2014 - 2020 Interreg V-A
Italy - Croatia CBC Programme
Call for proposal 2019 Strategic

MARLESS (MARine Litter cross-border awarenESS and innovation actions)

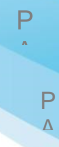
Priority Axis: Environment and cultural heritage; Specific objective: 3.3 - Improve the environmental quality conditions of the sea and coastal area by use of sustainable and innovative technologies and approaches

3.3.4 Model for the calculation of the trajectories of floating objects released into the sea

Activity 3.3

WP 3

Version: FINAL
Distribution: PUBLIC
Date: 29/06/2023

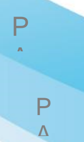


PROJECT MARLESS

Work Package:	WP3 Monitoring optimization
Activity:	Activity 3.3 Marine litter hot spots identification
WP Leader:	PP2
Deliverable:	D3.3.4 Model for the calculation of the trajectories of floating objects released into the sea

Version:	Final	Date:	29/06/2023
Type:	Report		
Availability:	Public		
Responsible Partner:	PP2		
Involved Partner	ARPA FVG		
Editor:	ARPA FVG		
Contributors:	/		

DISCLAIMER: PP2 reflects the project MARLESS views; the IT-HR Programme authorities are not liable for any use that may be made of the information contained therein.



Model for the calculation of the trajectories of floating objects released into the sea

Activity Deliverable (3.3.4)

WP3
Activity 3.3

Deliverable: D.3.3.4
Version: final
Confidential level: Partnership
Date of release: 29/06/2023

Introduction

The deliverable D3.3.4 is composed by three outputs: the identification of the model, the documentation of its features and its implementation; the development and the computational implementation of the model on the High Performance Computation infrastructure; the simulation of marine litter trajectories and back trajectories in the Adriatic Sea.
After the identification of the model, the expected spatial resolution has to be computed and then all the simulations has to be performed considering the advice of the PPs involved to find the better starting point for the forward and back trajectories. As the forward trajectories have been performed massively into the deliverable D3.3.2, in this document we are going to focus on the back trajectories.
The most suitable candidate for this work is the Parcels model.
On the first part of the document, the model features are presented, then on the second part its implementation and some issues and tests are shown. On the third part the tests of the backtrajectories are presented with the results obtained for some specific beaches.

Parcels (Probably A Really Computationally Efficient Lagrangian Simulator)

Parcels is a model developed by the OceanParcels project. It consists of a set of Python classes and methods to create customisable particle tracking simulations using output from Ocean Circulation models. The model is flexible, enables wide range of applicability and allows to build complex simulations.
The code is licensed under an open source MIT (Massachusetts Institute of Technology) license.
Parcels model has been chosen since it is a new tool for the Lagrangian particle trajectories framework, with a community that works on the model and that has provided some tutorials to help the new users, available in:
https://mybinder.org/v2/gh/OceanParcels/parcels_examples_binder/master?urlpath=lab/tree/parcels_examples/parcels_tutorial.ipynb

The model documentation is available in:
https://oceanparcels.org/gh-pages/html/. This documentation is equipped with useful tutorial examples.

Documentation of the model and its features

Below the documentation and the features of the model are presented.

General Parcels structure [1]

It is a good practice to separate the simulation code into the following sections/classes (even if the simulation is complex, it is good to keep these different steps separate to keep a clear overview and find bugs in an easier way):

- **FieldSet**: load and set up the fields necessary to the movement of particles.
  The class requires at least the 2D hydrodynamic data that will move the particles (the 'U' and 'V' wind velocity, but it can be added any other variable). The general method to use is *FieldSet.from_netcdf*, which requires filenames, variables and dimensions.

```
fname = 'GlobCurrent_example_data/*.nc'
filenames = {'U': fname, 'V': fname}
variables = {'U': 'eastward_eulerian_current_velocity',
             'V':'northward_eulerian_current_velocity'}
dimensions = {'U': {'lat': 'lat', 'lon': 'lon', 'time': 'time'},
              'V': {'lat': 'lat', 'lon': 'lon', 'time': 'time'}}
fieldset = FieldSet.from_netcdf(filenames, variables, dimensions)
```

  The fields can be in different grids type:

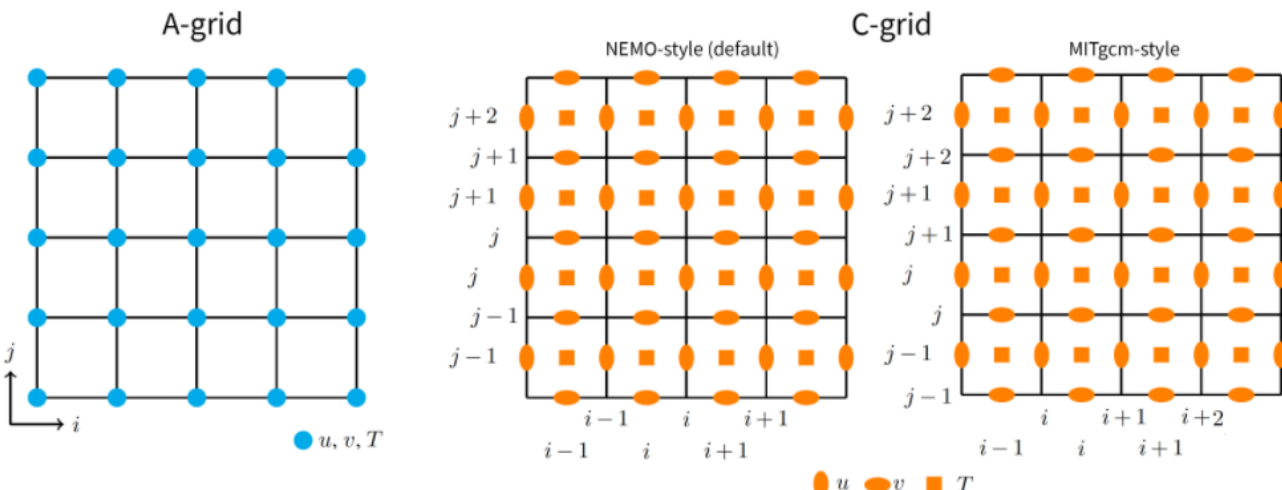


Figure 1: grids type. Image taken from [2]

The unstructured grids are not yet supported in Parcels [3], so it is not possible to use the high resolution currents as in the deliberable D3.3.2.
On a Curvilinear grid, determining the location of each Particle on the grid is more complicated and therefore takes longer than on a Rectilinear grid. A function is available on the ParticleSet class, that speeds up the look-up. After creating the ParticleSet, but before running the ParticleSet.execute(), you can simply call the function ParticleSet.populate_indices().

```
pset = ParticleSet.from_list(field_set, JITParticle, lon=lonp, lat=latp)
pset.populate_indices()
```

- **ParticleSet:** define the type of particles you want to release, what Variables they have and their initial conditions. This object requires: the *FieldSet* on which the particles live; the type of Particle, that contains the information stored by each particle; the initial conditions for each Variable defined in the Particle (e.g. the release locations in lon and lat).

The different Particle types available are the *JITParticle* and the *ScipyParticle*, but it is very easy to create particle class which includes other Variables.

```
pset = ParticleSet(fieldset=fieldset,   # the fields on which the particles are advected
                   pclass=JITParticle,  # the type of particles (JIT- or Scipy-Particle)
                   lon=28,     # release longitudes  lat=-33)          # release latitudes
```

- **Execute kernels**: define and compile the kernels that encode what the particles need to do each timestep and execute them.
  Running a simulation in parcels means executing kernels, little snippets of code that are run for each particle at each timestep. The most basic kernels are the advection kernels which calculate the movement of each particle based on the FieldSet in which the ParticleSet lives.
  To store the particle data generated in the simulation, it is necessary to define the *ParticleFile* to which the output of the kernel execution will be written. Once the *ParticleSet* it is defined, the method *ParticleSet.execute()* can be executed; this method requires the following arguments:
  1. the kernels (which define how particles move) to be executed;
  2. the runtime defining how long the execution loop runs (the total length of the run in seconds), alternatively, the endtime at which the execution loop stops can be defined;
  3. the timestep *dt* at which to execute the kernels;
  4. optionally, the *ParticleFile* object to write the output to.

```
output_file = pset.ParticleFile(name="GCParticles.nc", outputdt=3600) # the file
                                                      name and the time step of the outputs
pset.execute(AdvectionRK4,            # the kernel (which defines how particles move)
             runtime=86400*6,         # the total length of the run
             dt=300,                  # the timestep of the kernel
             output_file=output_file)
```

Kernels

One of the most powerful features of Parcels is the ability to write custom Kernels. However, there are some key limitations in writing kernels on your own:

- every Kernel must be a function with only the following arguments (particle, fieldset, time);
- In order to run successfully in JIT mode, Kernel definitions can only contain the following types of commands: basic arithmetical operators and assignments; basic logical operators; if and while loops and break statements.

There are some collections of pre-built kernels that are shown in the following sections.

Collection of pre-built advection kernels:

1. ***AdvectionAnalytical(particle, fieldset, time)*****:** advection of particles using 'analytical advection' integration: it is based on Ariane/TRACMASS algorithm (see more information in [4]): the time-dependent scheme is implemented with 'intermediate timesteps' (default 10 per model timestep) and not yet with the full analytical time integration.
   The analytical scheme works with a few limitations: the velocity field should be defined on a C-grid; it works only for Scipy Particles; since it does not use timestepping, the dt parameter in pset.execute() should be set to np.inf for forward-in-time simulations and to -np.inf for backward-in-time ones; for time-varying fields, only the 'intermediate timesteps' scheme is implemented. Tutorial in [5].
2. ***AdvectionEE(particle, fieldset, time)*****:** advection of particles using Explicit Euler (Euler Forward) integration. This function needs to be converted to Kernel object before execution.
3. ***AdvectionRK4(particle, fieldset, time):*** advection of particles with fourth-order Runge-Kutta integration. This function needs to be converted to Kernel object before execution.
4. ***AdvectionRK45(particle, fieldset, time)*****:** advection of particles using adaptive Runge-Kutta 4/5 integration. The Times-step dt is halved if the error is larger than tolerance,

and doubled if the error is smaller than 1/10th of tolerance (tolerance set to 1e-5 * dt by default).

5. ***AdvectionRK4_3D(particle, fieldset, time)***: advection of particles using fourth-order Runge-Kutta integration including vertical velocity. This function needs to be converted to Kernel object before execution.

In Figure 2, a schematic comparison between the Analytical Advection scheme and the fourth order Runge-Kutta scheme is shown.

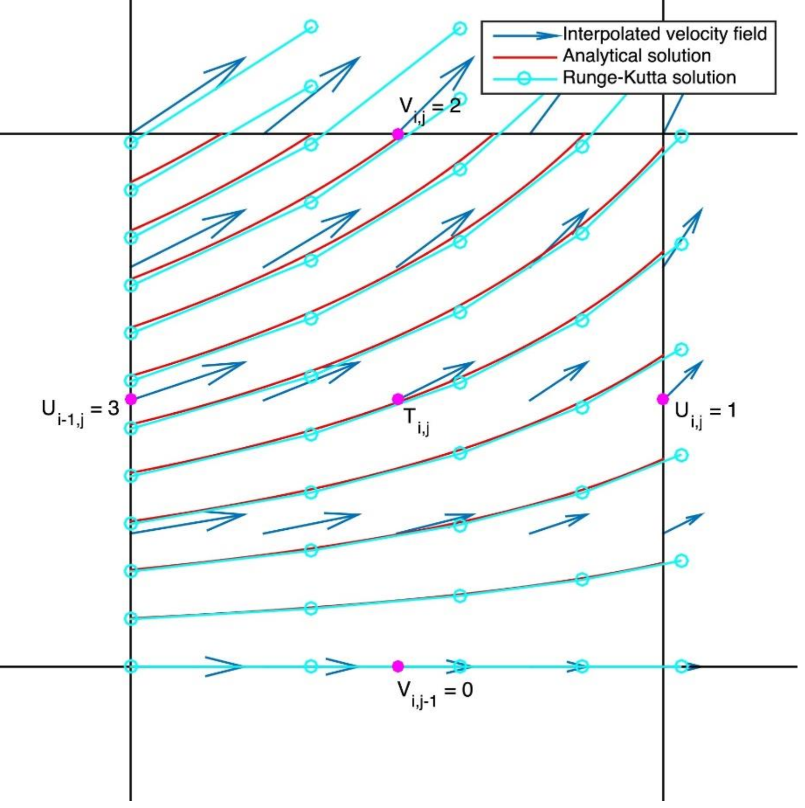


Figure 2: Illustration of time stepping solutions on an Arakawa C-grid with edges of non-dimensional length=1. Velocities (u, v) across the four edges are given in numbers at the magenta dots. The blue arrows are the linearly interpolated velocities within the grid.

Particles are released on the (left) edge. The red lines are pathlines of the analytical solution for these particles. The cyan piecewise linear lines are the solutions to RK4 timestepping with dt=0.1 . The two types of integration lead to similar solution. Image taken from [6].

Collection of pre-built advection-diffusion kernels (tutorial in [7]):

1. ***AdvectionDiffusionEM(particle, fieldset, time)*****:** 2D advection-diffusion solved using the Euler-Maruyama scheme (EM). It assumes that fieldset has as fields *Kh_zonal* and *Kh_meridional* and variable fieldset.dres, setting the resolution for the central difference gradient approximation (this should be of the order of the local gridsize). The Euler-Maruyama scheme is of strong order 0.5 and weak order 1. The Wiener increment dW is normally distributed with zero mean and a standard deviation of sqrt(dt).
2. ***AdvectionDiffusionM1(particle, fieldset, time)*****:** 2D advection-diffusion solved using the Milstein scheme at first order (M1). It assumes that fieldset has fields *Kh_zonal* and *Kh_meridional* and variable fieldset.dres, setting the resolution for the central difference gradient approximation (this should be of the order of the local gridsize). This Milstein scheme is of strong and weak order 1, which is higher than the Euler-Maruyama scheme. It experiences less spurious diffusivity by including extra computationally cheap correction terms. The Wiener increment dW is normally distributed with zero mean and a standard deviation of sqrt(dt). The AdvectionDiffusionM1 kernel should be the default choice, instead of the previous kernel, as the increased accuracy comes at negligible computational cost.
3. ***DiffusionUniformKh(particle, fieldset, time)*****:** simple 2D diffusion where diffusivity (Kh) is assumed uniform. Assumes that fieldset has constant fields *Kh_zonal* and Kh_meridional. The constant fields can be added through: *fieldset.add_constant_field("Kh_zonal", kh_zonal, mesh=mesh)* and *fieldset.add_constant_field("Kh_meridional", kh_meridional, mesh=mesh)*

where mesh can be either 'flat' or 'spherical', latitude and longitude in meters or in degree respectively. This kernel, assuming null diffusivity gradients, is more efficient. Since the perturbation due to diffusion is isotropic independent, this kernel contains no advection and can be used in combination with a separate advection kernel. The Wiener increment dW is normally distributed with zero mean and a standard deviation of sqrt(dt).

