Software Package for an Adaptive Satellite-based Sampling for Oceanographic cruises (SPASSO) Technical Document

Version 2.0

L. Rousselet, A.M. Doglioli, A. Petrenko, S. Barillon, F. d'Ovidio

July 19, 2024

Contents

1 Introduction

The horizontal mesoscale and submesoscale circulation variability strongly affects biogeochemical budgets. Therefor it is a real challenge during *in situ* measurements to follow a sampling strategy that will provide a specific representative situation. d'Ovidio et al. (2012) developed several diagnostics based on the study of near-real time altimetry data to map physical structures of biogeochemical interest (fronts, eddy cores, temperature filaments). The analysis of high-resolution satellite-derived data allows for the upstream identification of potential biogeochemical regions to sample or even where phytoplanktonic bloom might occur. This approach has already been successfully used during many campaigns such as Latex10 (2010), KEOPS2 (2011) or even STRASSE (2012) to identify the center of an eddy as the most stable region. Based on these previous works SPASSO has been updated in order to make it available for any oceanographic campaign, (OUTPACE 2015, FUMSECK 2019).

SPASSO automatically and independently download daily satellite-derived data such as surface height, velocity, temperature, salinity and chlorophyll-a concentration. Most of these data are freely distributed by CMEMS with support from CNES (more details in Section 4.6). Data are then mapped over the studied region and diagnostics for lagrangian analysis are computed. SPASSO outputs figures are directly send to user's email and are included in a daily bulletin (.tex file).

The following document describes in detail SPASSO code architecture and functioning.

2 Installation

2.1 Software requirements

SPASSO is a software package designed for Python __version__ < 3.19, >=3.9. It has been tested on Linux and Mac systems and requires the following packages:

- `python3`
- `wget`
- a LaTeX compiler such as `latexmk` or `pdflatex` (for example texlive package)

SPASSO uses various Python libraries that also need to be installed. We highly recommend to use `pip` or `pip3` the official package manager and

command for Python 3 to install all required Python packages. `pip3` should be automatically installed with Python 3 but you can also install it manually. Alternatively, you can also use and install `pip`. To install Python packages then open a terminal window and type:

```
pip3 install <package_name>
```

or

```
python -m pip install <package_name>
```

The required Python packages are:

- `termcolor`
- `pandas`
- `netCDF4`
- `motuclient`
- `matplotlib`
- `cmocean`
- `scipy`
- `xarray`
- `basemap`: (might require `pyproj`, `pyshp`, `geos`)
- `basemap-data-hires`
- `tabulate`
- `simplekml`
- `requests`
- `importlib-metadate`
- `pylatex`
- `copernicusmarine`

The installation of Basemap package can be troublesome for Mac users. Here is an other solution using conda environment (more instructions here):

```
conda create –name myenv
conda activate myenv
conda install numpy
conda install matplotlib
conda install cython
conda install -c conda-forge scipy
conda install -c conda-forge basemap
conda install -c conda-forge glob2
conda install -c conda-forge netcdf4
conda install -c conda-forge basemap-data-hires
```

Similarly all required Python packages listed above must be installed in the conda environment.

2.2 Download and update

SPASSO can be downloaded through Github. Open a terminal and type:

```
git clone...
```

