CoTeDe Documentation

Release 0.21.1

Guilherme Castelão

Mar 31, 2020

CONTENTS

CHAPTER

ONE

USER DOCUMENTATION

CoTeDe at a glance

1.1 Overview

1.1.1 CoTeDe

CoTeDe is an Open Source Python package to quality control (QC) oceanographic data such as temperature and salinity. It was designed to attend individual scientists as well as real-time operations on large data centers. To achieve that, CoTeDe is highly customizable, giving the user full control to compose the desired set of tests including the specific parameters of each test, or choose from a list of preset QC procedures.

I believe that we can do better than we have been doing with more flexible classification techniques, which includes machine learning. My goal is to minimize the burden on manual expert QC improving the consistency, performance, and reliability of the QC procedure for oceanographic data, especially for real-time operations.

CoTeDe is the result from several generations of quality control systems that started in 2006 with real-time QC of TSGs and were later expanded for other platforms including CTDs, XBTs, gliders, and others.

Why use CoTeDe

CoTeDe contains several QC procedures that can be easily combined in different ways:

- Pre-set standard tests according to the recommendations by GTSPP, EGOOS, XBT, Argo or QARTOD;
- Custom set of tests, including user defined thresholds;
- Two different fuzzy logic approaches: as proposed by Timms et. al 2011 & Morello et. al. 2014, and using usual defuzification by the bisector;
- A novel approach based on Anomaly Detection, described by Castelao 2015.

Each measuring platform is a different realm with its own procedures, metadata, and meaningful visualization. So CoTeDe focuses on providing a robust framework with the procedures and lets each application, and the user, to decide how to drive the QC. For instance, the pySeabird package is another package that understands CTD and uses CoTeDe as a plugin to QC.

Documentation

A detailed documentation is available at http://cotede.readthedocs.org, while a collection of notebooks with examples is available at http://nbviewer.ipython.org/github/castelao/CoTeDe/tree/master/docs/notebooks/

Note: On version 0.20 there was an inversion of roles and instead of depending on PySeabird package, now CoTeDe is an independent package that can be installed as an extra-requirement of PySeabird package. The functionalities to quality control CTD and TSG are now in the package PySeabird. This allowed CoTeDe to focus on QC and better generalize for other platforms and instruments.

1.2 Installation

CoTeDe was intentionally kept simple, avoiding dependencies, but when inevitable it uses fundamental libraries. There are indeed many benefits from modern libraries like pandas and xarray (yes, CoTeDe is old), but the goal was to allow other applications to adopt CoTeDe with ease. The optional extras allows some expansion.

1.2.1 Requirements

- Python 2.7 or 3.X (recommended >=3.7)
- Numpy (>=1.11)
- Scipy (>=0.18.0)

Optional requirement

- GSW: a Python implementation of the Thermodynamic Equation of Seawater 2010 (TEOS-10). It is used to derive variables like sea water density from pressure, temperature, and salinity.
- OceansDB: a database of climatologies and bathymetry of the oceans. It is a requirement for tests like valid position at sea, climatology comparison, and others.
- Matplotlib: a powerfull library for data visualization. It is required for the graphic tools, like the visual inspection and classification of the data.
- Shapely: a Python package for computational geometry. It is required by the Regional Range test to evaluate which measurements are inside the considered domain.

1.2.2 Installing CoTeDe

Virtual Environments

You don't need to, but I strongly recommend to use virtualenv or conda.

Using pip

If you don't already have PIP running on your machine, first you need to install pip, then you can run:

```
$ pip install cotede
```

Custom Install

To install with GSW support, which allows to estimate density on the fly, you can run:

```
pip install cotede[GSW]
```

To install with OceansDB in order to be able to run climatology tests, you can run:

```
pip install cotede[OceansDB]
```

To install multiple extras:

```
pip install cotede[GSW,OceansDB,Shapely]
```

Last night's version

It is possible to install the latest version directly from the oven but, like a good bread, it might not taste as good as if you wait it to cool down:

```
$ pip install git+https://github.com/castelao/CoTeDe.git
```

If you can, use the standard pip install as shown previously.

1.2.3 Custom setup

The directory ~/.config/cotederc is the default home directory for CoTeDe support files, including the user custom QC setup. To use another directory, one can set and environment variable COTEDE_DIR. For example, if you use bash you could include the following lines in your .barsh_profile:

```
$ export COTEDE_DIR='~/my/different/path'
```

1.2.4 Optional

Climatology and bathymetry

The climatology comparison test and the at sea test use the package OceansDB, which maintains local files with the climatologies and bathymetry. Those files are automatically downloaded on the first time that they are required, but you can force the download by executing:

```
>>> import oceansdb; oceansdb.CARS()['sea_water_temperature']
>>> import oceansdb; oceansdb.WOA()['sea_water_temperature']
>>> import oceansdb; oceansdb.ETOPO()['topography']
```

That will create, if it doesn't already exist, a directory in your home: ~/.config, and place the required WOA, CARS, and etopo files there. That is it, you're ready to run cotede in place with any of the preset configurations. If you're going to a cruise, remember to run this before leave the dock, while you still have cheap and fast access to the network.