The advection component of the first two kernels presented above is similar to that of the Explicit Euler advection kernel (AdvectionEE). If the diffusivity is constant over the entire domain, the diffusion-only kernel DiffusionUniformKh can be used in combination with an advection kernel of choice.
Advantages of the method of solving the advection-diffusion equation is that it is accurate at big time scales and large areas. The method also has two big disadvantages: it is not always mass conserving and for high concentration gradients, negative concentrations can occur. [8]
The random walk models are mass conserving, because individual particles are modeled, which is an advantage. A disadvantage is that a really large number of particles have to be calculated, to find an accurate solution. The bigger the area or the timespan, the more particles have to be simulated. Therefore, it is not suitable for large times or big areas. For example, a dumping of plastics from a ship, can be modeled better with the random walk model since this is only a small area or small time, but a large concentration gradient. Instead, to discover a permanent accumulation zone, the advection-diffusion equation is a better approach. Since a large area and timescale is needed to find permanent zones. [8]

Since the diffusivity here is space-independent, gradients are not calculated, increasing the efficiency. The diffusion-step can, in this case, be computed after or before advection.

For computing the gradient in diffusivity, *dres* is needed: the gradients in diffusivities are approximated by using their values at the particle's location ± dres (in both x and y). A value

of dres must be specified and added to the FieldSet (e.g. fieldset.add_constant("dres", 0.01)). Currently, it is unclear what the best value of dres is. From experience, dres should be smaller than the spatial resolution of the data, but within reasonable limits of machine precision to avoid numerical errors.

Other pre-built kernels

Moreover, there are other pre-built kernels that calculate: sea water density; adiabatic temperature, from depth in meters and latitude; potential temperature; temperature from potential temperature at the reference pressure and in situ pressure.

- **Output:** write and store the output to a NetCDF file. While executing the ParticleSet, the model stores the data in npy files in an output folder. To take all the data and store them in a netcdf file, the *ParticleFile.export()* can be used if you want to keep the folder with npy files; or *ParticleFile.close()* if you only want to keep the netcdf file.
  There is some simple basic plotting functionality built into Parcels.

```
output_file.export()      # or output_file.close()
```

Implementation on the HPC infrastructure and preliminary tests

Implementation of the model

Below the trial installation in presented.
Parcels code has been installed using Anaconda and the Parcels Conda-Forge package with its latest release. This package will automatically install (almost) all the requirements for a fully functional installation of the model.
The model has been installed in the C3HPC cluster. Below, the steps necessary to install the model in the Linux environment are shown:

1. Install Anaconda's Miniconda following the steps at https://conda.io/docs/user-guide/install/ selecting the Python-3 version.

2. Start a terminal, activate the root (or base) environment of Miniconda and create an environment containing Parcels, all its essential dependencies, and the nice-to-have Jupyter, cartopy, and ffmpeg packages:

```
conda activate root
conda create -n py3_parcels -c conda-forge parcels jupyter cartopy ffmpeg
conda install -n py3_parcels pytest   # if necessary
```

3. Activate the newly created Parcels environment:

```
conda activate py3_parcels
```

To verify the correct installation of the model, it is required to get a copy of the Parcels tutorials and examples and the data required:

```
parcels_get_examples parcels_examples
```

and then, run the simplest of the examples:

```
cd parcels_examples
python example_peninsula.py --fieldset 100 100
```

Preliminary tests, issues and trial of the implementation of some algorithms

In this section some preliminary tests and issues are shown together with the trial of the implementation of some algorithms (i.e. the beaching and resuspension one and the windage one).

Test JIT (Just-In-Time compilation) VS Scipy mode

JIT is almost always faster than the Scipy mode. By the way, Scipy is easier to debug when writing custom kernels, so can provide faster development of new features. If you want to run Parcels in Scipy mode there are ways to make Parcels a bit faster shown in this link:

https://nbviewer.jupyter.org/github/OceanParcels/parcels/blob/master/parcels/examples/tutorial_jit_vs_scipy.ipynb (If you want to run in Scipy mode add particle at the end of your Field sampling).

Time test

In this test, with the Peninsula dataset available in the tutorial, the measure of the computational time is performed using *timer* module for the particle type JIT and Scipy (see tutorial tutorial_jit_vs_scipy.ipynb). The test is performed changing only the number of particles and not the time parameters and the advection kernel.

- For 100 particles:

```
        (100%)  Timer root              : 1.160e+01 s
( 1%)   ( 1%) Timer fieldset creation : 1.583e-01 s
( 80%)  ( 80%) Timer scipy            : 9.294e+00 s
( 19%)  ( 19%) Timer jit              : 2.147e+00 s
```

- For 1000 particles:

```
        (100%)  Timer root              : 1.318e+02 s
( 0%)   ( 0%) Timer fieldset creation : 1.499e-01 s
( 97%)  ( 97%) Timer scipy            : 1.282e+02 s
( 3%)   ( 3%) Timer jit               : 3.436e+00 s
```

- For 10000 particles:

```
        (100%)  Timer root              : 0:17:10.689796
( 0%)   ( 0%) Timer fieldset creation : 0:00:00.097088
(100%)  (100%) Timer scipy            : 0:17:07.498055
( 0%)   ( 0%) Timer jit               : 0:00:03.074338
```

Test: how to combine different Fields for advection into a 'SummedField' object

If you want to advect particles using a combination of different velocity data sets (for example a combination of currents and winds) one option would be to write a Kernel that computes the movement of particles due to each of these flows. However, it is possible to directly combine

different flows (without interpolation) and feed them into the built-in AdvectionRK4 kernel using the SummedField objects.
In the following code zonal and meridional velocity field on a 1kmx1km grid with a flat mesh are defined (the zonal velocity is uniform and 1 m/s, and the meridional velocity is zero everywhere).

```
xdim, ydim = (10, 20)
Uflow = Field('U', np.ones((ydim, xdim), dtype=np.float32),
lon=np.linspace(0., 1e3, xdim, dtype=np.float32),
 lat=np.linspace(0., 1e3, ydim, dtype=np.float32))
Vflow = Field('V', np.zeros((ydim, xdim), dtype=np.float32), grid=Uflow.grid)
fieldset_flow = FieldSet(Uflow, Vflow)
```

Now, let's define another set of velocities (Ustokes, Vstokes) on a different, higher-resolution grid (this flow is southward at -0.2 m/s).

```
gf = 10  # factor by which the resolution of this grid is higher than of the original one.
Ustokes = Field('U', np.zeros((ydim*gf, xdim*gf), dtype=np.float32),
            lon=np.linspace(0., 1e3, xdim*gf, dtype=np.float32),
            lat=np.linspace(0., 1e3, ydim*gf, dtype=np.float32))
Vstokes = Field('V', -0.2*np.ones((ydim*gf, xdim*gf), dtype=np.float32), grid=Ustokes.grid)
fieldset_stokes=FieldSet(Ustokes, Vstokes)
```

Now we can simply define a FieldSet with a summation of different Fields:

```
fieldset_sum = FieldSet(U = fieldset_flow.U+fieldset_stokes.U,  V=fieldset_flow.V+fieldset_stokes.V)
```

Then, we can proceed with the ParticleSet structure and the kernel execution.

Test: how to 'delay' the start of particle advection

If you want to release particles at different times throughout a simulation or at a constant rate from the same set of locations, there are two ways to delay the start: either by defining the whole ParticleSet at initialisation and giving each particle its own time or by using the *repeatdt* argument.

The first and simplest way to delay the start of a particle is to use the time argument for each particle:

```
npart = 10  # number of particles to be released
lon = 3e3 * np.ones(npart)
lat = np.linspace(3e3 , 45e3, npart, dtype=np.float32)
time = np.arange(0, npart) * delta(hours=1).total_seconds()      # release every particle one hour later
pset = ParticleSet(fieldset=fieldset, pclass=JITParticle, lon=lon, lat=lat, time=time)
output_file = pset.ParticleFile(name="DelayParticle_time.nc", outputdt=delta(hours=1))
pset.execute(AdvectionRK4, runtime=delta(hours=24), dt=delta(minutes=5),
        output_file=output_file)
output_file.export()  # export the trajectory data to a netcdf file
```

The second method to delay the start of particle releases is to use the *repeatdt* argument when constructing a ParticleSet. This is especially useful if you want to repeatedly release particles from the same set of locations:

```
npart = 10  # number of particles to be released
lon = 3e3 * np.ones(npart)
lat = np.linspace(3e3 , 45e3, npart, dtype=np.float32)
repeatdt = delta(hours=3)  # release from the same set of locations every 3 hours
pset = ParticleSet(fieldset=fieldset, pclass=JITParticle, lon=lon, lat=lat, repeatdt=repeatdt)
output_file = pset.ParticleFile(name="DelayParticle_releasedt", outputdt=delta(hours=1))
pset.execute(AdvectionRK4, runtime=delta(hours=24), dt=delta(minutes=5),
        output_file=output_file)
```

If you want at some point to stop the repeatdt, the easiest implementation is to use two calls to *pset.execute():*

```
pset = ParticleSet(fieldset=fieldset, pclass=JITParticle, lon=lon, lat=lat, repeatdt=repeatdt)
output_file = pset.ParticleFile(name="DelayParticle_releasedt_9hrs",
outputdt=delta(hours=1))

# first run for 3 * 3 hrs
pset.execute(AdvectionRK4, runtime=delta(hours=9), dt=delta(minutes=5),
    output_file=output_file)

# now stop the repeated release
pset.repeatdt = None

# now continue running for the remaining 15 hours
pset.execute(AdvectionRK4, runtime=delta(hours=15), dt=delta(minutes=5),
    output_file=output_file)

output_file.export()  # export the trajectory data to a netcdf file
```

Test: how to combine different Fields into a NestedField object

The *NestedField* object can be used if you have access to different fields that each covers only part of the region of interest and you have to combine them all together or if you have a field covering the entire region and another one only covering part of it, but with a higher resolution. The model will try to successively interpolate the different fields.
The order of the fields in the NestedField matters. In particular, the smallest/finest resolution fields have to be listed before the larger/coarser resolution fields.

First we define a zonal and meridional velocity field defined on a high resolution (dx = 100m) 2kmx2km grid with a flat mesh. The zonal velocity is uniform and 1 m/s, and the meridional velocity is equal to 0.5 *cos(lon /200 *pi/2) m/s.

```
V1_data = np.cos(lon_g / 200 * np.pi/2)
U1 = Field('U1', np.ones((dim, dim), dtype=np.float32), lon=lon, lat=lat)
V1 = Field('V1', V1_data, grid=U1.grid)
```

Now we define the same velocity field on a low resolution (dx = 2km) 20kmx4km grid.

```
V2_data = np.cos(lon_g / 200 * np.pi/2)
U2 = Field('U2', np.ones((ydim, xdim), dtype=np.float32), lon=lon, lat=lat)
V2 = Field('V2', V2_data, grid=U2.grid)
```

We now combine those fields into a NestedField and create the fieldset:

```
U = NestedField('U', [U1, U2])
V = NestedField('V', [V1, V2])
fieldset = FieldSet(U, V)
```

Test with ROMS current data without wind field

In this first test the nertCDF ROMS current data of the 09/09/2021 are used without the addition of the wind field.

Firstly, the U e V wind are defined together with their dimensions. In the ParticleSet, 5 particles are released in a line that has the start and finish position as above. The particles are advected with the AdvectionRK4 kernel.

```
from parcels import (FieldSet, ParticleSet, Variable, JITParticle, ScipyParticle, AdvectionRK4, plotTrajectoriesFile)
import numpy as np
import math
from datetime import timedelta
from operator import attrgetter

filenames = {'U': "/lustre/arpa/farrisc/GNOME_test/9sett21/GNOME-df_WRF_ROMSfvg9sett21/GNOME-df_WRF_ROMS/roms_20210909_00+72h.nc",
             'V': "/lustre/arpa/farrisc/GNOME_test/9sett21/GNOME-df_WRF_ROMSfvg9sett21/GNOME-df_WRF_ROMS/roms_20210909_00+72h.nc"}
variables = {"U": "u", "V": "v"}
dimensions = {"U": {"time": "time", "depth": "sigma", "lat": "ny", "lon": "nx"},
              "V": {"time": "time", "depth": "sigma", "lat": "ny", "lon": "nx"}}
fieldset0 = FieldSet.from_netcdf(filenames, variables, dimensions, mesh="spherical")
pset = ParticleSet.from_line(fieldset=fieldset0, pclass=ScipyParticle,
                             size=5,            # releasing 5 particles
                             start=(13.0,45.5),   #
                             finish=(13.1, 45.5))  #
output_file0 = pset.ParticleFile(name="tutorial1.nc", outputdt=timedelta(hours=1))
pset.execute(AdvectionRK4,
             runtime=timedelta(days=2),
             dt=timedelta(minutes=15),
             output_file=output_file0)
output_file0.export()
```

ERROR using pclass=ScipyParticle

```
Exception ignored in: <function ParticleFileSOA.__del__ at 0x2b85a08208b0>
Traceback (most recent call last):
  File "/u/arpa/farrisc/miniconda3/envs/py3_parcels/lib/python3.9/site-packages/parcels/particlefile/particlefilesoa.py", line 40, in __del__
  File "/u/arpa/farrisc/miniconda3/envs/py3_parcels/lib/python3.9/site-packages/parcels/particlefile/baseparticlefile.py", line 235, in __del__
  File "/u/arpa/farrisc/miniconda3/envs/py3_parcels/lib/python3.9/site-packages/parcels/particlefile/baseparticlefile.py", line 240, in close
  File "/u/arpa/farrisc/miniconda3/envs/py3_parcels/lib/python3.9/site-packages/parcels/particlefile/particlefilesoa.py", line 125, in export
TypeError: 'NoneType' object is not callable
```