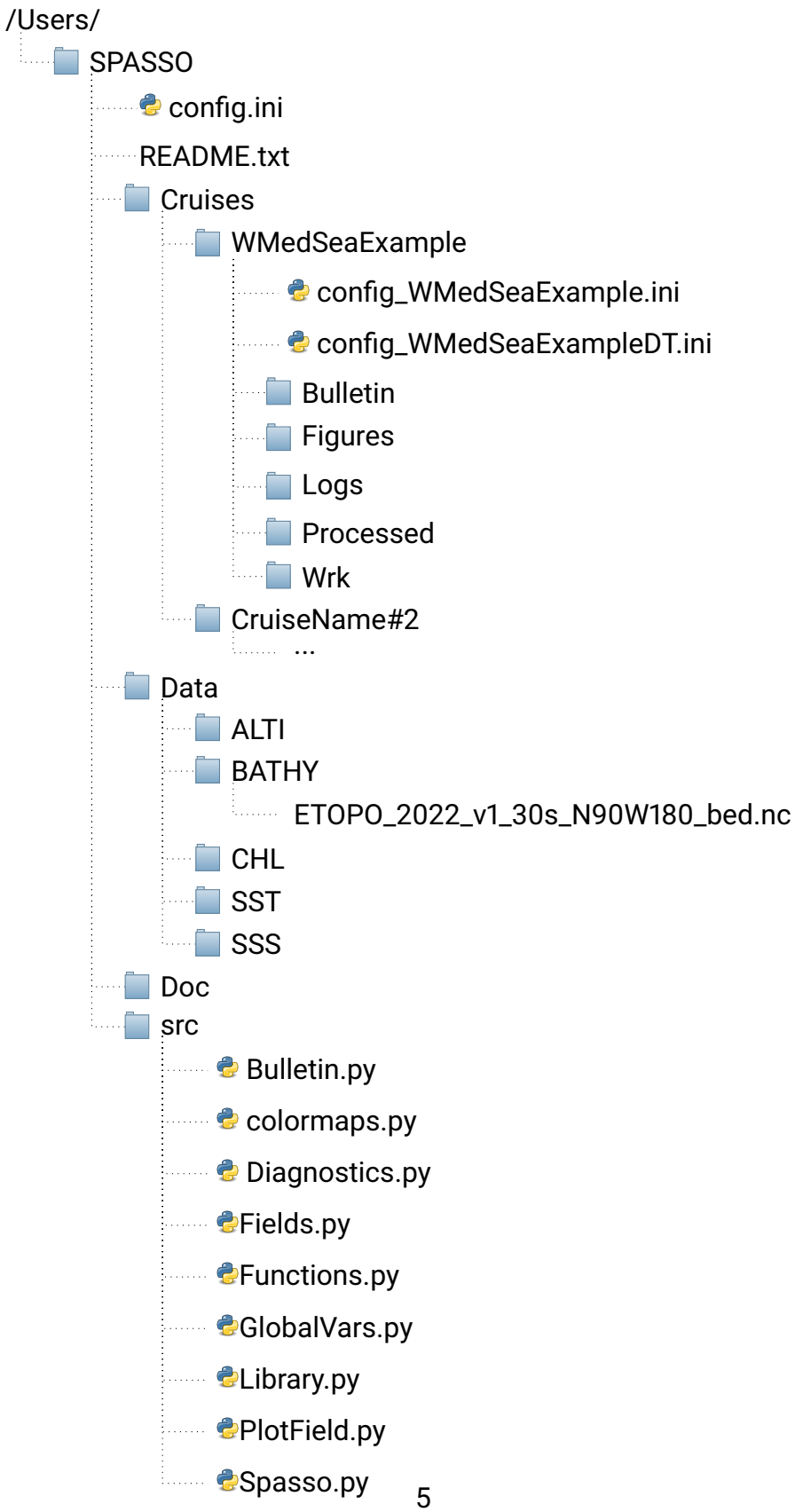
To update the latest version type:

```
git pull...
```

3 Directories and contents

3.1 Directory tree

The software package is organized as follows:


/Users/
SPASSO
config.ini
README.txt
Cruises
WMedSeaExample
config_WMedSeaExample.ini
config_WMedSeaExampleDT.ini
Bulletin
Figures
Logs
Processed
Wrk
CruiseName#2
...
Data
ALTI
BATHY
ETOPO_2022_v1_30s_N90W180_bed.nc
CHL
SST
SSS
Doc
src
Bulletin.py
colormaps.py
Diagnostics.py
Fields.py
Functions.py
GlobalVars.py
Library.py
PlotField.py
Spasso.py

-`Cruises/`: contains the different cruise directories
    -`CruiseName#1` : contains the initialization file (see Section 4.1) and all SPASSO outputs organized in the corresponding directories

-`Data/`: contains the data files organized in the corresponding directories built similarly to CMEMS ftp.
    `BATHY/`: must contain a NETCDF file including global bathymetry that can be downloaded from NOAA.
-`Doc/`: contains user manual and useful documents such as references.

-`src/`: contains all source files:
    -`Bulletin.py`: code preparing a tex and pdf bulletin file including all maps computed with SPASSO.
    -`colormaps.py`: contains some useful colormaps.
    -`Diagnostics.py`: code computing Eulerian and Lagrangian diagnostics (see 3.2).
    -`Fields.py`: code to download, load and create netcdf for all the fields used by SPASSO such as satellite, diagnostics and bathymetry fields.
    -`Functions.py`: containing some additional functions such as one to compute climatologies.
    -`GlobalVars.py`: code loading into memory some dictionaries containing all the global variables such as directory paths, dates, diagnostic parameters, emailing parameters, and bulletin parameters.
    -`Library.py`: code containing SPASSO run functions to print messages, write in a log file, send email, copy files, clean directories, execute shell requests.. .
    -`PlotField.py`: code mapping data defined in the initialization file.
    -`Spasso.py`: main code launching SPASSO software.

3.2 Computed maps and diagnostics

SPASSO software computes maps of various satellite-derived data such as Sea Surface Height (SSH), surface currents, Sea Surface Temperature (SST), Sea Surface Salinity (SSS) and surface ocean color (Chlorophyll-a concentration). These data are freely available in near-real-time or in delayed-time mode through Copernicus Marine Data Store (CMEMS, `https://marine.copernicus.eu/`). The software only requires a user name and password, achieved after CMEMS registration, to be listed in the initialization file (see in Section 4.1). The software can also use data that are not freely available as long as the user provide a ftp command with user and password. An

example is provided in Section 4.6 for Collecte Localisation Satellites (CLS, https://www.cls.fr/en/) data.

