frequency (e.g. hourly or half hourly) data including dew temperature and relative humidity from ea and es if available. The calculations occur when first loading input data, i.e. when *Data.df* attribute is accessed. Saturation vapor pressure (es) if calculated at hourly/daily frequency is now saved and added to *Data.df* and *QaQc.df* properties.

Require Pandas >= 1.0, changes are not backwards compatible due to internal pandas argument deprecations particularly in the `pandas.grouper` object.

Require Bokeh >= 2.0, changes are not backwards compatible due to legend keyword argument name changes in Bokeh 2.

Minor changes to remove package deprecation warnings from `Pandas` and `Bokeh` related to their respective large changes.

Add package dependency `openpyxl` package as a fallback for reading in headers of Excel files when `xlrd` is unmaintained and failing with previously working tools for reading metadata on Excel files.

Add a requirements.txt file with package.

8.4 Version 0.1.3

Add option to use gridMET grass reference ET (ETo) and EToF for gap filling daily ET. The default behavior still uses alfalfa reference ET, to use ETo assign the `refET="ETo"` keyword argument to *QaQc.correct_data* or directly to `QaQc._ET_gap_fill`. The ET and ET reference fraction plot labels are updated to show the correct reference ET variable used.

Improve scaling of scatter plots to give equal x and y axis lengths, change return of *Plot.scatter_plot* to return tuple of (xmin, xmax, ymin, ymax) for use in plotting one to one lines or limiting axes lengths.

8.5 Version 0.1.2

Change default functionality of the *QaQc.write* method to use the internal variable names (as opposed to the input names) of `flux-data-qaqc` in the header files of the output daily and monthly time series CSV files. For example, the column for net radiation is always named and saved as "Rn". This can be reversed to the previous behavior of using the user's input names by setting the new `use_input_names` keyword argument to *QaQc.write* to `True`.

Change the *Plot.scatter_plot* underlying call to the `bokeh` modules scatter plot as opposed to the set circle glyph plot. This allows the user to change the symbol from circle to others by passing a valid value to the scatter_plot's `marker` keyword argument, e.g. `marker='cross'`.

8.6 Version 0.1.1

Add least squares linear regression method for single or multivariate input; specifically the `QaQc.lin_regress()` method. It can be used to correct energy balance components or for any arbitrary time series data loaded in a `QaQc` instance. It produces and returns a readable table with regression results (fitted coefficients, root-mean-square-error, etc.) which can be accessed from `QaQc.lin_regress_results` after calling the method. The default regression if used to correct energy balance components assumes net radiation is accurate (as the dependent variable):

$Rn = c_0 + c_1 G + c_2 LE + c_3 H$

where $c_0 = 0$.

This regression utilizes the scikit-learn Python module and therefore it was added to the environment and setup files as a dependency.

8.7 Version 0.1.0

Add hourly ASCE standardized reference ET calculation to the `Data` class as *`Data.hourly_ASCE_refET`* with options for short and tall (grass and alfalfa) reference ET calculations. If the input data is hourly or higher frequency the input data for the reference ET calculation will automatically be resampled to hourly data. If the input data is hourly then the resulting reference ET time series will be merged with the *`Data.df`* attribute otherwise if the input data is at a temporal frequency > hourly, then the reference ET time series will be return by the *`Data.hourly_ASCE_refET`* method.

Add methods and options to linearly interpolate energy balance variables based on length of gaps during daytime ($Rn > 0$) and night ($Rn < 0$). These methods are run automatically by the `QaQc` constructor if temporal frequency of input is detected as less than daily. New keyword arguments to `QaQc` are `max_interp_hours` and `max_interp_hours_night` respectively.

Other notable changes:

- first release on GitHub
- creation of this file/page (the Change Log)
- add optional return options to plot methods of `Data` and `QaQc` objects for custimization of default plots or to show/use a subset of them

8.8 Version 0.0.9

Major improvements and notabable changes include:

- add package to PyPI
- change allowable gap percentage for monthly time series to 10 % from 70 %
- add reading of wind direction data, BSD3 license, add package data
- fix bugs related to filtering of subday gaps
- improve plots and other error handling, add feature to hide lines in line plots

8.9 Version 0.0.5

Major improvements and notabable changes include:

- first documentation on ReadTheDocs
- add multiple pages in docs such as installation, config options, basic tutorials, full API reference, etc.
- improve and streamline config file options

- add vapor pressure and vapor pressure deficit calculations for hourly or lower frequency data in the `Data.df` property (upon initial loading of time series into memory
- add automatic unit conversions and checks on select input variables using the `Convert` class in the `util` module
- add new plots in default plots from `QaQc` class, e.g. filtered and raw ETrF
- many rounds of improvements to plots, e.g. hover tooltips, linked axes, style, options for columns, etc.
- modify Energy Balance Ratio to filter out extreme values of filtered Energy Balance Ratio correction factors
- improve temporal resampling with options to drop days with certain fraction of sub-daily gaps
- track number of gap days in monthly time series of corrected ET
- add examples of ET gap-filling to docs and change most example data to use Twitchel Island alfalfa site data from AmeriFlux
- add plotting of input data using `plot` method of `Data` instance which allows for viewing of input data at its initial temporal frequency

8.10 Version 0.0.1

First working version, many changes, milestones included:

- basic templates and working versions of the `Data`, `QaQc`, and `Plot` classes
- versions and improvements to daily and monthly resampling
- Bowen and Energy Balance Ratio correction routines
- example Jupyter notebooks including with FLUXNET and USGS data
- calculation of potential clear sky radiation
- changing variable naming system to use internal and user names
- ability to read in multiple soil heat flux and soil moisture measurements and calculate weighted averages
- make package installable and Conda environment
- add input data filtering using quality control flags (numeric threshold and flags)
- reading of input variables' units
- added the `util` submodule with methods for resammpling time series
- ability to take non-weighted averages for any acceptable input variable
- add config file options like date parsing
- removed filtering and smoothing options from Bowen Ratio method and other modifications to it
- add methods for downloading gridMET variables based on location in CONUS
- add routine for gap filling ET based on gridMET ETrF that is smoothed and filtered
- improved `Plot` class to contain modular plot methods (line and scatter) for use with arbitrary data
- changed internal variable naming, e.g. etr to ETr
- methods to estimate ET from LE that consider the latent heat of vaporization is affected by air temp.
- other updates to improve code structure and optimization of calculations

CHAPTER 9

Indices and tables

- genindex
- modindex
- search

Python Module Index

Index