ERROR using pclass=JITParticle

```
INFO: Compiled ArrayJITParticleAdvectionRK4 ==> /tmp/parcels-19004/libf508b41eaa796e8d4e4c7d407145d1fd_0.so
Exception ignored in: <function ParticleFileSOA.__del__ at 0x2abc4a80a8b0>
```

```
Traceback (most recent call last):
  File "/u/arpa/farrisc/miniconda3/envs/py3_parcels/lib/python3.9/site-
packages/parcels/particlefile/particlefilesoa.py", line 40, in __del__
  File "/u/arpa/farrisc/miniconda3/envs/py3_parcels/lib/python3.9/site-
packages/parcels/particlefile/baseparticlefile.py", line 235, in __del__
  File "/u/arpa/farrisc/miniconda3/envs/py3_parcels/lib/python3.9/site-
packages/parcels/particlefile/baseparticlefile.py", line 240, in close
  File "/u/arpa/farrisc/miniconda3/envs/py3_parcels/lib/python3.9/site-
packages/parcels/particlefile/particlefilesoa.py", line 125, in export
TypeError: 'NoneType' object is not callable
WARNING: compiled library already freed.
Exception ignored in: <function KernelSOA.__del__ at 0x2abc4a7f00d0>
Traceback (most recent call last):
  File "/u/arpa/farrisc/miniconda3/envs/py3_parcels/lib/python3.9/site-
packages/parcels/kernel/kernelsoa.py", line 157, in __del__
  File "/u/arpa/farrisc/miniconda3/envs/py3_parcels/lib/python3.9/site-
packages/parcels/kernel/basekernel.py", line 99, in __del__
  File "/u/arpa/farrisc/miniconda3/envs/py3_parcels/lib/python3.9/site-
packages/parcels/kernel/basekernel.py", line 199, in remove_lib
  File "/u/arpa/farrisc/miniconda3/envs/py3_parcels/lib/python3.9/site-
packages/parcels/kernel/basekernel.py", line 220, in get_kernel_compile_files
  File "/u/arpa/farrisc/miniconda3/envs/py3_parcels/lib/python3.9/site-
packages/parcels/tools/global_statics.py", line 35, in get_cache_dir
  File "/u/arpa/farrisc/miniconda3/envs/py3_parcels/lib/python3.9/pathlib.py",  line 1072, in
__new__
  File "/u/arpa/farrisc/miniconda3/envs/py3_parcels/lib/python3.9/pathlib.py",  line 697, in
_from_parts
  File "/u/arpa/farrisc/miniconda3/envs/py3_parcels/lib/python3.9/pathlib.py",  line 678, in
_parse_args
```

TypeError: isinstance() arg 2 must be a type or tuple of types

First problem solved

The error has been solved replacing the output_file definition inside the pset.execute() command.

```
#output_file0 = pset.ParticleFile(name="tutorial1.nc", outputdt=timedelta(hours=1))
#pset.execute(AdvectionRK4,
#           runtime=timedelta(days=2),
#           dt=timedelta(minutes=15),
#           output_file=output_file0,
#          recovery={ErrorCode.ErrorOutOfBounds: DeleteParticle})

pset.execute( AdvectionRK4 , runtime=timedelta(days=3), dt=timedelta(minutes=15),
  output_file=pset.ParticleFile(name='atutorial1.nc', outputdt=timedelta(minutes=15)),
            recovery={ErrorCode.ErrorOutOfBounds: DeleteParticle})
```

Second problem solved

We decided to use the original ROMS netcdf output.

The Particle were not advected, the problem has been solved defining in a different way the lat and lon variables: instead of using the dimension nx and ny the model needs the variables defined as ‘lat’ and ‘lon’ that are both function of nx and ny.

```
import numpy as np
import xarray as xr
import math
from operator import attrgetter
from parcels import FieldSet, AdvectionAnalytical, ParticleSet, JITParticle, ScipyParticle
from parcels import AdvectionRK4, ErrorCode, AdvectionDiffusionEM, plotTrajectoriesFile
from datetime import timedelta

def DeleteParticle(particle, fieldset, time):
    particle.delete()

data_file = "/lustre/arpa/farrisc/roms/roms_20211001.nc"
filenames = {'U': {'lon': data_file, 'lat': data_file, 'data': data_file},
             'V' :{'lon': data_file, 'lat': data_file, 'data': data_file}}

variables = {"U": "u", "V": "v"}
dimensions = {'U': {'lon': 'lon', 'lat': 'lat',  'time': 'time'},
              'V': {'lon': 'lon', 'lat': 'lat',  'time': 'time'}}

fieldset = FieldSet.from_c_grid_dataset(filenames, variables, dimensions)

pset = ParticleSet.from_line(fieldset, pclass=JITParticle, #ScipyParticle
                             size=10,           # releasing 10 particles
                             start=(13.5183, 45.6685),
                             finish=(13.5495, 44.6486))
print(pset)

pset.execute( AdvectionRK4 , runtime=timedelta(days=3), dt=timedelta(minutes=15),
              output_file=pset.ParticleFile(name='atutorial1.nc', outputdt=timedelta(minutes=15)),
              recovery={ErrorCode.ErrorOutOfBounds: DeleteParticle})
print(pset)
a = plotTrajectoriesFile('atutorial1.nc', mode='movie2d') #show the particles' trajectories in a movie
```

For curvilinear grids, the parcels's tutorial uses a mesh mask but for our dataset is not needed.

Test number 1

After having read correctly the netcdf file with the sea surface currents, a test with the release of 10 particles for 3 days has been performed. The 10 particles have the start points fixed. Some different simulations have been done in order to test if with the AdvectionRK4 kernel, a kernel with

no diffusion and any other stochastic components, all the particles' end points were the same. The test has been passed.

Test number 2

A second test is performed for the AdvectionRK4 kernel: 2 simulations with the release of 10 particles for 3 days are done, one for the forward mode and one for the backward mode. The aim of the test is to verify if, taking the final positions of the forward simulation and putting those position as initial position of the backward simulation, the final positions of the backward simulation and the initial positions of the forward one coincide. Looking to the results and making the difference between these points, the error is of about 0.00001° (i.e. of about 1 m).

Problem with particles off domain

Particles that go outside the domain lose their id number and the latitude and longitude are defined as Nan values. Moreover, the id becomes 0.

```
9       54000  18.830      40.592
9       54900  18.831      40.590
9       55800  18.832      40.589
9       56700  18.833      40.587
9       56700  18.833      40.587
0 -2147483648         NaN         NaN
0 -2147483648         NaN         NaN
0 -2147483648         NaN         NaN
0 -2147483648         NaN         NaN
0 -2147483648         NaN         NaN
```

Time test

1000 particles from Isonzo river

```
#for diffusion
kh_zonal = 1  #in m^2/s
```

```
        kh_meridional = 1 #in m^2/s both values are converted to degrees/s under the hood since the mesh is spherical and not flat
        fieldset.add_constant_field("Kh_zonal", kh_zonal, mesh=mesh)
        fieldset.add_constant_field("Kh_meridional", kh_meridional, mesh=mesh)
        pset = ParticleSet.from_line( fieldset, pclass=JITParticle, size=1000, start=(13.5597,45.7246),
                                      finish=(13.5597,45.7246))
        pset.execute( pset.Kernel(AdvectionRK4) + DiffusionUniformKh , runtime=timedelta(days=3),
                      dt=timedelta(minutes=15),
                      output_file=pset.ParticleFile(name='tutorial1_Isonzo.nc',
                      outputdt=timedelta(minutes=15))
```

Results:
real 1m20.150s
user 0m35.221s
sys 0m6.173s

10000 particles from Isonzo river

Same code as above but with size= 10000.
Results:
real 1m13.021s
user 0m29.346s
sys 0m6.597s
with these information:
INFO: Compiled ArrayJITParticleAdvectionRK4DiffusionUniformKh ==> /tmp/parcels-19004/libce7308433d4103aa799d818350013660_0.so
INFO: Temporary output files are stored in out-DTFHMSVT.
INFO: You can use "parcels_convert_npydir_to_netcdf out-DTFHMSVT" to convert these to a NetCDF file during the run.

100% (259200.0 of 259200.0) |##| Elapsed Time: 0:00:02 Time: 0:00:02

Test of constant releases from point-like and segment-like sources

Other tests have been performed considering particles released at a constant rate from the same location. For these tests, the COPERNICUS currents data is used. One test is performed considering a point source near the Isonzo's mouth releasing every 1 minute for 1 hour 10 particles and considering a total amount of 48 hours for the simulation or every 1 minute for 48 hours. A similar test is performed considering a release from a segment, always near the Isonzo's mouth.
Moreover, it is possible to release particle on polygonal chains creating specific arrays for latitude and longitude that reproduce the geometry required and go on with the release from_list.

Trial of the implementation of the beaching-resuspension algorithm

Beaching

A simple beaching model is implemented as a stochastic process in the coastal zone in order to take into account for the uncertainty/unreliability of the ocean current data in land adjacent ocean cells. For every time step the beaching probability p_B is calculated as:

$$p_B = \begin{cases} if\ d \leq D \quad p_B = 1 - \exp\left(-\frac{dt}{\lambda_B}\right) \\ if\ d > D \quad p_B = 0 \end{cases}$$

1

Where d is the distance of a particle to the nearest coastline, D is a predefined distance to the shore within we decide beaching to occur, dt is the integration timestep and λ_B is the characteristic timescale of plastic beaching (the number of days that a particle must spend within beaching zone such that there is a 63.2% chance that the particle has beached [9]).
In the Mediterranean, analysis of GPS trajectories of drifter buoys suggests λ_B =76 days, and an inverse modeling study suggests λ_B =26 days for plastic debris [9].

Resuspension

Also the resuspension is implemented as a stochastic process, where the resuspension probability p_R of a beached particle is define as:

$$p_R = 1 - \exp\left(-\frac{dt}{\lambda_R}\right)$$

2

Where dt is the integration timestep and λ_R is the characteristic timescale of plastic resuspension. A research has studied the resuspension timescales of plastic object with different sizes and found a range of λ_R= 69-273 days for this parameter [9]. When a particle beaches, its last position is saved and if the particle resuspends it continues its trajectory from this last position.

To implement the beaching-resuspension algorithm is necessary to construct a field with all the distances from the nearest shore dependent from latitude and longitude. This is done with the software QGIS. Firstly, we take a .shp of the coastal map then this file is converted into a raster file. After that, with the Proximity option from raster analysis the distance from the coast is calculated for all the points enclosed between the maximum and the minimum latitude and longitude of the land map. This distance is calculated in degrees and it is saved in a netCDF file.
Briefly, the algorithm reads the distance of every particle in each timestep, if this distance is less than the D chosen in the equation 1 the beaching probability is calculated and random number is generated, if this number is higher than the p_B so the particle is flagged as beached and all the possible displacements are no more calculated for this particle. As far as the resuspension algorithm is concerned, for all the particles flagged as beached is calculated the probability in equation 2, again a random number is generated and if this number is higher than the p_R the particle can be advected again and the flag beached is removed.
Since the particle class JIT has some limitations for the implementation of new kernels, we have switched to the Scipy class.

Problem on the diffusion kernel

We noticed that, after running the simulation with the same parameters for more than one time, the final position are always the same. So we decided to run some test on the tutorial parcels_tutorial.ipynb on the binder of the site http://www.oceanparcels.org.

We ran the same simulation multiple times and every 5 times we restarted the Jupyter kernel. In appedix the code is presented in Jupyther code for diffusion test.

First notebook kernel restart

First simulation:
P[0](lon=272499.468750, lat=100960.531250, depth=0.000000, time=518400.000000)
P[1](lon=261695.750000, lat=317240.531250, depth=0.000000, time=518400.000000)
Second simulation:
P[2](lon=221895.500000, lat=113273.695312, depth=0.000000, time=518400.000000)
P[3](lon=280482.750000, lat=359012.812500, depth=0.000000, time=518400.000000)
Third simulation:
P[4](lon=249686.281250, lat=73855.156250, depth=0.000000, time=518400.000000)
P[5](lon=212862.156250, lat=356235.312500, depth=0.000000, time=518400.000000)
Fourth simulation:
P[6](lon=231850.765625, lat=82502.718750, depth=0.000000, time=518400.000000)
P[7](lon=244108.359375, lat=323376.062500, depth=0.000000, time=518400.000000)
Fifth simulation:
P[8](lon=261284.984375, lat=76569.210938, depth=0.000000, time=518400.000000)
P[9](lon=263710.937500, lat=378645.875000, depth=0.000000, time=518400.000000)

Second notebook kernel restart

First simulation:
P[0](lon=272499.468750, lat=100960.531250, depth=0.000000, time=518400.000000)
P[1](lon=261695.750000, lat=317240.531250, depth=0.000000, time=518400.000000)

Second simulation:
P[2](lon=221895.500000, lat=113273.695312, depth=0.000000, time=518400.000000)
P[3](lon=280482.750000, lat=359012.812500, depth=0.000000, time=518400.000000)
Third simulation:
P[4](lon=249686.281250, lat=73855.156250, depth=0.000000, time=518400.000000)
P[5](lon=212862.156250, lat=356235.312500, depth=0.000000, time=518400.000000)
Fourth simulation (here I've re-run also the fieldset):
P[6](lon=229463.656250, lat=82117.406250, depth=0.000000, time=518400.000000)
P[7](lon=216254.078125, lat=336935.812500, depth=0.000000, time=518400.000000)
Fifth simulation:
P[8](lon=272024.125000, lat=91632.546875, depth=0.000000, time=518400.000000)
P[9](lon=269014.906250, lat=329866.125000, depth=0.000000, time=518400.00000)

Third notebook kernel restart

First simulation:
P[0](lon=272499.468750, lat=100960.531250, depth=0.000000, time=518400.000000)
P[1](lon=261695.750000, lat=317240.531250, depth=0.000000, time=518400.000000)
Second simulation:
P[2](lon=221895.500000, lat=113273.695312, depth=0.000000, time=518400.000000)
P[3](lon=280482.750000, lat=359012.812500, depth=0.000000, time=518400.000000)
Third simulation:
P[4](lon=249686.281250, lat=73855.156250, depth=0.000000, time=518400.000000)
P[5](lon=212862.156250, lat=356235.312500, depth=0.000000, time=518400.000000)
Fourth simulation:
P[6](lon=231850.765625, lat=82502.718750, depth=0.000000, time=518400.000000)
P[7](lon=244108.359375, lat=323376.062500, depth=0.000000, time=518400.000000)
Fifth simulation:
P[8](lon=261284.984375, lat=76569.210938, depth=0.000000, time=518400.000000)
P[9](lon=263710.937500, lat=378645.875000, depth=0.000000, time=518400.000000)

From these tests it can be noticed that with the first and third notebook kernel restart the final positions are exactly the same for every simulation. With the second restart the final positions are different after the simulation in which we've re-run also the fieldset.
The problem is solved using a different algorithm for the diffusion (presented below); probably the problem was the function **ParcelsRandom.normalvariate(0, math.sqrt(math.fabs(particle.dt)).**

```
def BrownianMotion2D(particle, fieldset, time):
    """Kernel for simple Brownian particle diffusion in zonal and meridional direction.
    Assumes that fieldset has fields Kh_zonal and Kh_meridional
    """
    r = 1/3.
    kh_meridional = fieldset.Kh_meridional[time, particle.depth, particle.lat, particle.lon]
    lat_p = particle.lat + random.uniform(-1., 1.) * math.sqrt(2*math.fabs(particle.dt)*kh_meridional/r)
    kh_zonal = fieldset.Kh_zonal[time, particle.depth, particle.lat, particle.lon]
    lon_p = particle.lon + random.uniform(-1., 1.) * math.sqrt(2*math.fabs(particle.dt)*kh_zonal/r)
    particle.lon=lon_p
    particle.lat=lat_p
```

Windage implementation

The windage implementation is performed adding the wind fields from the WRF netCDF files with the function add_field that adds the new fields to the current fieldset. It is necessary to perform a unit conversion is such a way to have the m/s in degrees/s, conversion that happens under the hood for the current fields. The functions used are: GeographicPolar for the U field and Geographic for the V one. The wind fields are scaled with a random factor of 1%-4% every dt of integration for each particle.