Eulerian and Lagrangian diagnostics are also computed using the satellite-derived surface velocity fields and Lagrangian particle trajectories advected with the latter. The diagnostics include Kinetic Energy (KE); Okubo-Weiss parameter (OW); finite-time Lyapunov exponents (FTLE); advection of longitude, latitude and various tracers such SST, SSS or patches; time, longitude and latitude from a bathymetric level (TIMEFROMBATHY); retention parameter (OWTRAJ). More details about the diagnostics mathematical computation and use can be found in the associated document (in prep!).

4 Running the code

4.1 Initialization file: config_*.ini

Before launching SPASSO the user needs to create a cruise directory in `Cruises/` and to create an initialization file `config_*.ini` containing all the cruise parameters. One example of `config_*.ini` is provided in `Cruises/Example/`. The following details all the items contained in the `config_*.ini`.

##
######## Section for main SPASSO options for specific cruise
`[cruises]`
`cruise` = cruise name must be identical as cruise name used for the directory
`mode` = SPASSO near-real-time or delayed time mode (`NRT` or `DT`)
 # The following parameters are used only in DT mode
`refdate` = date(s) for SPASSO maps and diagnostics. Several dates can be set with YYYY-MM-DD format and must be separated by a ",".
`dtmode` = mode for delayed-time outputs: SPASSO can run for each date listed in `refdate` individually (`ind`), i.e. outputs include maps for each of these dates or SPASSO can compute maps for each date between a two dates (`range`).

```
# Examples:
refdate = 2022-01-21,2022-05-14,2022-06-03
dtmode = ind
    -> SPASSO only loads 01/21, 05/14 and 06/03 data

refdate = 2021-04-21,2021-05-14
dtmode = range
    -> SPASSO loads all data between 04/21 and 05/14
```

outmode = outputs mode set to daily for daily figures or to clim to compute a climatology of each satellite products. Computing climotologies requires dtmode to be set to range.
d0 = option to select data file if multiple exist (d0 = first produced or d1 = second produced in delayed-time).

######## User and password section
[userpwd]
userCMEMS = personal CMEMS user name
pwdCMEMS = personal CMEMS password

######## Satellite products section
[products]
products = uppercase abbreviated names, separated by "," of satellite products to run in SPASSO. For each satellite product the following parameters are required to be set:
***abbreviated name*prod** = class name of the product used in Fields.py
path_*abbreviated name* = ftp path to the product
***abbreviated name*prod_date** = CMEMS production date in near-real-time. It can be set to 0_ago, 1_ago, 3_ago or 6_ago. Setting this parameter to 0_ago means the product date is identical to production date; 1_ago means the product date corresponds to the previous date of production date etc...
***abbreviated name*_data** = lowercase abbreviated name of the product

```
        # CMEMS altimetry example:
        phyprod = Copernicus_PHY
        path_phy = nrt.cmems-du.eu/Core/SEALEVEL_GLO_PHY_L4_NRT_
OBSERVATIONS_008_046/dataset-duacs-nrt-global-merged-allsat-phy-l4
        phy_date = 0_ago
        phy_data = phy
```

######## Eulerian parameters section
[Eulerian]
diag = abbreviated names of Eulerian diagnostics [1].
products = uppercase abbreviated name of velocity field product
dayv = date for Eulerian diagnostic (YYYY-MM-DD); if set to default then dayv = today in NRT mode or dayv = refdate in DT mode.
loni = longitude range of the domain with lonmin, lonmax.
lati = latitude range of the domain with latmin, latmax.

[1] KE: Kinetic Energy; OW: Okubo-Weiss parameter; or None

delta0 = meridional and latitudinal interval in degrees for output maps.
UVunit = units of u,v product. It can be `m/s` or `cm/s`.

######## Lagrangian parameters section
`[Lagrangian]`
diag = uppercase abbreviated names of Lagrangian diagnostics[2].
products = uppercase abbreviated name of velocity field product
mode = particle trajectory advection mode: `backward` or `forward`.
dayv = date for Eulerian diagnostic (YYYY-MM-DD); if set to `default` then dayv = today in NRT mode or dayv = refdate in DT mode.
loni = longitude range of the domain with lonmin, lonmax.
lati = latitude range of the domain with latmin, latmax.
numdays = number of days for particle advection.
delta0 = meridional and latitudinal interval in degrees for output maps.
PeriodicBC = boolean for periodic boundary conditions (`True` or `False`).
method = method to solve differential equation: Runge-Kutta 1st order: `rk1flat`; Runge-Kutta 4th order: `rk4flat`. Default is set to `rk4flat`.
numstep = number of step for method.
bathyfile = name of file containing bathymetry data.
bathylvl = bathymetric level (negative).
sstprod = uppercase abbreviated name of SST product to use for SST advection.
sstadvd = number of days for SST advections (default set to `3` days).

######## Cruise parameters section
`[cruise_param]`
nb_domain = number of domains.
Lon = longitude range of the domain with lonmin, lonmax
Lat = latitude range of the domain with latmin, latmax.