1.3 Data Model

Inside CoTeDe, the dataset to be analyzed is treated as a single object that contains all variables (temperature, salinity, fluorescence, ...), coordinates (pressure, depth, ...), and metadata. This data model is the same independent of the sampling platform, therefore, temperature measurements collected by an XBT, a Spray glider, a CTD rosette, a mooring or a Saildrone are all accessed in the same way. The difference is that each case might use a different set of QC tests, and each test decides what should be used to evaluate the quality of the dataset.

Other applications can connect with CoTeDe by providing the data using this data model. For example, pySeabird is another Python package able to parse CTD raw data and organize it as described on this session before calling CoTeDe to QC the profiles.

1.3.1 Data

Each variable is expected to be accessible as an item of the dataset object, and it should return a sequence, preferably a numpy array. Considering a dataset named 'ds', to access the temperature:

```
$ ds['TEMP']
>>> masked_array(data=[17, 16.8, 16], mask=False, fill_value=1e+20)
```

Coordinates and other auxiliary variables with the same dimension of the variable of interest should be available on the same way, thus for a profile the depth of each measurement would be accessible as:

```
$ ds['DEPTH']
>>> masked_array(data=[0, 10, 20], mask=False, fill_value=999999)
```

1.3.2 Metadata

Scalar metadata representative for the whole dataset should be available in the property attrs. For example, to obtain the nominal time of a CTD cast:

```
$ ds.attrs['datetime']
>>> datetime.datetime(2019, 11, 22, 5, 15, 57, 619332)
>>> numpy.datetime64('2019-11-22T05:16:56.932129')
```

or the nominal latitude of a mooring:

```
$ ds.attrs['latitude']
>>> 15
```

but if latitude has the same dimension of the data, like the along track latitude for a TSG, it should be available together with the data, like:

```
$ ds['latitude']
>>> masked_array(data=[14.998, 15.0, 15.001], mask=False, fill_value=np.nan)
```

Note: Numpy masked array is the prefered choice. In that case, whatever is masked will be considered that the data is unavailable. If not using masked arrays , missing data should be assigned with np.nan.

1.3.3 Minimalist solution

Possibly the simplest model to achieve that is:

```
class CoTeDeDataset(object):
  def __init__(self):
      self.attrs = {}
      self.data = {}

  def __getitem__(self, key):
      return self.data[key]

  def keys(self):
      return self.data.keys()
```

So that:

```
$ ds = CoTeDeDataset()
$ ds.data['TEMP'] = np.array([15, 14.8, 14.3])
$ ds.attrs['longitude'] = -38
...
```

Check the data model notebook for a complete example on how to use it.

1.4 Getting Started with CoTeDe

To quality control CTD or TSG, please check the package pySeabird.

1.4.1 Inside python

First load the module

```
>>> import cotede
```

With a data object from a CTD as described in the Data Model section, we can run the QC

```
>>> pqc = cotede.ProfileQC(ds)
```

The keys() will give you the data loaded from the CTD, similar to the ds itself

```
>>> pqc.keys()
```

To see one of the variables listed on the previous step

```
>>> pqc['sea_water_temperature']
```

The flags are stored at pqc.flags and is a dictionary, being one item per variable evaluated. For example, to see the flags for the salinity instrument

```
>>> pqc.flags['sea_water_salinity']
```

or for a specific test

```
>>> pqc.flags['sea_water_salinity']['gradient']
```

The class cotede.ProfileQCed is equivalent to the cotede.ProfileQC, but it already masks the non approved data (flag > 2). It can also be used like

```
>>> p = cotede.ProfileQCed(data)
>>> p['sea_water_temperature']
```

To choose which QC criteria to apply

```
>>> pqc = cotede.ProfileQC(ds, 'cotede')
```

or

```
>>> pqc = cotede.ProfileQC(ds, 'gtspp')
```

To define manually the test to apply

```
>>> pqc = cotede.ProfileQC(ds, {'sea_water_temperature': {'gradient': {'threshold': 6}
↪}})
```

1.4.2 More examples

I keep a notebooks collection of practical examples to Quality Control CTD data . If you have any suggestion, please let me know.

1.5 Tests for Quality Control

An automatic quality control is typically a composition of checks, each one looking for a different aspect to identify bad measurements. This section covers the concept of the available checks and some ways how those could be combined.

A description and references for each test are available in qctests. The result of each test is a flag ranking the quality of the data as described in *Flags*. Finally, most of the users will probably follow one of the recommended procedures (GTSPP, Argo, QARTOD . . .) described in *Quality Control Procedures*. If you are not sure what to do, start with one of those QC procedures and later fine tune it for your needs. The default procedure for CoTeDe is the result of my experience with the World Ocean Database.