Test marless_parcels version 0.1

The 0.1 version of marless_parcels has different implemented kernels: the advection one (with first or fourth Runge-Kutta method's order), the windage one (with with first or fourth Runge-Kutta method's order), the diffusion one, and the beaching one. In this version the beaching kernel is kept

simple (or basic): every particle that, after the execution of all the kernels, has a distance from the coast less than or equal to zero is flagged as beached. There is no refloating kernel in this version, so every beached particle stays in its last position without be advected no more.
In this test we have studied the execution time of a simulation of 48 hours with a release firstly in open sea. We have examined the first and the fourth Runge-Kutta method's order for the advection and the windage kernels with different time interval of integration (1, 5, 10 and 15 minutes) and different number of particles to be released instantaneously (10, 100 and 1000).
In the following the schematic summary of the test and the tables of the results are shown.

TEST VERSION 0.1 PARCELS
Test: smulation of 48 hours,
instatnaneus release in open sea,
different time intervals of integration (1, 5, 10, 15 minutes)
different number of released particles (10, 100, 1000)
different order of Runge-Kutta method (first and fourth)
dt -> time interval of integration in minutes
n -> number of released particles
rk -> order of Runge-Kutta method for both advection and windage
tot beached -> total number of particles beached during the simulation;
the particle is considered beached if its distance from the coast is ≤ 0
time -> time duration of the simulation

dt (min)	n	rk	tot beached	time	time in seconds
1	10	1	3	00:06:56	416
1	100	1	20	01:09:44	4184
1	1000	1	187	11:22:22	40942
5	10	1	1	00:01:28	88
5	100	1	16	00:15:14	914
5	1000	1	187	02:11:32	7892
10	10	1	3	00:00:45	45
10	100	1	23	00:07:27	447
10	1000	1	176	01:10:05	4205
15	10	1	2	00:00:33	33
15	100	1	17	00:04:35	275
15	1000	1	165	00:46:47	2807

dt (min)	n	rk	tot beached	time	time in seconds
1	10	4	nan	00:08:22	502
1	100	4	nan	01:21:52	4912
1	1000	4	nan	13:06:03	47163
5	10	4	nan	00:01:45	105
5	100	4	nan	00:17:14	1034
5	1000	4	nan	02:41:38	9698
10	10	4	nan	00:00:55	55
10	100	4	nan	00:08:35	515
10	1000	4	nan	01:23:03	4983
15	10	4	nan	00:00:38	38
15	100	4	nan	00:05:46	346
15	1000	4	nan	00:53:24	3204

The tables of the results show the time interval of integration in minutes, the number of particles, the Runge-Kutta method's order, the total number of beached particles, the duration of the simulation in the format hours:minutes:seconds and in seconds.
Considering the ratios of the time over the number of particles, shown in the following tables, fixing the dt this ratio is independent from the number of particles, instead fixing the n it is dependent from the time interval of integration.

rk1: time (sec) / n				
n \ dt (min)	1	5	10	15
10	41.60000	8.80000	4.50000	3.30000
100	41.84000	9.14000	4.47000	2.75000
1000	40.94200	7.89200	4.20500	2.80700

rk4: time (sec) / n				
n \ dt (min)	1	5	10	15
10	50.20000	10.50000	5.50000	3.80000
100	49.12000	10.34000	5.15000	3.46000
1000	47.16300	9.69800	4.98300	3.20400

rk1: beached particles / released particles				
n \ dt (min)	1	5	10	15
10	30.0%	10.0%	30.0%	20.0%
100	20.0%	16.0%	23.0%	17.0%
1000	18.7%	18.7%	17.6%	16.5%

Trajectory extraction from models output file

The netCDF file structure of the output of the simulation with the GNOME and Parcels models is shown in the following figures.

The output of the GNOME model has two dimensions: time and data. The latter represents the number of particle times the time steps. Since the variables are dependent only from one dimension the best way to extract the data is to use a FORTRAN90 program with the help of the netcdf libraries. On the other hands, the output of the Parcels model has two dimensions (obs and traj) and each variable is dependent from both the dimensions. Nevertheless the more efficient way to extract the data is using a FORTRAN90 program with the netcdf libraries.

```
dimensions:
        time = 289 ;
        data = UNLIMITED ; // (289000 currently)
variables:
        double time(time) ;
                time:units = "seconds since 2021-09-09 00:00:00" ;
                time:long_name = "time" ;
                time:standard_name = "time" ;
                time:calendar = "gregorian" ;
                time:comment = "unspecified time zone" ;
        int particle_count(time) ;
                particle_count:units = "1" ;
                particle_count:long_name = "number of particles in a given timestep" ;
                particle_count:ragged_row_count = "particle count at nth timestep" ;
        float longitude(data) ;
                longitude:long_name = "longitude of the particle" ;
                longitude:units = "degrees_east" ;
        float latitude(data) ;
                latitude:long_name = "latitude of the particle" ;
                latitude:units = "degrees_north" ;
        float depth(data) ;
                depth:long_name = "particle depth below sea surface" ;
                depth:units = "meters" ;
                depth:axis = "z positive down" ;
        float mass(data) ;
                mass:units = "grams" ;
        int age(data) ;
                age:description = "from age at time of release" ;
                age:units = "seconds" ;
        short status_codes(data) ;
                status_codes:long_name = "particle status flag" ;
                status_codes:valid_range = 0, 10 ;
                status_codes:flag_values = 2, 3, 7, 10 ;
                status_codes:flag_meanings = "2:in_water 3:on_land 7:off_maps 10:evaporated" ;
        int id(data) ;
                id:description = "particle ID" ;
                id:units = "1" ;
```

Figure 3: netCDF file structure of the output of the GNOME model simulation.

```
dimensions:
        obs = 25 ;
        traj = 20 ;
variables:
        int64 trajectory(traj, obs) ;
                trajectory:_FillValue = -9223372036854775808LL ;
                trajectory:long_name = "Unique identifier for each particle" ;
                trajectory:cf_role = "trajectory_id" ;
        double time(traj, obs) ;
                time:_FillValue = NaN ;
                time:long_name = "" ;
                time:standard_name = "time" ;
                time:units = "seconds" ;
                time:axis = "T" ;
        float lat(traj, obs) ;
                lat:_FillValue = NaNf ;
                lat:long_name = "" ;
                lat:standard_name = "latitude" ;
                lat:units = "degrees_north" ;
                lat:axis = "Y" ;
        float lon(traj, obs) ;
                lon:_FillValue = NaNf ;
                lon:long_name = "" ;
                lon:standard_name = "longitude" ;
                lon:units = "degrees_east" ;
                lon:axis = "X" ;
        float z(traj, obs) ;
                z:_FillValue = NaNf ;
                z:long_name = "" ;
                z:standard_name = "depth" ;
                z:units = "m" ;
                z:positive = "down" ;
        float p(traj, obs) ;
                p:_FillValue = NaNf ;
                p:long_name = "" ;
                p:standard_name = "p" ;
                p:units = "unknown" ;
```

Figure 4: netCDF file structure of the output of the Parcels model simulation.

Time test for the data extraction

As far as the GNOME simulation output is concerned, the FORTRAN90 program is implemented in two ways depending on the possibility of the particle counts to change during the simulation.
The first way works if the number of particles doesn't change, the second one if it changes; in the following table the results of a time test for the extraction with the two different ways are shown for 1000 and 10000 particles with 289 time-steps per particle.

Number of particles	***Time – first way***	***Time – second way***
1000	1 m 37.200 s	4 m 22.256 s
10000 (node = 1 ppn = 2)	Walltime: 22 m 4 s	Walltime: 5 h 1 m 26 s

Another time test is performed with a GMT and a NCL programs that plot all the trajectories together. In the following table the results of this time test are shown:

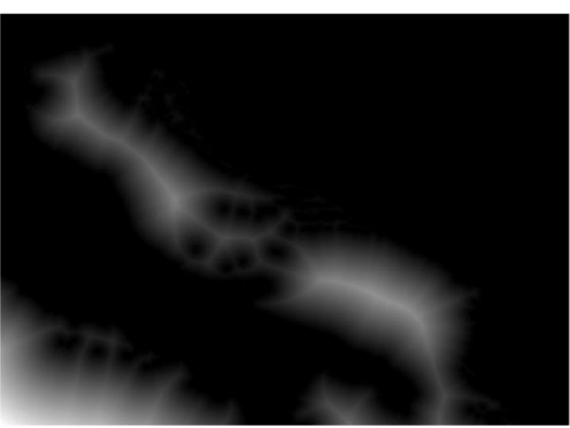
Number of particles	***GMT***	***NCL***
1000	1 m 16.742 s	Walltime = 26 m 27 s
10000	Walltime: 37 m 12 s	-

With pyGNOME with a continuous release of particles, it is necessary to implement a third way to extract the data because for every timestep there are a new amount of particle added in the particle count. The program is written in FORTRAN90 and use the netcdf libraries.

Backtrajectories

Since in the model there isn't the possibility to insert a map, it is necessary to find a way to insert the information of the coastlines.

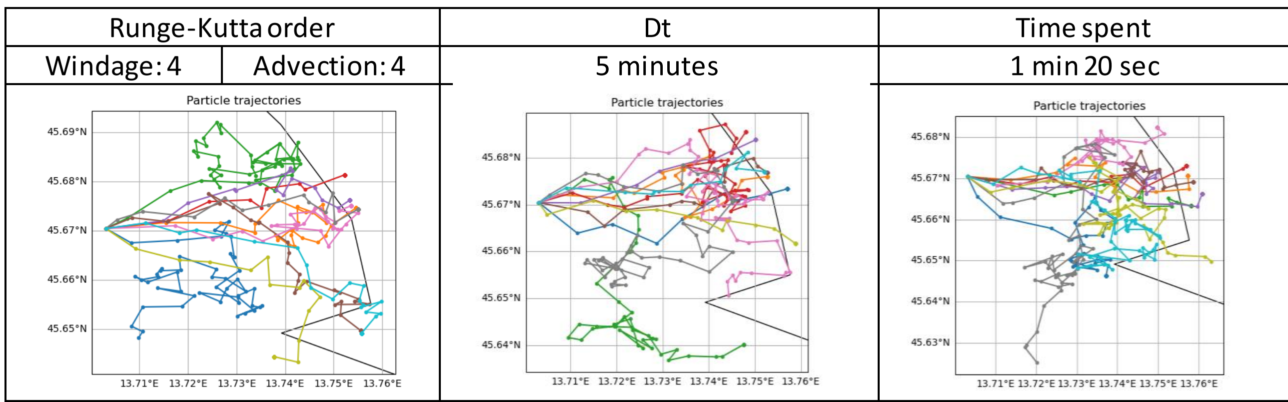
The best solution found is to construct a map of proximity, namely a map in which in every point the value of the shorter distance from the coast is stored.
This is performed with the software QGIS. The generic map has to be inserted as a shapefile, then it is necessary to merge all the elements (coastlines of the continental parts and the coastlines of the islands) in one single element. Afterwards, the shapefile has to be rasterize and then it is possible to implement the function *Proximity (raster distance).*
The result is shown in the following image on the right.



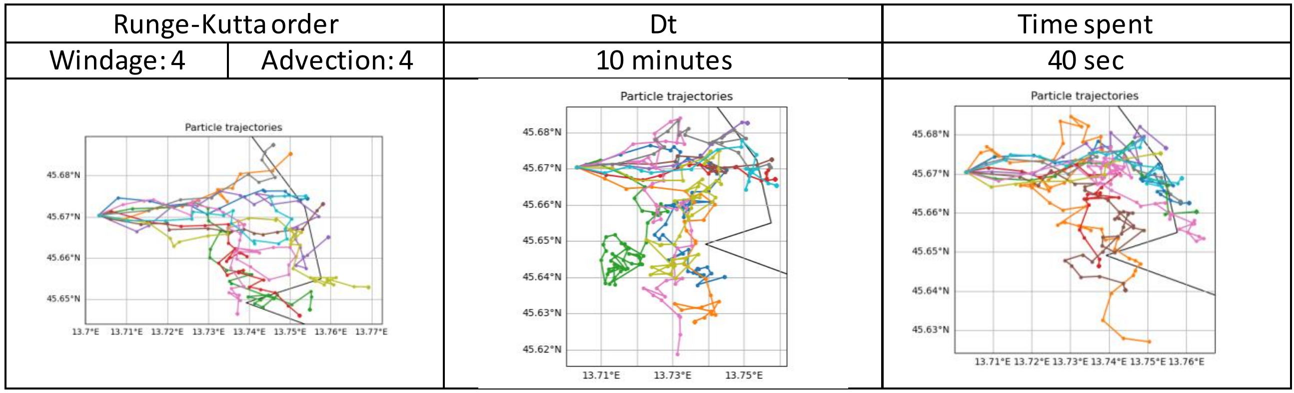
After that, the information is stored in a netcdf file that is going to be imported in the model.
To simulate the backtrajectories, the model moves the lagrangian elements with the marine currents, the wind field and, with a simple beaching algorithm, these particles are beached. This simple beaching algorithm works reading the information of the distance from the coast in every timestep, as soon as the particles reach a distance equal to zero they stop in that point until the end.
In the following section some tests are going to be presented.

Preliminary tests

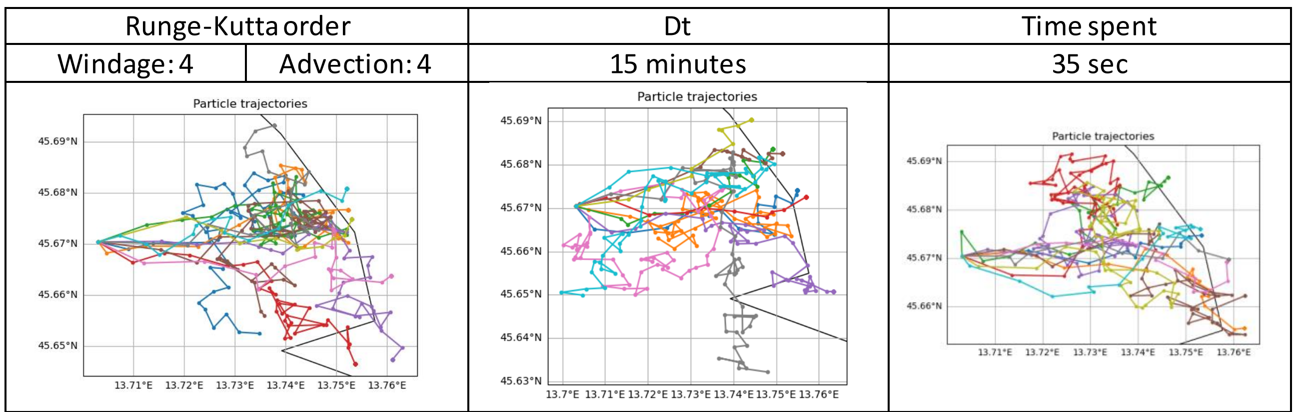
For all the results presented, the simulations have a duration of 48 hours, with an instantaneous release of 10 particles in the position lat= 45.670411, lon=13.703133 (in the front of the Trieste's harbour). The simulations start on the 30 of November 2017 and end on the 28 of November 2017. The Runge-Kutta order for the windage and the advection is changed together with the dt of integration. For every set of parameters 3 different simulations are performed and the time spent is stored.