######## Plotting parameters section
`[plot_param]`
 # below, each vectors should have as much elements as nb_domain
***abbreviated name*min** = min colormap value.
***abbreviated name*max** = max colormap value.
***abbreviated name*unit** = product unit for plot labeling.
 # if product includes u and v vectors:
***abbreviated name*uv** = arrows scale used in quiver plotting.
***abbreviated name*uvstep** = steps to plot u,v vectors (i.e. an arrow is

[2] `FTLE`: Finite-Time Lyapunov Exponents; `LLADV`: Lon/Lat advections; `OWTRAJ`: retention parameter;`TIMEFROMBATHY`: time from bathymetric level; `SSTADV`: SST advection; `None`

plotted each *step* values): stepU,stepV.
 # if product includes multiple variables:
values must be separated by a ",".
Lon/Lat advection example:
lladvmin = -5,-4 (i.e. lon adv min = -5, lat adv min = -4)
lladvmax = 5,4 (i.e. lon adv max = 5, lat adv max = 4)
lladvunit = ΔLon[$^\circ$],ΔLat[$^\circ$]
(i.e. lon adv unit, lat adv unit)

parallels = vector with values for parallels to draw on maps.
meridians = vector with values for meridians to draw on maps.

######## Bulletin parameters section
[bulletin]
authors = name of bulletin authors, separated with a "," if multiple.
acknow = text file name that contains acknowledgments (OPTIONAL).

######## Emailing section
[email]
sender_mail = sender mailing address (set to None remove the emailing option)
receiver_mail = receiver mailing address, separated with a "," if multiple.
smtp_server = sender smtp server address
port = port number
login = sender mailing login[3]
password = sender mailing password
attach = type of files to be attached with email: tex,tar,pdf to attach tex and pdf bulletin, and Figures in a tar.gz.

######## Library paths section
These are optional but highly recommended especially when running cron jobs
[library]
motulib = path of motuclient package
latexcompiler = path of pdflatex

##

Once the config_*.ini is saved, open a terminal window to run SPASSO

[3]Login and password are optional. If server does not need authentication, leave login and password empty.

by typing:

```
cd src/
cr=CruiseName#1
clear; python3 Spasso.py $cr
```

A normal computation of the code should end with the line:

```
########################################
#      Successfully end program on:
#     *date*
#      Thanks for using SPASSO.
########################################
```

For debugging, more information are saved in a log file located in the cruise working directory (`Cruises/CruiseName#1/Wrk/spassolog.txt`). More details about every parameters in the `config_*.ini` file can be found in `spasso2.0/config.ini`.

4.2 Output files and figures

All the SPASSO current run outputs are located in the cruise working directory. The downloaded satellite data and computed diagnostics are saved in netcdf files. They are also copied in `Data/` and `Cruises/CruiseName#1/Processed/`. All figures are in png format and are copied in `Cruises/CruiseName#1/Figures/`. A compressed `*.tar.gz` file is also created with all figures. A bulletin, containing all the figures computed, is also produced in `pdf` and `tex` format and is saved in `Cruises/CruiseName#1/Bulletin/`. The log file is automatically saved with the corresponding running date in `Cruises/CruiseName#1/Logs/`. At the beginning of each new SPASSO run the cruise working directory is cleaned but every file was previously saved in the corresponding directory so no action is required from the user.

4.3 Adding zoomed-in figures

When dealing with long duration campaign, one would want to visualize the situation over the whole sampling region but also more closely around specific areas. This can be done by setting multiple domains in the `config_*.ini` file:

```
[cruise_param]
nb_domain = 2
Lon = -2,10,0,6
Lon = 40,44.5,40,42
```

Longitude and Latitude range for each domain must be declared as is: lonmin1,lonmax1,lonmin2,lonmax2. The user also needs to defined the minimum and maximum values for each plot parameter by similarly add the new limits. Example for Lon/Lat advection:

lladvmin = -5,-4,-3,-2 (i.e. lon adv min1 = -5, lat adv min1 = -4, lon adv min2 = -3, lat adv min2 = -2)

lladvmax = 5,4,3,2 (i.e. lon adv max1 = 5, lat adv max1 = 4, lon adv max2 = 3, lat adv max2 = 2)

lladvunit = ΔLon[$^\circ$],ΔLat[$^\circ$] (since units are the same, only one occurrence is needed).

The last step for setting up multiple domain is to define parallels and meridians to be plotted:

parallels = np.arange(-90,90+1,2);np.arange(-90,90+1,0.5) (note that here domain1 and 2 should be separated by ";")

meridians = np.arange(-180,180+1,2);np.arange(-180,180+1,0.5)

4.4 Adding optional plot on maps

SPASSO produces maps of satellite data only but there are several options to superimpose data on the maps. The following steps describe how to add optional parameters on the maps.

1. Activate and create plot options in config_*.ini:

   ```
   [plot_options]
   options =
   ```

   If **options** is empty then no options are mapped. One example is to plot the position of hydrographic stations using longitude and latitude coordinates.:

   options = stations

   The user then needs to create a section [*section*] with keys corresponding to options to plot.

