Unleashing the Power of Argo Data with Ease: Discover, Analyze, Visualize with argopy

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement no 824131. Call INFRADEV-03-2018-2019: Individual support to ESFRI and other world-class research infrastructures.

AUTHORS Guillaume Maze and Kevin Balem

Laboratory for Ocean Physics and Satellite remote sensing (LOPS), Brest, France
http://www.umr-lops.fr/

Full documentation

Why argopy ?

Surprisingly, the Argo community never provided its user base with a Python software to easily access and manipulate Argo measurements: argopy aims to fill this gap.

Despite, or because, its tremendous success in data management and in developing good practices and well calibrated procedures, the Argo dataset is very complex: with thousands of different variables, tens of reference tables and a user manual more than 100 pages long: argopy aims to help you navigate this complex realm.

User modes: Less data wrangling, more analysis

In order to ease Argo data analysis for the vast majority of users, we implemented in argopy different levels of verbosity and data processing to hide or simply remove variables only meaningful to experts.

argopy provides 3 user modes to set the "behind the scene" level of data post-processing:

- **expert** mode return all the Argo data, without any post-processing,
- **standard** mode simplifies the dataset, remove most of its jargon and return a priori good data, namely: QC=[1,2] & DM=[R,D,A]. This is the default mode.
- **research** mode simplifies the dataset to its heart, preserving only data of the highest quality for research studies, including studies sensitive to small pressure and salinity bias (e.g. calculations of global ocean heat content or mixed layer depth), namely: QC=1 & DM=D.

Select your user mode with a context, fetcher option or the global option:

```
import argopy
argopy.set_options(mode='expert')
```

Data sources

argopy allows users to fetch Argo data from several sources among the following:

- the **Ifremer erddap** server. Updated daily, this database holds the complete dataset and is efficient for large requests
- a **GDAC** server. This could be one of the 2 ftps or the Ifremer http server. It can also points toward your own local copy of the GDAC
- the **Argovis** server. Updated daily, provides access to QC=1 data only

Select your data source with a context, fetcher option or the global option:

```
import argopy
argopy.set_options(src='erddap')
argopy.set_options(src='gdac', ftp='https://...')
```

Other features

Control performances

- all data requests can seemingly be cached
- all data requests can automatically be chunked and parallelised

Documentation and support

- all argopy API are fully documented online
- online user support using chat and GitHub issues
- source code is continuously tested for all platforms and Python versions

BGC data

- working hard, coming up gradually in 2023 !

How to cite ?

Maze, G., & Balem, K. (2020). argopy: A Python library for Argo ocean data analysis. Journal of Open Source Software, 5(53). https://doi.org/10.21105/joss.02425

Usage

Fetch Argo data

Import the data fetcher, define an access point in *region*, *float* or *profile* and trigger download of data or index:

```
from argopy import DataFetcher
fetcher = DataFetcher().region([-75, -45, 20, 30, 0, 100, '2011-01', '2011-06'])
fetcher = DataFetcher().float([6902746, 6902755])
fetcher = DataFetcher().profile(6902746, [1,12])
fetcher.to_xarray()
fetcher.to_dataframe()
fetcher.to_index()
```

Manipulating Argo data

Use methods from the **argo** xarray accessor

Transformation

Points vs profiles

By default, argopy returns data as a collection of points that is easy to transform into a collection of profiles, and vice-versa:

```
ds.argo.point2profile()
ds_profiles.argo.profile2point()
```

Pressure levels: Interpolation

```
ds_profiles.argo.interp_std_levels([0,100,200,500])
```

Pressure levels: Group-by bins

```
b = np.arange(0.0, np.max(ds["PRES"]), 250.0)
ds.argo.groupby_pressure_bins(bins=b, select='deep')
ds.argo.groupby_pressure_bins(bins=b, select='random')
```

Filters

If you fetched data with the **expert** mode, you can use filters to help you curate the data:

QC flag filter

```
ds = ds.argo.filter_qc(QC_list=[1,2], QC_fields='all')
ds = ds.argo.filter_qc(QC_list=1, QC_fields='PSAL')
```

Data mode filter

```
ds = ds.argo.filter_data_mode()
```

OWC variables filter

```
ds.argo.filter_scalib_pres(force='default')
```

Complement original data

You can compute additional ocean variables from the TEOS-10:

```
ds = ds.argo.teos10(['SA', 'CT', 'CNDC'])
```

Argo meta data

You can use argopy to easily access the realm of Argo meta data:

Index files (support: core, synthetic and bio profiles index)

```
from argopy import ArgoIndex
ArgoIndex().N_RECORDS
ArgoIndex().to_dataframe()
ArgoIndex().search_lat_lon([-60, -55, 40, 45])
ArgoIndex().search_wmo([1901393, 6902755])
```

Reference tables (from NVS)

```
from argopy import ArgoNVSReferenceTables
ArgoNVSReferenceTables().all_tbl_name()
ArgoNVSReferenceTables().all_tbl()
ArgoNVSReferenceTables().tbl_name('R01')
ArgoNVSReferenceTables().tbl('R01')
```

Deployment Plan (from Ocean-OPS)