Runge-Kutta order		Dt	Time spent
Windage: 4	Advection: 4	5 minutes	1 min 20 sec



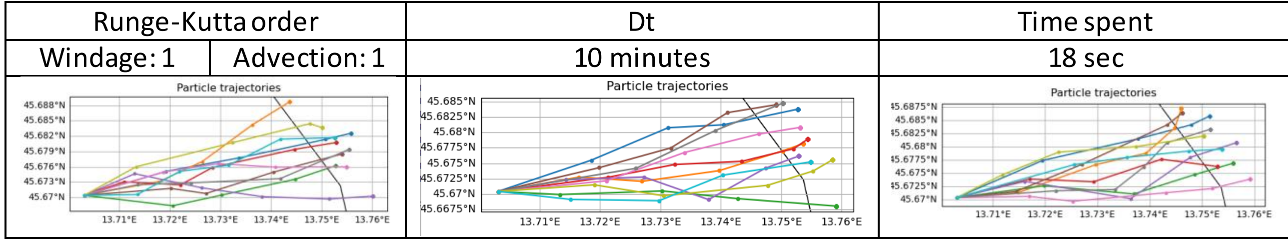
Runge-Kutta order		Dt	Time spent
Windage: 4	Advection: 4	10 minutes	40 sec



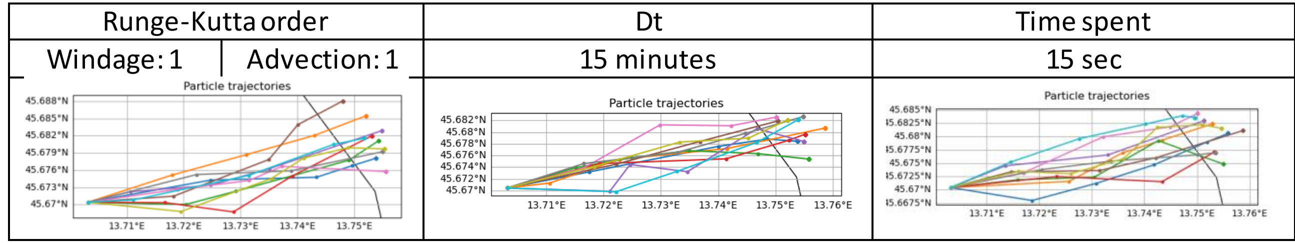
Runge-Kutta order		Dt	Time spent
Windage: 4	Advection: 4	15 minutes	35 sec

Runge-Kutta order		Dt	Time spent
Windage: 1	Advection: 1	5 minutes	22 sec

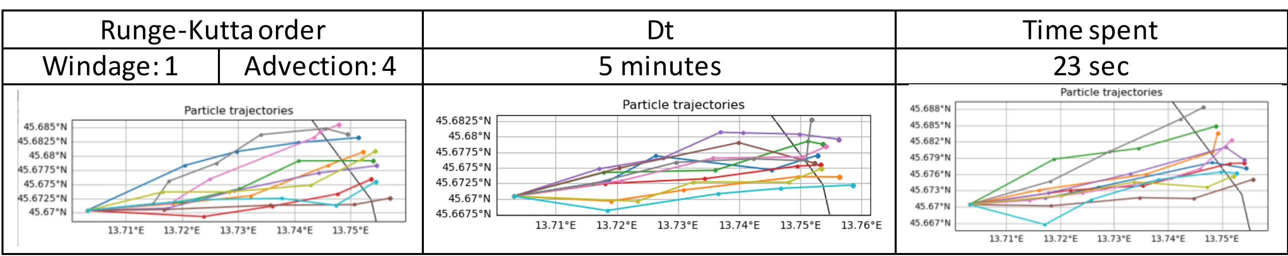
Runge-Kutta order		Dt	Time spent
Windage: 1	Advection: 1	10 minutes	18 sec



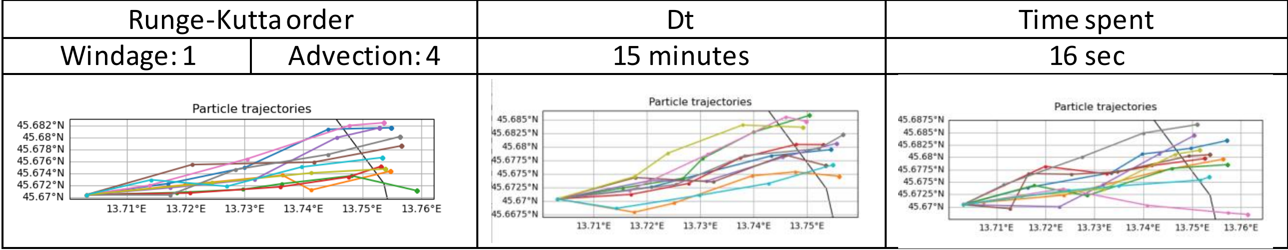
Runge-Kutta order		Dt	Time spent
Windage: 1	Advection: 1	15 minutes	15 sec



Runge-Kutta order		Dt	Time spent
Windage: 1	Advection: 4	5 minutes	23 sec

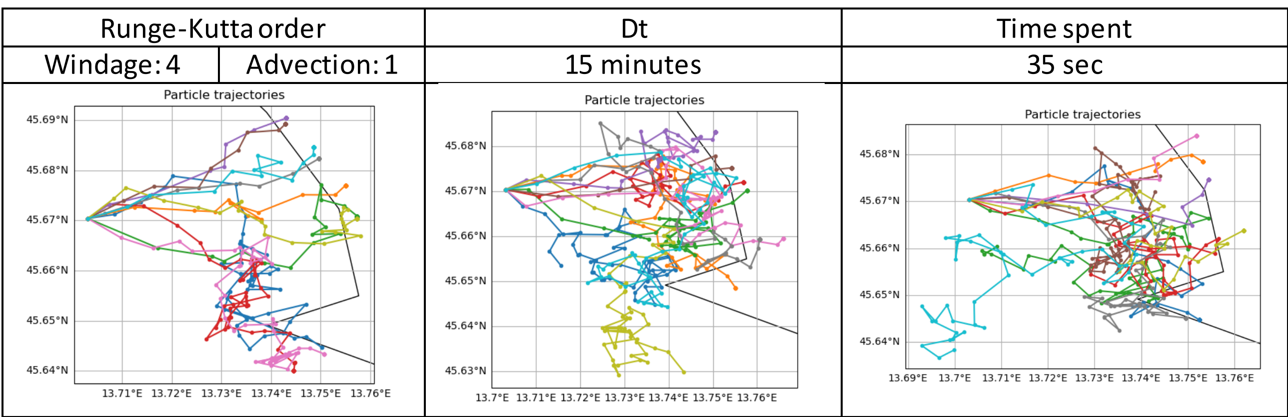


Runge-Kutta order		Dt	Time spent
Windage: 1	Advection: 4	10 minutes	19 sec



Runge-Kutta order		Dt	Time spent
Windage: 1	Advection: 4	15 minutes	16 sec

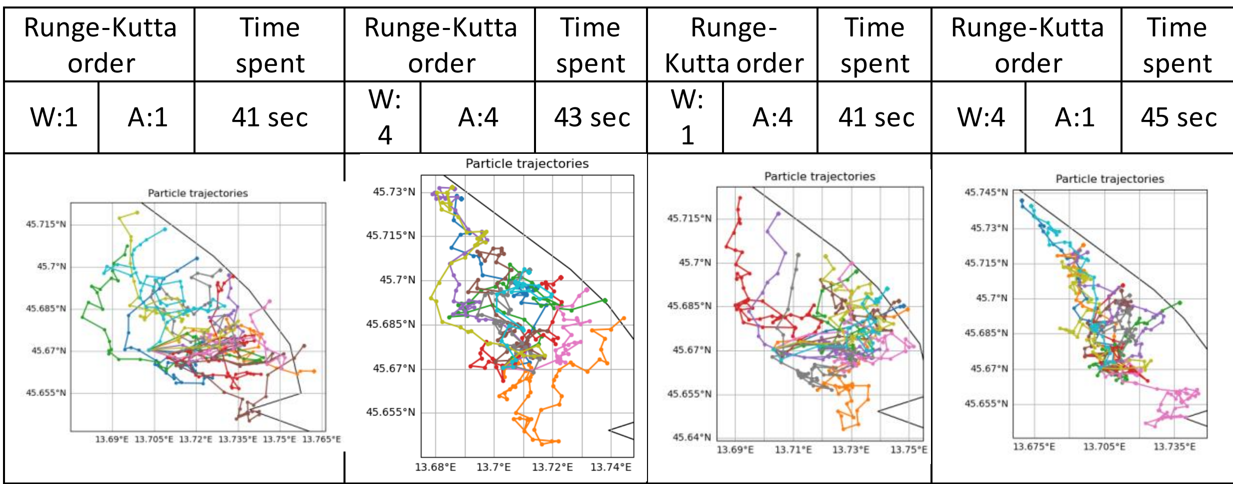
For the following tests the dt is fixed to 15 minutes.



Runge-Kutta order		Dt	Time spent
Windage: 4	Advection: 1	15 minutes	35 sec

From these tests, it can be noticed that with the Runge-Kutta order 4, the trajectories present more steps in their path. Moreover, it can be seen that the windage seems to have a major influence on the transport that can be related to a particular meteorological situation. For this reason these test are performed for another period of time, starting from the 30 of June 2018 back to the 28 of June

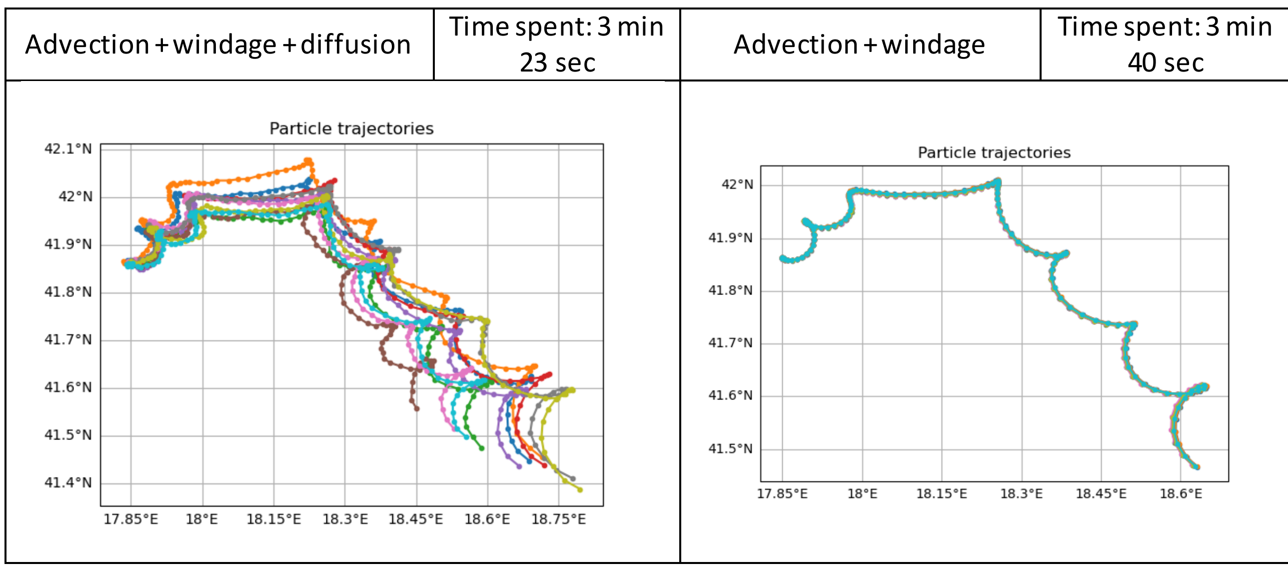
2018. For these test only one simulation is performed for each set of parameter and the dt is kept fixed to 15 minutes.



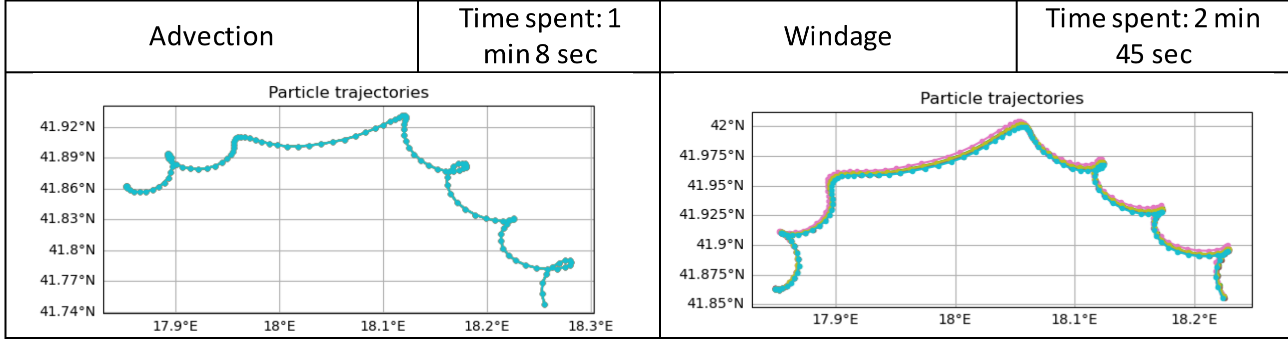
Runge-Kutta order		Time spent	Runge-Kutta order		Time spent	Runge-Kutta order		Time spent	Runge-Kutta order		Time spent
W:1	A:1	41 sec	W: 4	A:4	43 sec	W: 1	A:4	41 sec	W:4	A:1	45 sec

The following tests are performed with a dt of 5 minutes, with a Runge-Kutta order of 4 for windage and advection. The simulations start on the 30 of June 2018 and end on the 25 of June 2018. In these tests the windage, the advection and the diffusion are deactivated in turn in order to test the transport algorithms. The starting point of release is lat = 41.862331, lon = 17.852863, with an instantaneous release of 10 particles.