   ```
   [stations]
   coordlon = -56,-55.5,-55,-54.5
   coordlat = -2,0,1,2.5
   ```

2. Write corresponding code for plot option in `PlotField.py`:

```
class PlotField():
    def Plot(cruise,Field, *args, **kwargs):

        def PeriodicLon(Lon,lon):
            [...]
            return sl,Lonp

        #get param
        GlobalVars.configIni(cruise)
        [...]
        Library.Done(nfig)
```

Listing 1: sample of the original `PlotField.py` code

The `Plot` function in class `PlotField` is the main code that plots every data on maps. It may contain subfunctions such as `PeriodicLon` as shown in List 1. Each new plot option defined in `config_*.ini` should be added as a new subfunction of `Plot`. In this way the user only needs to add a new "block" of code without modifying the code core of the original `Plot` function. This is illustrated in Listing 2 with the example of stations option (red-shaded box). All new subfunction must take the object `mymap` as an argument since it contains all the maps parameter. Several subfunctions are already coded in `PlotField.py`, such as `waypoints` plotting cruise route or `cities` plotting the location and name of a remarkable city in the sampling region, and can be used as example for new options.

3. Run SPASSO using the usual command.

```
class PlotField():
    def Plot(cruise,Field, *args, **kwargs):

        def stations(mymap):
            lon_stat = [float(x) for x in \
              GlobalVars.config.get('stations','coordlon').split(',')]
            lat_stat = [float(x) for x in \
              GlobalVars.config.get('stations','coordlat').split(',')]
            x_stat,y_stat = mymap(lon_stat,lat_stat)
            mymap.plot(x_stat,y_stat,'-',color='k',zorder=1)
            mymap.scatter(x_stat,y_stat,s=8,color='k',zorder=1)
            return

        def PeriodicLon(Lon,lon):
            [...]
            return sl,Lonp

        #get param
        GlobalVars.configIni(cruise)
        [...]
        Library.Done(nfig)
```

Listing 2: sample of `PlotField.py` code with plot option `stations`

4.5 Adding optional maps output format

SPASSO output maps are automatically saved by default in `png` and `nc` file format. Sometimes it can be useful to save SPASSO figures in other file format. For example, the `kml/kmz` format allows for visualization in Google Earth, a tool widely used in oceanography. The following steps describe how to activate this option and/or create your own:

1. Activate outputs option in `config_*.ini`:
   ```
   [plot_options]
   outopt =
   ```
   If **`outopt`** is empty then no extra outputs are saved. Hereafter the example for `kml/kmz` file format:
   ```
   outopt = kml
   ```

2. Write corresponding code for output option in `PlotField.py`:

```
class PlotField():
    def Outputs(outp,pltarg):
        '''
        Call sub functions to save figures in other file format.
        [...]
        '''
        # loop over the file formats listed in outopt
        for out in outp:
                    eval(out+"(pltarg)")
```

Listing 3: sample of the original `PlotField.py` code

The `Outputs` function in class `PlotField` is the main code running all the extra output files creation (List 3). It may contain subfunctions (definitions). Each new saving file format defined in `config_*.ini` should be added as a new subfunction of `Outputs`. In this way the user only needs to add a new "block" of code without modifying the code core of the original `Outputs` function. This is illustrated in Listing 4 with the example of saving `kml/kmz` files (red-shaded box). All new subfunction must take the object `pltarg` as an argument since it contains all the plot parameters and data.

```
class PlotField():
    def Outputs(outp,pltarg):
        '''
        Call sub functions to save figures in other file format.
        [...]
        '''

        # definition name here must match the one used in outopt
        def kml(pltarg):
            #how to save kml/kmz file
            [...]
            return

        # loop over the file formats listed in outopt
        for out in outp:
                    eval(out+"(pltarg)")
```

Listing 4: sample of `PlotField.py` code with file format `kml/kmz`

3. Run SPASSO using the usual command.

4.6 Adding new fields

There are many pre-defined fields (satellite or diagnostics) that Spasso can use but the user is free to add new fields. For example if one want to add a new satellite-derived salinity field:

1. Create a new children class in `Fields.py`. If it is a Copernicus field you can add it at the end of the Copernicus data (CMEMS) section. The new class must have the following structure:

```
class Product_name(Load,Create):
   def __init__(self,fname,**kwargs):
        self.fname = fname
        data = GlobalVars.config.get('products','*abbreviated name*_data')
        var = Library.GetVars(data)
        self.data_dir   = var['direct']
        self.lon_name='name of longitude variable in netcdf file'
        self.lat_name='name of latitude variable in netcdf file'
        self.d3_name = 'name of 3rd dimension in netcdf file'
        self.var_name = 'name of variable in netcdf file'
        self.var_units = 'variable units'
        self.cmap = 'colormap code for plotting variable'

    def download():
        # setting global variables to local for a shorter req
        data = GlobalVars.config.get('products', '*abbreviated name*_data')
        var = Library.GetVars(data)

        for nf in range(len(var['date'])):
            # dowloading data in /DATA
            req_wget = "wget ..." request to remotely downlad the data
            Library.execute_req(req_wget)
            #check if file exist
            Library.ExistingFile(file path and name,date)

            # copying data in /Wrk
            req_cp = "cp " + file
            Library.execute_req(req_cp)

        return
```

The new children class has specific attributes defined in `__init__` and "saved" in `self`. These attributes include variable names and paths

that are different for each field. The `self` keywords allows to pass these attributes to parent classes such as `Load` and `Create` defined at the beginning of `Fields.py`. The `__init__` method thus allows for creating new field classes without modifying Load and Create function that are generic for each field. The new children class must have a specific `download` method since it is impossible to create a generic download for every type of fields.