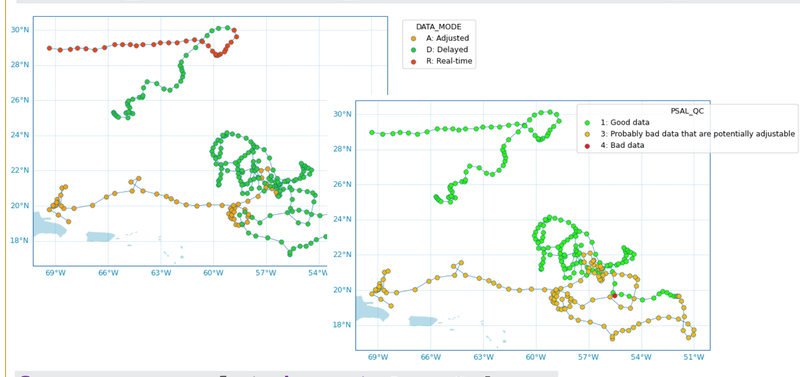
```
from argopy import OceanOPSDeployments
OceanOPSDeployments().to_dataframe()
OceanOPSDeployments([-90, 0, 0, 90]).to_dataframe()
OceanOPSDeployments().plot_status()
```

ADMT documentation (with DOI)

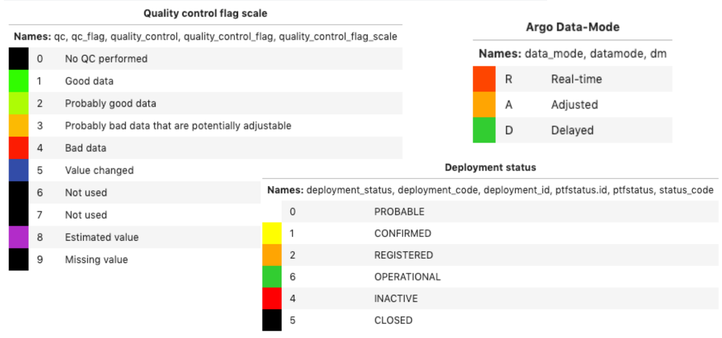
```
from argopy import ArgoDocs
ArgoDocs().list
ArgoDocs(35385)
ArgoDocs(35385).open_pdf(page=12)
ArgoDocs().search('CDOM')
```

Data visualisation

```
from argopy.plot import scatter_map
scatter_map(ds, hue='DATA_MODE')
scatter_map(ds.isel(N_LEVELS=0), hue='PSAL_QC')
```

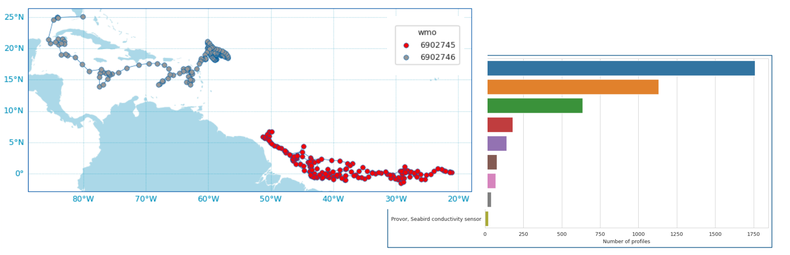



```
from argopy.plot import ArgoColors
ArgoColors('data_mode')
ArgoColors('qc_flag')
ArgoColors('deployment_status')
```



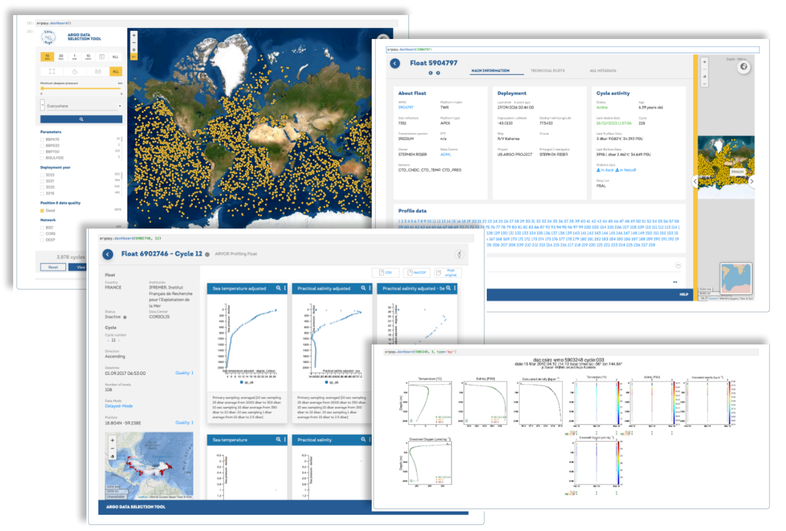
Make plots directly from your data fetchers:

```
fetcher.plot('trajectory')
fetcher.plot('profiler')
```



Easy and direct access to Euro-Argo, BGC, Ocean-Ops, Coriolis and Argovis dashboards:

```
argopy.dashboard()
argopy.dashboard(6902746)
DataFetcher().float(6902746).dashboard()
argopy.dashboard(6902746, 12)
argopy.dashboard(5903248, 3, type='bgc')
```



Data quality control

Download regional topography

```
from argopy import TopoFetcher
TopoFetcher([-65, -55, 10, 20], cache=True).to_xarray()
```

Checkout altimetry test figures from CLS

```
DataFetcher().float(6902745).plot('qc_altimetry')
```

Prepare Data Source files for the OWC analysis

```
ds = DataFetcher(mode='expert').float(6902766).load().data
ds.argo.create_float_source('output_folder')
```

Download core reference data (including password protected CTDs)

```
from argopy import CTDRefDataFetcher
dsref_argo = Datafetcher(src='erddap', ds='ref').region([-65, -55, 10, 20, 0, 5000]).to_xarray()
with argopy.set_options(user='jane_doe', password='****'):
    dsref_ctd = CTDRefDataFetcher([-65, -55, 10, 20, 0, 5000]).to_xarray()
```