Advection + windage + diffusion	Time spent: 3 min 23 sec	Advection + windage	Time spent: 3 min 40 sec

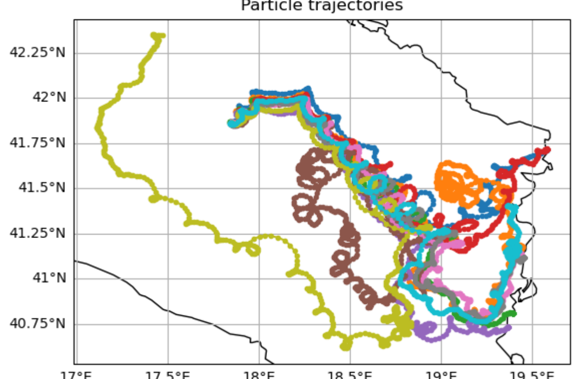


Advection	Time spent: 1 min 8 sec	Windage	Time spent: 2 min 45 sec



The windage is the algorithm that consumes more time because for every time step the model picks a random number between 1 and 4 in order to set a the 1%-4% of the wind speed to compute the windage. With the diffusion there is a spread of the particles at the end that is about 30 km.

The following simulation is performed with a dt of 5 minutes, with the Runge-Kutta at the fourth order and for a duration of 30 days starting from the 30 of June 2018 back in time. There is an instantaneous release of 10 particles from the point lat = 41.862331, lon = 17.852863.
The time spent for this simulation is of 16 minutes.



Tests: same parameters, same start time but different positions

Punctual release of 5 particle per point; start time: 24 October 2018 00:30UTC; dt 15 minutes; Runge-Kutta fourth order for windage and advection; minimum distance from the coast: 500 m.

lat_start = 43.6836007
lon_start = 13.3002594
all beached after 3 minutes

lat_start = 43.6864559
lon_start = 13.3057096
all beached after 9 minutes

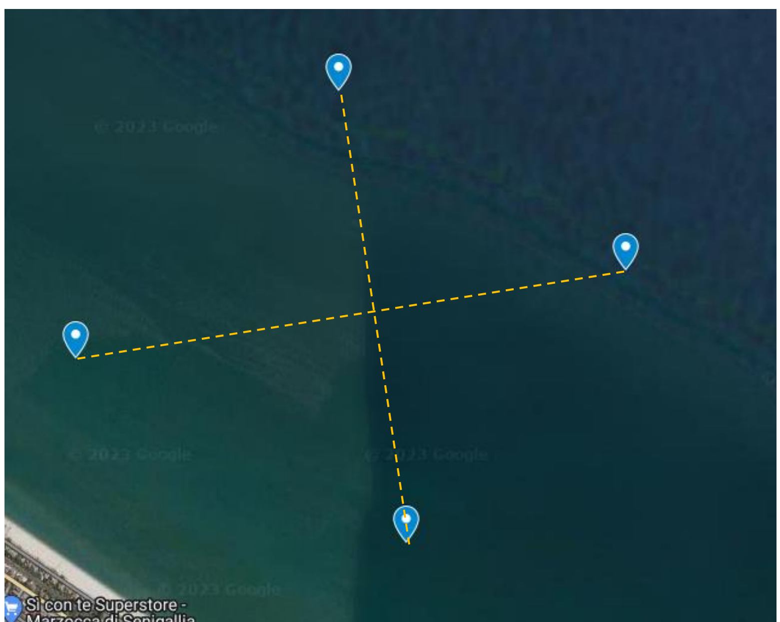
lat_start = 43.6863007
lon_start = 13.2950666
all beached after 2 minutes

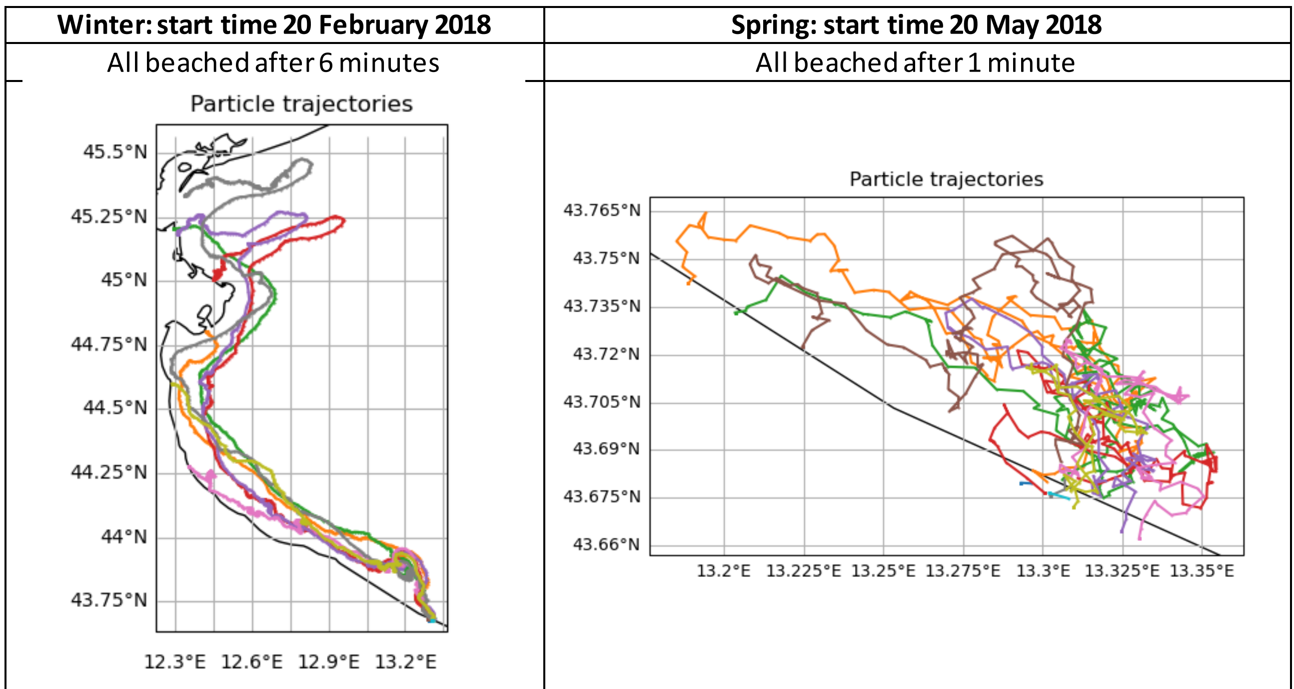
lat_start = 43.679565
lon_start = 13.2963755
all beached after 2 minutes

lat_start = 43.6809006
lon_start = 13.3053234
all beached after 2 minutes

Tests: same parameters, release from a polygon, different start times

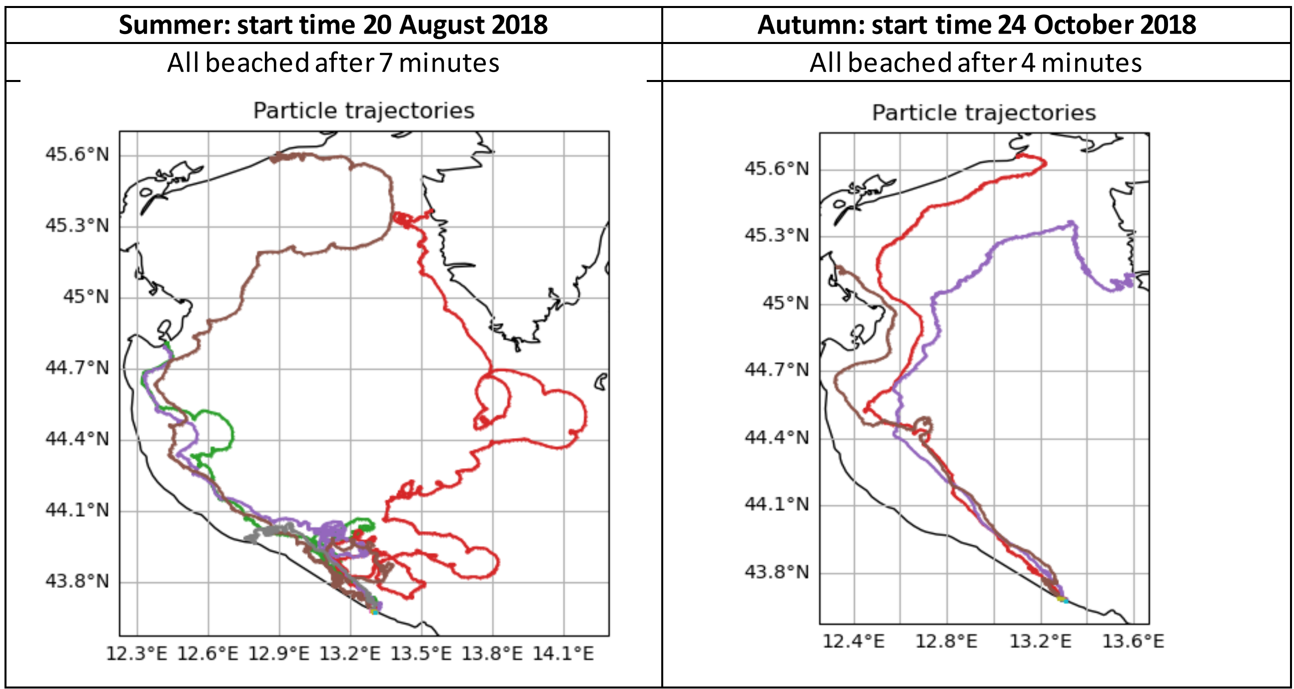
Release from a polygon 1km x 1 km, identified by 2 diagonal segments both with a line-release of 5 particle, so a total release of 10 particles per run. Four different start times are taken: one per each season. Runge-Kutta fourth order for windage and advection; dt of 15 minutes; minimum distance from the coast: 500 m.





Winter: start time 20 February 2018	Spring: start time 20 May 2018
All beached after 6 minutes	All beached after 1 minute

Summer: start time 20 August 2018	**Autumn: start time 24 October 2018**
All beached after 7 minutes	All beached after 4 minutes



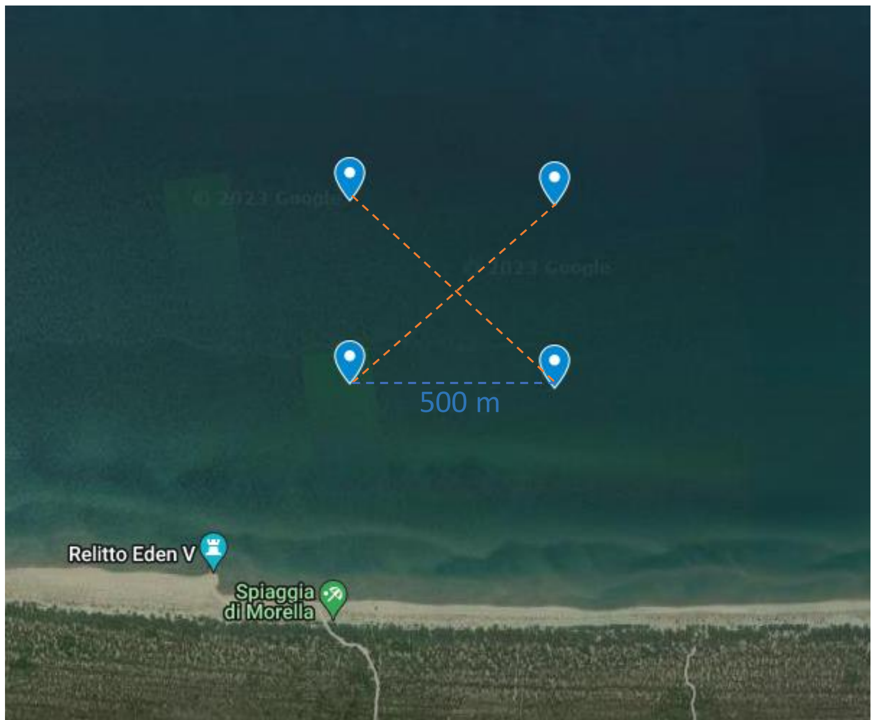
Tests: continuous release for 2 months

Release from a point line source: lat = 43.6864559, lon = 13.3057096; continuous release of 5 particles/day; Runge-Kutta fourth order for windage and advection; dt of 15 minutes:
Time spent for a simulation of 2 months: 88 minutes.

Tests: 334 days of simulation from the Bosco Isola Lesina beach in Puglia

Time start of the back trajectories: 2018-10-25 00:30:00 UTC. Fourth Runge Kutta order for advection and windage.
Dt of integration: 15 minutes; continuous release of 5 particles per day in each polygon diagonal (as you can see on the following image, the polygon is a square with dimension of 500x500m).

Time spent for the simulations: 7 hours; NetCDF output dimension: 513 MB. Time spent to extract data: 1 minute; final ASCII file dimension: 220 MB.



On the following picture, all the possible source points found on the last time step of the simulation are shown. There is no accumulation points, all the shown data represent a single point not a superposition of different ones. The spatial resolution of the sources points is of the order of 1 meter (precisely, the minimum distance between two coordinates is of 0.83 meters).

These results are going to be analysed with a Machine Learning clustering algorithm to find the higher probability areas in which the sources can be found. In the following section, the different clustering algorithm are going to be presented.

Valuation of the different Machine Learning Clustering algorithm

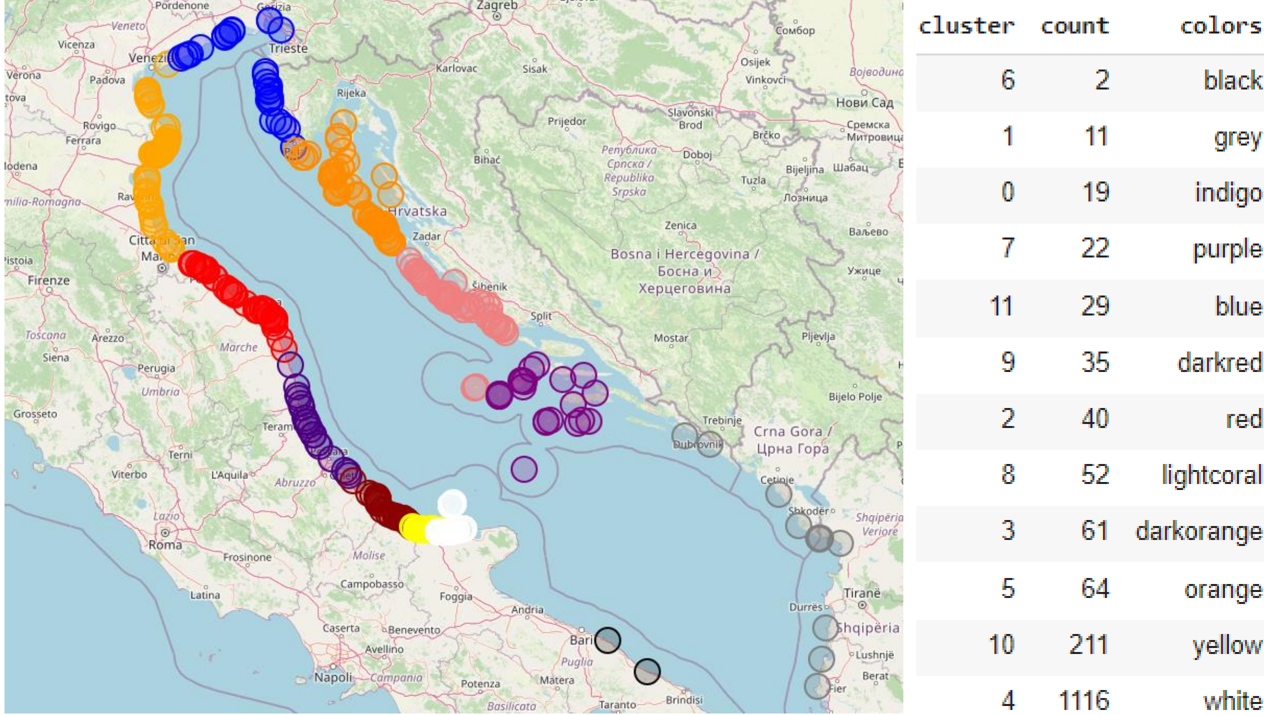
We need to use an unsupervised clustering algorithm as we don’t have the ground truth to compare the clustering algorithm to the true labels to estimate its performance. So we want to try to

investigate the structure of the data by grouping the coordinate data points into individual subgroups.