2. Add the new field as a product in `config_*.ini`. In [`products`] add the abbreviated name in item **`products`** list. Define ***`abbreviated name`*`prod`** = **`Product_name`**., **`path_*abbreviated name*`**, ***`abbreviated name`*`prod_date`** and ***`abbreviated name`*`_data`** as described in Section 4.1. Also define the mmin, max value and unit in section `[plot_param]`.

3. Run Spasso.py with the usual command.

The same can be done to define a new diagnostic but no `download` method needs to be defined:

```
class Diagnostic_name(Load,Create):
   def __init__(self,fname,**kwargs):
        self.fname = fname
        self.lon_name='longitude name in netcdf'
        self.lat_name='latitude name in netcdf'
        self.d3_name = '3rd dimension name'
        self.var_name = 'variable name in netcdf'
        self.var_units = 'variable unit'
        self.cmap = 'colormap code for plotting variable'
```

The following Section details how to add a new diagnostic for SPASSO to compute.

4.7 Adding diagnostics

The code to compute Eulerian and Lagrangian diagnostics (`Diagnostics.py`) is embedded in SPASSO but it can be used independently to compute diagnostics in an offline mode. Specific tutorials have been developed and are available through `Jupyter Notebooks` t learn how to use `Diagnostics.py` autonomously: Diagnostics tutorials.

`Diagnostics.py` include two classes (**`Lagrangian`** and **`Eulerian`**) and one child class (**`ParticleSet`**) that inherits the functionality from **`Lagrangian`**

class. If your new diagnostic uses particle trajectories then you need to include it as a new function (`def`) in the `Lagrangian` class:

1. Create and write your own diagnostic: lines 20-23 highlighted in red in the example of Listing 5. Use `self` to access the attributes and methods from `ParticleSet` (i.e. particle variables). If you **do not** need attributes in `self` you can remove it from the argument list of `NEWDIAG`.

2. Add the new diagnostic in the diag list: line 11 of Listing 5. If `self` is **not** used in the previous point then just call `NEWDIAG` on line 11 as is:

   ```
   if i == 'NEWDIAG': dd = NEWDIAG(trjf,**kwargs)
   ```

```
1  class Lagrangian():
2      def diag(self,diag=None,method=None,f=None,**kwargs):
3          [...]
4          if diag != None:
5            for i in diag:
6                if i == 'LLADV': dd = self.LLADV(trjf,**kwargs)
7                if i == 'SSTADV': dd = self.SSTADV(trjf,**kwargs)
8                if i == 'FTLE': dd = self.FTLE(trjf,**kwargs)
9                if i == 'OWTRAJ': dd = self.OWTRAJ(trjf,**kwargs)
10               if i == 'TIMEFROMBATHY': dd = self.TIMEFROMBATHY(trjf,**kwargs)
11               if i == 'NEWDIAG': dd = self.NEWDIAG(trjf,**kwargs)
12               out.append(dd)
13         return out
14
15     [...]
16
17     def TIMEFROMBATHY(self,trjf,**kwargs):
18         [...]
19         return timfbathy
20     def NEWDIAG(self,trjf,**kwargs):
21         #your code to compute new diagnostic. Ex:
22         newdiag = trjf['lonf'] - 2
23         return newdiag
24 class ParticleSet(Lagrangian):
25         [...]
```

Listing 5: sample of `Diagnostics.py` code for `Lagrangian` class.

If your new diagnostic is instead Eulerian then you include it as a new function (`def`) in the `Eulerian` class (Listing 6):

```
class Eulerian():
    def __init__(self,fieldset=None,dayv=None):
        [...]

    def diag(self,diag=None,**kwargs):
        out = []
        if diag != None:
          for i in diag:
              if i == 'KE': dd = self.KE(**kwargs)
              if i == 'OW': dd = self.OW(**kwargs)
              if i == 'EULDIAG': dd = self.EULDIAG(**kwargs)
              out.append(dd)
        return out

    [...]

    def def OW(self,**kwargs):
        [...]
        return OW

    def EULDIAG(self,**kwargs):
        #your code to compute new diagnostic. Ex:
        newdiag = trjf['lonf'] - 2
        return newdiag

def Launch(cruise,approach):
        [...]
```

Listing 6: sample of `Diagnostics.py` code for `Eulerian` class.