1.5.1 Flags

The outcome of the QC evaluation is encoded following the IOC recommendation given in the table below For example, if the climatology database is not available, the output flag would be 0, while a fail on the same climatology test would return a flag 3, if following the *GTSPP* recommendations. By the end of all checks, each measurement receives an overall flag that is equal to the highest flag among all tests applied. Therefore, one mesurement with flag 0 was not evaluated at all, while a measurement with overall flag 4 means that at least one check considered that a bad data.

Flag	Meaning
0	No QC was performed
1	Good data
2	Probably good data
3	Probably bad data
4	Bad data
9	Missing data

The flags 2 and 3 usually cause some confusion: "What do you mean by probably good or bad?" The idea is to allow some choice for the final user. The process of defining the criteria for any QC test is a duel between minimizing false positives or false negatives, thus it is a choice: What is more important for you? There is no unique answer for all cases. Most of the users will use anything greater than 2 as non-valid measurements. Someone willing to pay the price of loosing more data to avoid by all means any bad data would rather discard anything greater than 1. While someone more concerned in not wasting any data, even if that means a few mistakes, would discard anything greater than 3. When designing a test or defining a new threshold, please assume that flag 4 is pretty confident that is a bad measurement.

It is typically expected to have one flag for each measurement in the dataset, but it is possible to have a situation with a single flag for the whole dataset. For instance, if a profile is checked only for a valid geolocation, it would get a single flag for the whole profile.

Some procedures also provide a continuous scale usually representing the probablity of a measurement being good, like the Anomaly Detection and the Fuzzy Logic. For details on that, please check the description of the specific check.

1.5.2 Quality Control Procedures

Although I slightly modified the names of some Q.C. test, the concept behind is still the same. The goal was to normalize all tests to return True if the data is good and False if the data is bad. For example, Argo's manual define "Impossible Date Test", while here I call it "test-valid-date".

Profile

GTSPP

Test	Flag		Threshold	
	if succeed	if fail	Temperature	Salinity
test-valid-date	1	4		
test-valid-position	1			
test-location-at-sea	1			
test-global-range	1		-2 to 40 C	0 to 41
test-gradient	1	4	10.0 C	5
test-spike	1		2.0 C	0.3
test-climatology	1			
test-profile-envelop				

EuroGOOS

Test	Flag		Threshold	
	if succeed	if fail	Temperature	Salinity
test-valid-date	1	4		
test-valid-position	1	4		
test-location-at-sea	1	4		
test-global-range	1	4	-2.5 to 40	2 to 41
test-digit-rollover	1	4	10.0 C	5
test-gradient-cond • < 500 • > 500	1	4	• 9.0 C • 3.0 C	• 1.5 • 0.5
test-spike-cond • < 500 • > 500	1	4	• 6.0 C • 2.0 C	• 0.9 • 0.3
test-climatology	1			

Argo (Incomplete)

Test	Flag		Threshold	
	if succeed	if fail	Temperature	Salinity
Platform Identification				
Valid Date				
Impossible location test				
Position on land test				
Impossible speed test				
test-global-range				
test-regional-range				
Pressure increasing				
test-spike				
Top an dbottom spike test: obsolete				
test-gradient				
test-digit-rollover				
Stuck value test				
test-density-inversion				
Grey list				
Gross salinity or temperature sensor drift				
Visual QC				
Frozen profile test				
Deepest pressure test				

IMOS (Incomplete)

Test	Flag		Threshold	
	if succeed	if fail	Temperature	Salinity
test-valid-date	1	3		
test-valid-position	1	3		
test-location-at-sea	1	3		
test-global-range	1		-2.5 to 40	2 to 41
test-gradient	1	4	10.0 C	5
test-spike	1		2.0 C	0.3
test-climatology	1			

QARTOD (Incomplete)

Test	Flag		Threshold	
	if succeed	if fail	Temperature	Salinity
Gap				
Syntax				
Location at Sea				
Gross Range				
Climatological				
test-spike				
Rate of Change		4		
Flat Line				
Multi-Variate				
Attenuated Signal				
Neighbor				
TS Curve Space				
Density Inversion		3	0.03 kg/m3	

TSG

Based on AOML procedure. Realtime data is evaluatd by tests 1 to 10, while the delayed mode is evaluated by tests 1 to 15.

1. Platform Identification
2. test-valid-date
3. Impossible Location
4. **`Location at Sea`_**
5. Impossible Speed
6. **`Global Range`_**
7. test-regional-range
8. test-spike
9. Constant Value
10. test-gradient
11. test-climatology
12. NCEP Weekly analysis
13. Buddy Check
14. Water Samples
15. Calibrations

XBT

1.5.3 References

Help & reference

- api
- usedby
- history

1.6 Indices and tables

- genindex
- modindex
- search