K-Means algorithm

In K-Means algorithm each data point belongs to only one group so there is non-overlapping clusters. It assigns data points to a cluster such that the sum of the squared distance between the data points and the cluster's centroid (that is defined as the arithmetic mean of all the data points that belong to that cluster) is at the minimum. The less variation we have within clusters, the more homogeneous the data points are within the same cluster.

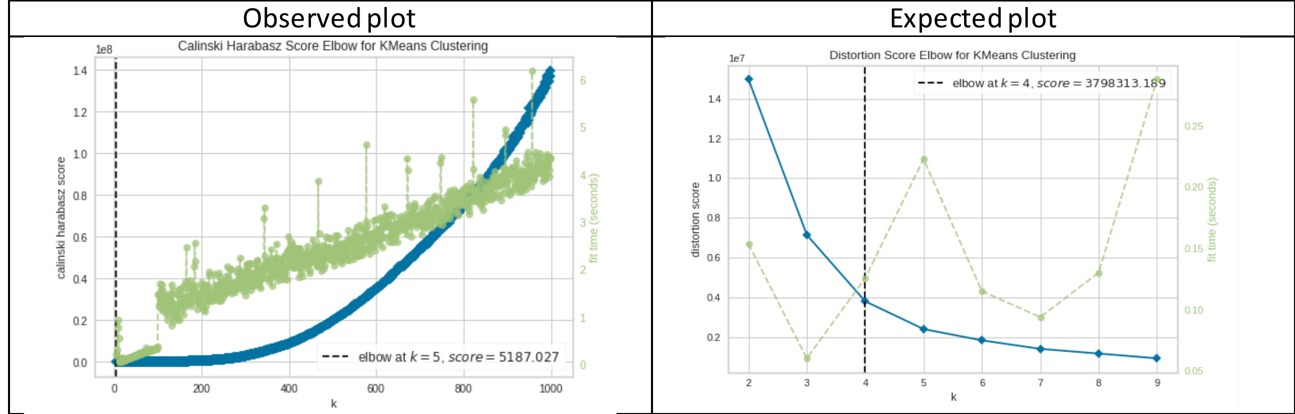
The algorithm needs to have the number of clusters specified. The first test is performed with the choice of 12 clusters; on the following image the result is shown together with the table that shows the sources counts for each colour.



cluster	count	colors
6	2	black
1	11	grey
0	19	indigo
7	22	purple
11	29	blue
9	35	darkred
2	40	red
8	52	lightcoral
3	61	darkorange
5	64	orange
10	211	yellow
4	1116	white

Since K-Means requires the number of clusters as an input and doesn't learn it from data, there is no right answer in terms of the number of clusters that we should have. To evaluate the algorithm performance we used the method called *Elbow method*.

This method gives us an idea on what a good number of clusters would be based on the sum of squared distance between data points and their assigned clusters' centroids. The following two images show the observed plot and the expected one. As it is can be noticed, the elbow method fail to find the number of possible clusters as it finds the elbow exactly at the beginning of the plot data, the curve is not monotonically decreasing and there is not any elbow or has an obvious point where the curve starts flattening out.



In conclusion, we can deduce that this algorithm is not the most suitable for our points; this can be inferred to the fact that it is not good in capturing structures that don't have a spherical-like shape. Moreover, K-Means is not an ideal algorithm for latitude-longitude data because it minimizes variance, not geodetic distance. There is substantial distortion at latitudes far from the equator.

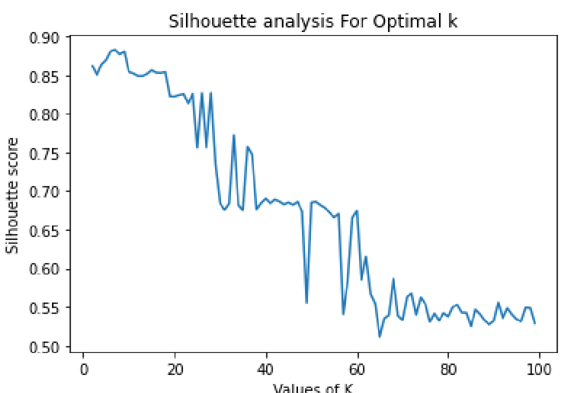
A more precise approach is to use the ***Silhouette Score***, which is the mean silhouette coefficient over all the points. The ***Silhouette Coefficient*** s for a single sample is given as:

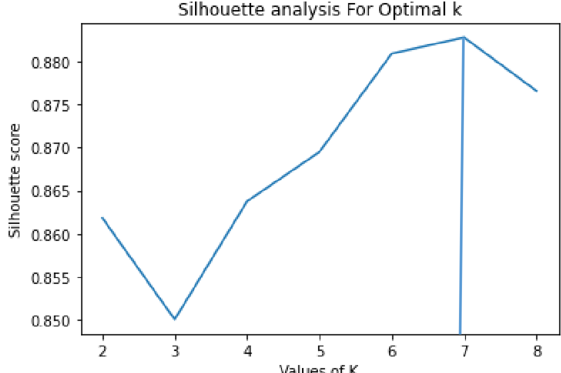
$$s = \frac{b - a}{\max(a, b)}$$

Where, **a** is the mean distance between a sample and all other points in the same class, and **b** is the mean distance between a sample and all other points in the next nearest cluster. Then, the *Silhouette Coefficient* for a set of samples is given as the mean of the *Silhouette Coefficient* for each sample.

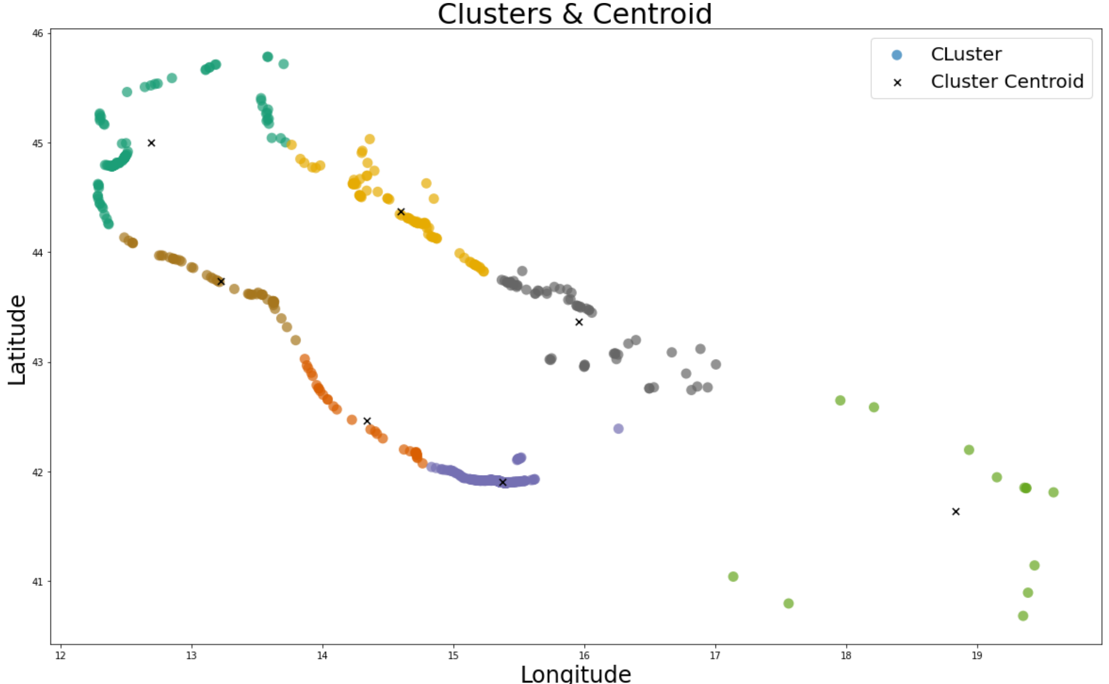
The silhouette coefficient can vary between -1 and +1: a coefficient close to +1 means that the sample is well inside its own cluster and far from other clusters, while a coefficient close to 0 means that it is close to a cluster boundary, and finally a coefficient close to -1 means that the sample may have been assigned to the wrong cluster.

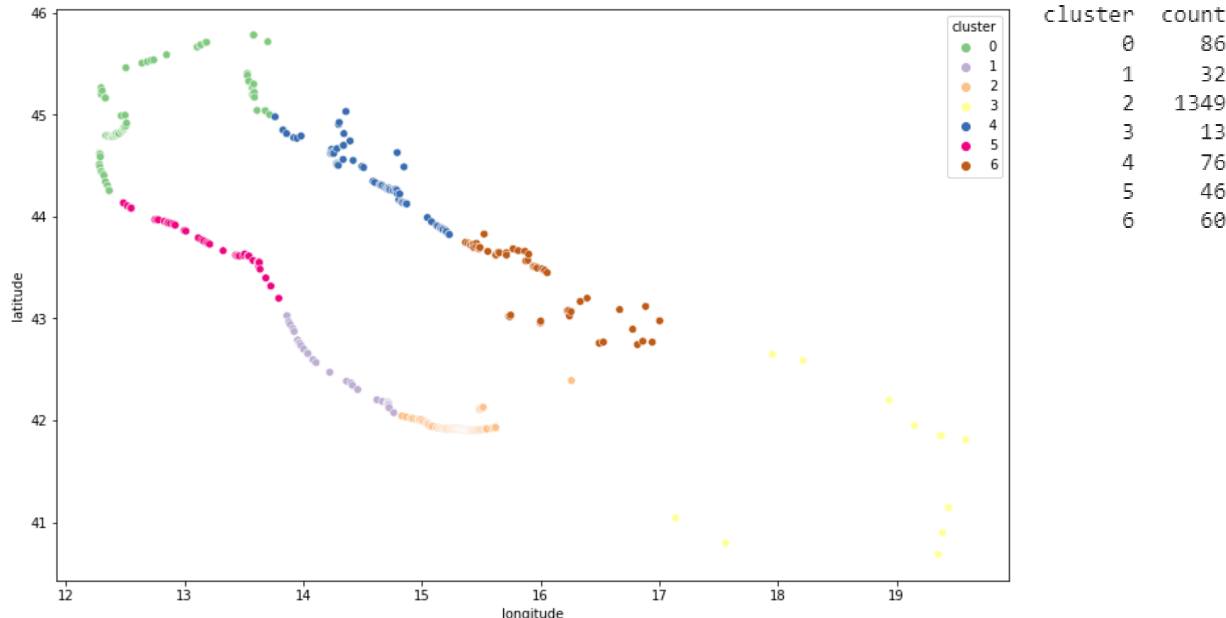
Selecting the number of clusters between 2 and 100 clusters (K), the following image shows the results of the silhouette score; the best value to put as number of clusters is the maximum value of this curve.
In this case the maximum silhouette score is obtained for 7 clusters.





With 7 clusters this is the clusterization obtained:


Clusters & Centroid
CLuster
Cluster Centroid
Latitude
Longitude
46
45
44
43
42
41
12
13
14
15
16
17
18
19



cluster	count
0	86
1	32
2	1349
3	13
4	76
5	46
6	60

DBSCAN (Density-based spatial clustering of applications with noise) algorithm

With this algorithm you don’t have to specify the number of clusters to use it and it is a density-based clustering algorithm. In density-based clustering, data is grouped by areas of high concentrations of data points surrounded by areas of low concentrations of data points. The algorithm finds the places that are dense with data points and calls those clusters.

The great thing about this is that the clusters can be of any shape. You aren't constrained to expected conditions. The clustering algorithms under this type don't try to assign outliers to clusters, so they get ignored.