5 Testing the code: Case of the western Mediterranean Sea

The following examples tests SPASSO functionality to make sure the code is running properly. They can be used as starting point to create your own run.

5.1 Near-Real-Time run

1. Review `config_WMedSeaExample.ini`

```
cd spasso2.0/Cruises/WMedSeaExample/
vi config_WMedSeaExample.ini
cr=WMedSeaExample
```

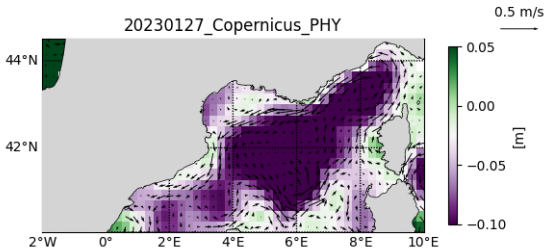
   Run is set in near-real-time mode with 4 satellite products from Copernicus. Fill in **`userCMEMS`** and **`pwdCMEMS`** with your own user and password. Also review `[library]` to fill in with the motuclient library and pdflatex library paths on your machine. Emailing is deactivated by default but you can change accordingly the emailing section to test this functionality.

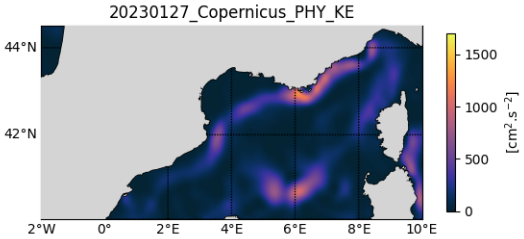
2. Launch `Spasso.py`

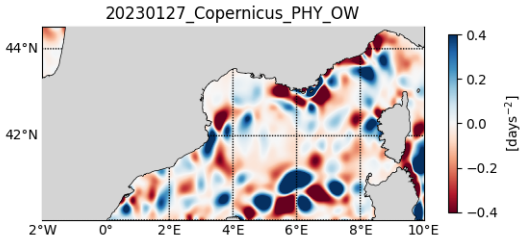
```
cd ../../src/
clear; python3 Spasso.py $cr
```

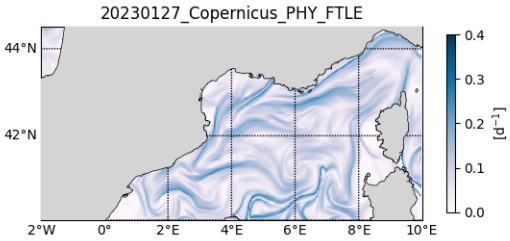
   Run should end in about few minutes with 7 figures, 1 Figures*.tar.gz, 7 netcdf files and one bulletin in pdf and tex including figures produced.

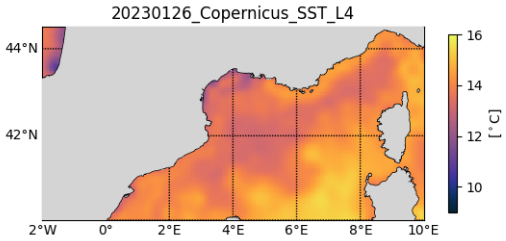
3. Check figures.
   The Lagrangian diagnostic (FTLE) is low-resoluted because of the sparse initial particle seeding and length of advection. You can increase numdays and delta0 to get higher resolution Lagrangian diagnostic. The running time will increase accordingly.

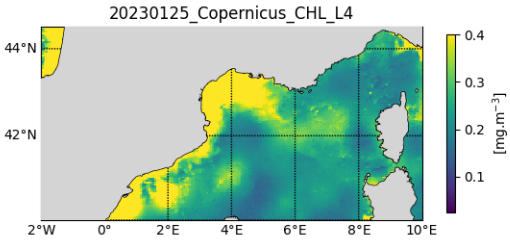

20230127_Copernicus_PHY
0.5 m/s
44°N
42°N
2°W
0°
2°E
4°E
6°E
8°E
10°E
0.05
0.00
−0.05
−0.10
[m]

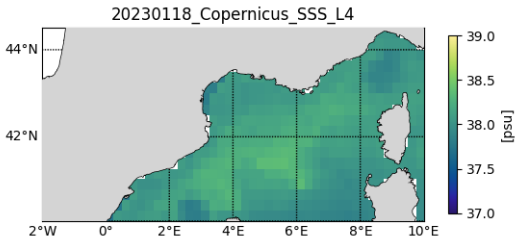

20230127_Copernicus_PHY_KE
44°N
42°N
2°W
0°
2°E
4°E
6°E
8°E
10°E
1500
1000
500
0
[cm².s⁻²]


20230127_Copernicus_PHY_OW
44°N
42°N
2°W
0°
2°E
4°E
6°E
8°E
10°E
0.4
0.2
0.0
−0.2
−0.4
[days⁻²]


20230127_Copernicus_PHY_FTLE
44°N
42°N
2°W
0°
2°E
4°E
6°E
8°E
10°E
0.4
0.3
0.2
0.1
0.0
[d⁻¹]


20230126_Copernicus_SST_L4
44°N
42°N
2°W
0°
2°E
4°E
6°E
8°E
10°E
16
14
12
10
[°C]


20230125_Copernicus_CHL_L4
44°N
42°N
2°W
0°
2°E
4°E
6°E
8°E
10°E
0.4
0.3
0.2
0.1
[mg.m⁻³]



5.2 Delayed-Time run

1. Test 1: Change and review `config_WMedSeaExample.ini`

```
cd spasso2.0/Cruises/WMedSeaExample/
mv config_WMedSeaExample.ini config_WMedSeaExampleNRT.ini
mv config_WMedSeaExampleDT.ini config_WMedSeaExample.ini
cr=WMedSeaExample
```

   Run is now set in delayed-time mode (each date between 01/01/2023 and 03/01/2023) with 3 satellite products from Copernicus. The Eulerian and Lagrangian diagnostics are computed only for 03/01/2023 to save some computation time. If you want to compute the diagnostics for each day defined in **`refdate`** you need to set **`dayv`** to `default` in both sections `Eulerian` and `Lagrangian`. In this example two options are added to plot the station positions and cruise route (see `stations` and `waypoints` in `plot_options`). Fill in **`userCMEMS`** and **`pwdCMEMS`** with your own user and password. Also review `[library]` to fill in with the motuclient library and pdflatex library paths on your machine. Emailing is deactivated by default but you can change accordingly the emailing section to test this functionality.

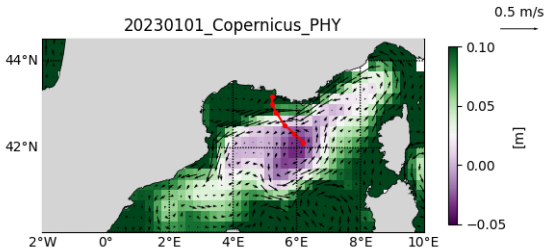
2. Launch `Spasso.py`

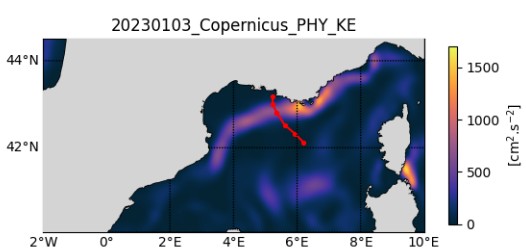
```
cd ../../src/
clear; python3 Spasso.py $cr
```

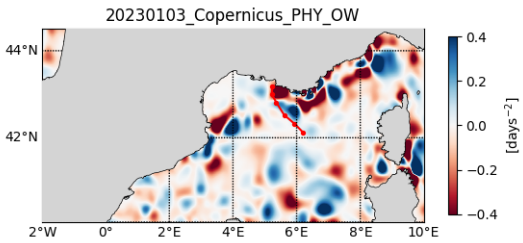
   Run should successfully end in about 5 minutes with 12 figures, one

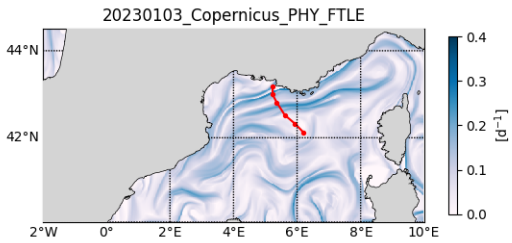
Figures*.tar.gz, 12 netcdf files and one bulletin in pdf and tex including figures produced.

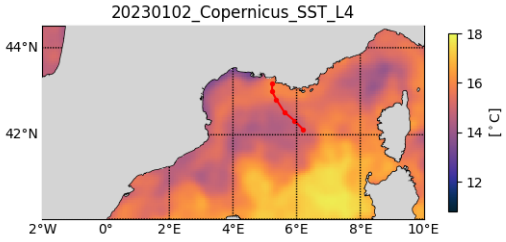
3. Check figures.

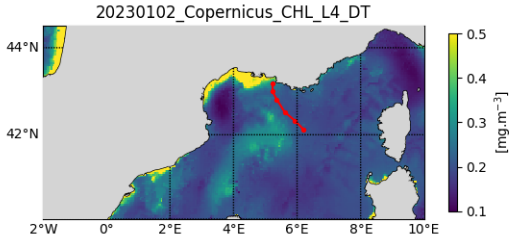












4. Test 2: Modify output mode for climatology and launch `Spasso.py` again.

```
vi ../Cruises/WMedSeaExample/config_WMedSeaExample.ini
line 35: outmode = clim
clear; python3 Spasso.py $cr
```

   Run should end in about few minutes with 6 figures, one Figures*.tar.gz, 7 netcdf files and one bulletin in pdf and tex including figures produced. In this new example, SPASSO produces climatologies for all 3 satellite products (diagnostics are unchanged),

5. Check figures (below only new figures).