The algorithm uses two parameters:

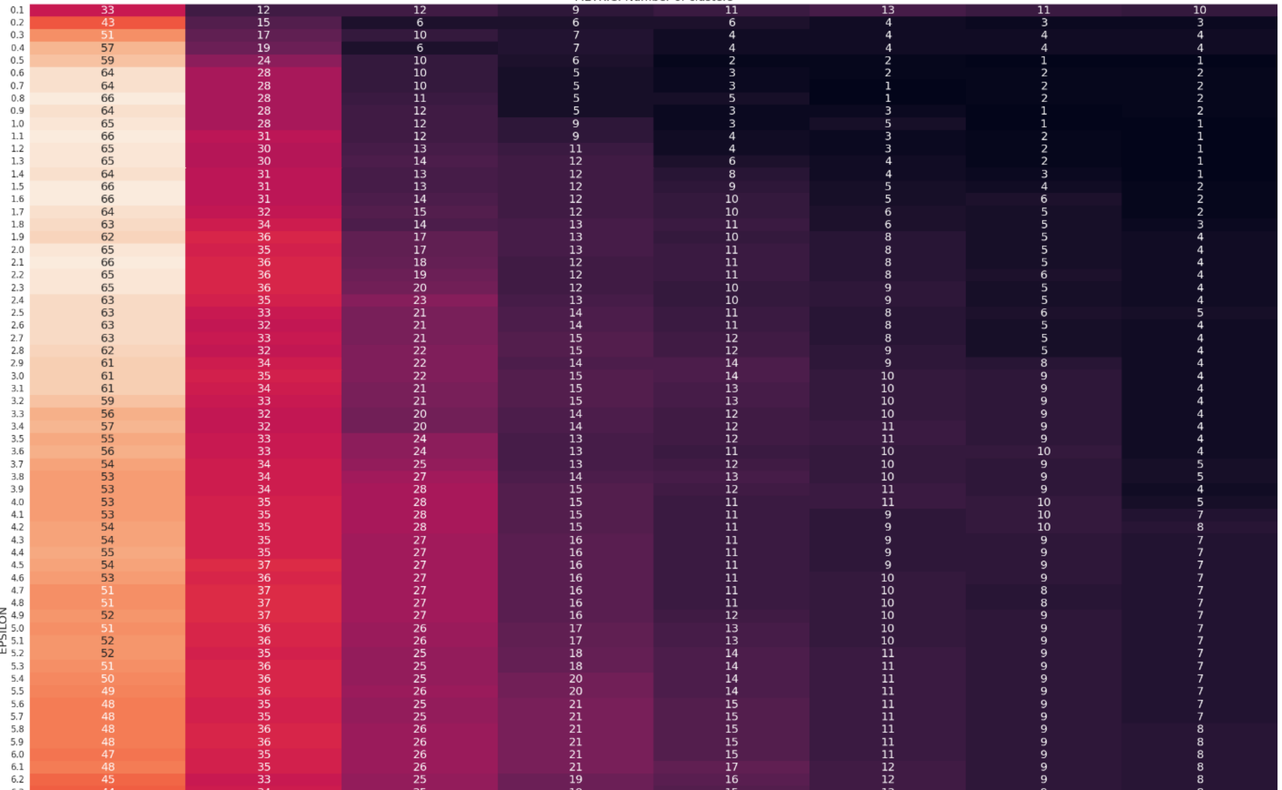
- **eps**: it defines the neighbourhood around a data point i.e. if the distance between two points is lower or equal to ‘eps’ then they are considered neighbours. If the eps value is chosen too

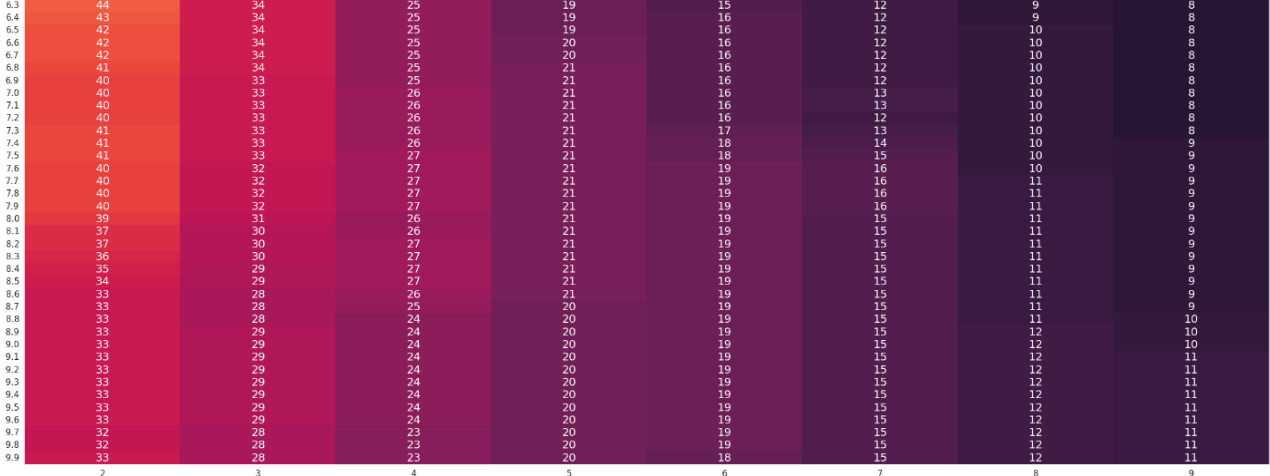
small then large part of the data will be considered as outliers. If it is chosen very large then the clusters will merge and the majority of the data points will be in the same clusters.

- **minPts**: Minimum number of neighbours (data points) within eps radius. Larger the dataset, the larger value of minPts must be chosen. Larger values are usually better for data sets with noise.

To evaluate the best value for these parameters, a heatmap that relates the eps parameter to the minPts (N) shows the number of clusters found for each pair of parameter.

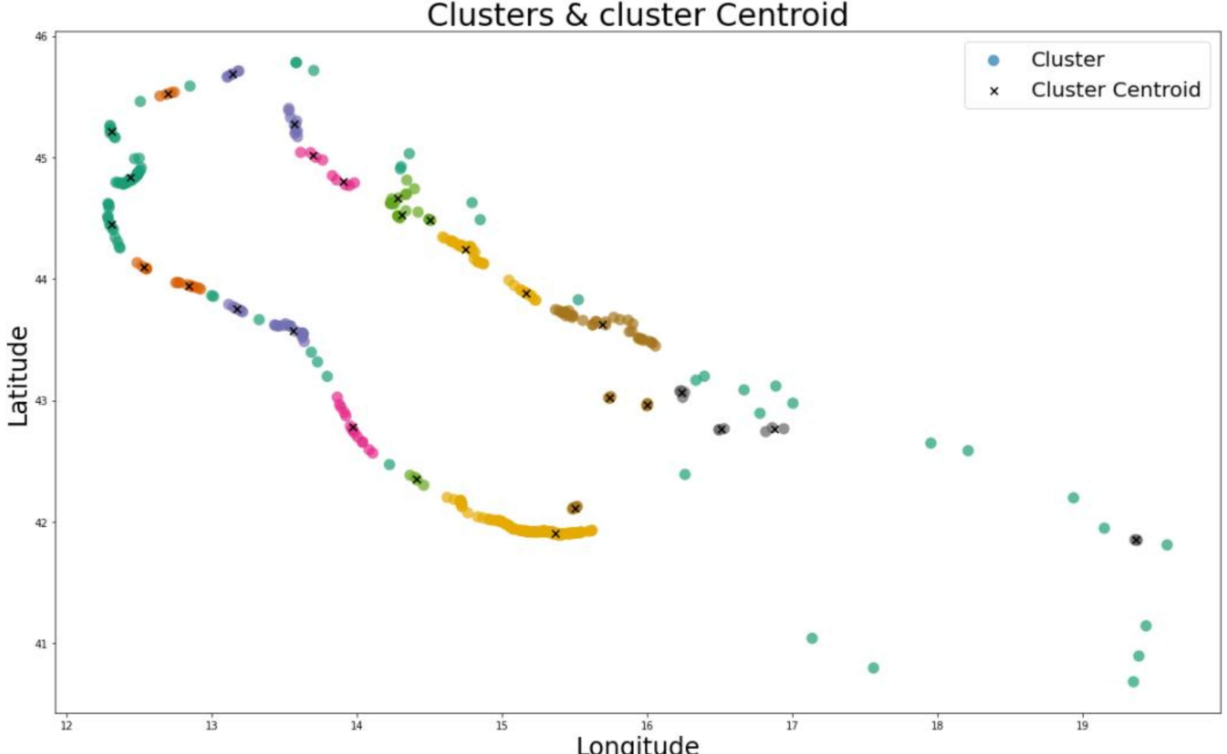
Also in this case we need to know approximately how many clusters there are in each dataset in order to choose the most suitable pair of parameters. So we chose to set eps=10 (km) and minPts =3. The algorithm found 21 clusters. The image shown the different clusters with their centroid and also the noise points found (label -1, colour white).

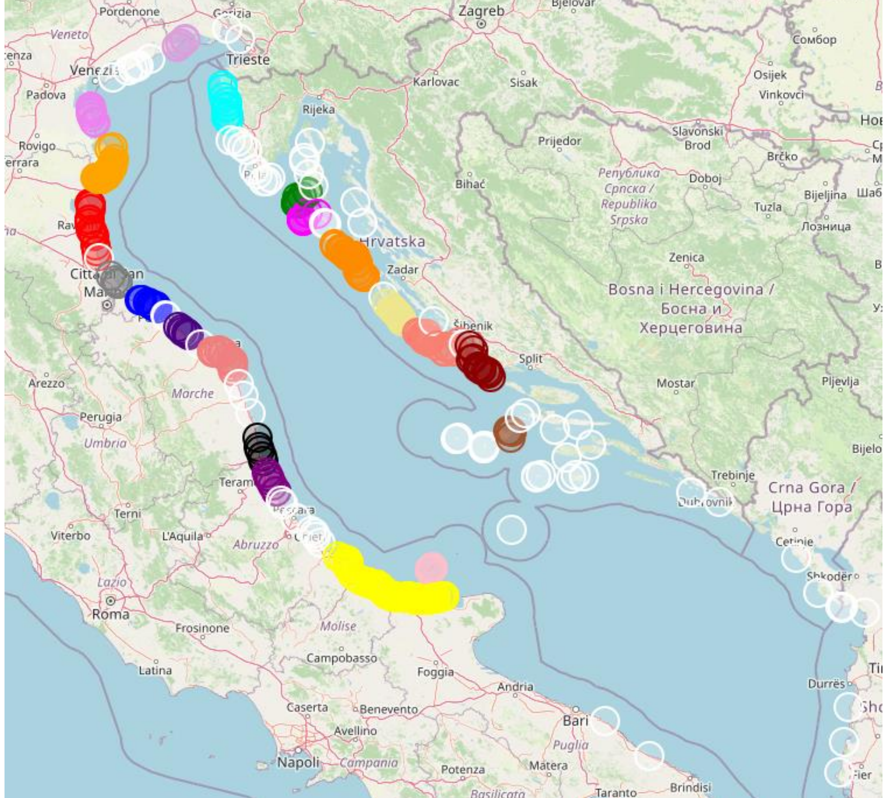

METRIC: Number of clusters
EPSILON
0.1 33 12 12 9 11 13 11 10
0.2 43 15 6 6 6 4 3 3
0.3 51 17 10 7 4 4 4 4
0.4 57 19 6 7 4 4 4 4
0.5 59 24 10 6 2 2 1 1
0.6 64 28 10 5 3 2 2 2
0.7 64 28 10 5 3 1 2 2
0.8 66 28 11 5 5 1 2 2
0.9 64 28 12 5 3 3 1 2
1.0 65 28 12 9 3 5 1 1
1.1 66 31 12 9 4 3 2 1
1.2 65 30 13 11 4 3 2 1
1.3 65 30 14 12 6 4 2 1
1.4 64 31 13 12 8 4 3 1
1.5 66 31 13 12 9 5 4 2
1.6 66 31 14 12 10 5 6 2
1.7 64 32 15 12 10 6 5 2
1.8 63 34 14 13 11 6 5 3
1.9 62 36 17 13 10 8 5 4
2.0 65 35 17 13 11 8 5 4
2.1 66 36 18 12 11 8 5 4
2.2 65 36 19 12 11 8 6 4
2.3 65 36 20 12 10 9 5 4
2.4 63 35 23 13 10 9 5 4
2.5 63 33 21 14 11 8 6 5
2.6 63 32 21 14 11 8 5 4
2.7 63 33 21 15 12 8 5 4
2.8 62 32 22 15 12 9 5 4
2.9 61 34 22 14 14 9 8 4
3.0 61 35 22 15 14 10 9 4
3.1 61 34 21 15 13 10 9 4
3.2 59 33 21 15 13 10 9 4
3.3 56 32 20 14 12 10 9 4
3.4 57 32 20 14 12 11 9 4
3.5 55 33 24 13 12 11 9 4
3.6 56 33 24 13 11 10 10 4
3.7 54 34 25 13 12 10 9 5
3.8 53 34 27 14 13 10 9 5
3.9 53 34 28 15 12 11 9 4
4.0 53 35 28 15 11 11 10 5
4.1 53 35 28 15 11 9 10 7
4.2 54 35 28 15 11 9 10 8
4.3 54 35 27 16 11 9 9 7
4.4 55 35 27 16 11 9 9 7
4.5 54 37 27 16 11 9 9 7
4.6 53 36 27 16 11 10 9 7
4.7 51 37 27 16 11 10 8 7
4.8 51 37 27 16 11 10 8 7
4.9 52 37 27 16 12 10 9 7
5.0 51 36 26 17 13 10 9 7
5.1 52 36 26 17 13 10 9 7
5.2 52 35 25 18 14 11 9 7
5.3 51 36 25 18 14 11 9 7
5.4 50 36 25 20 14 11 9 7
5.5 49 36 26 20 14 11 9 7
5.6 48 35 25 21 15 11 9 7
5.7 48 35 25 21 15 11 9 7
5.8 48 36 26 21 15 11 9 8
5.9 48 36 26 21 15 11 9 8
6.0 47 35 26 21 15 11 9 8
6.1 48 35 26 21 17 12 9 8
6.2 45 33 25 19 16 12 9 8
6.3 44 34 25 19 15 12 9 8



	2	3	4	5	6	7	8	9
6.3	44	34	25	19	15	12	9	8
6.4	43	34	25	19	16	12	9	8
6.5	42	34	25	19	16	12	10	8
6.6	42	34	25	20	16	12	10	8
6.7	42	34	25	20	16	12	10	8
6.8	41	34	25	21	16	12	10	8
6.9	40	33	25	21	16	12	10	8
7.0	40	33	26	21	16	13	10	8
7.1	40	33	26	21	16	13	10	8
7.2	40	33	26	21	16	12	10	8
7.3	41	33	26	21	17	13	10	8
7.4	41	33	26	21	18	14	10	9
7.5	41	33	27	21	18	15	10	9
7.6	40	32	27	21	19	16	10	9
7.7	40	32	27	21	19	16	11	9
7.8	40	32	27	21	19	16	11	9
7.9	40	32	27	21	19	16	11	9
8.0	39	31	26	21	19	15	11	9
8.1	37	30	26	21	19	15	11	9
8.2	37	30	27	21	19	15	11	9
8.3	36	30	27	21	19	15	11	9
8.4	35	29	27	21	19	15	11	9
8.5	34	29	27	21	19	15	11	9
8.6	33	28	26	21	19	15	11	9
8.7	33	28	25	20	19	15	11	9
8.8	33	28	24	20	19	15	11	10
8.9	33	29	24	20	19	15	12	10
9.0	33	29	24	20	19	15	12	10
9.1	33	29	24	20	19	15	12	11
9.2	33	29	24	20	19	15	12	11
9.3	33	29	24	20	19	15	12	11
9.4	33	29	24	20	19	15	12	11
9.5	33	29	24	20	19	15	12	11
9.6	33	29	24	20	19	15	12	11
9.7	32	28	23	20	19	15	12	11
9.8	32	28	23	20	19	15	12	11
9.9	33	28	23	20	18	15	12	11

N

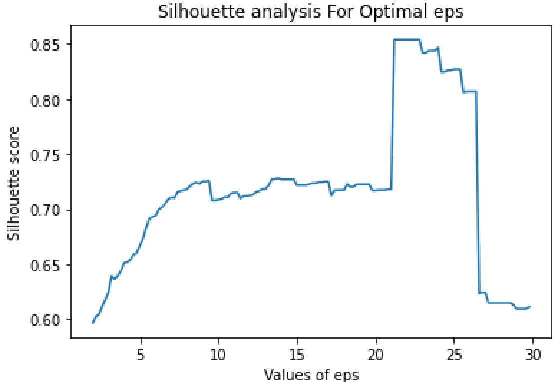

Clusters & cluster Centroid
Cluster
Cluster Centroid
Latitude
Longitude
46
45
44
43
42
41
12
13
14
15
16
17
18
19

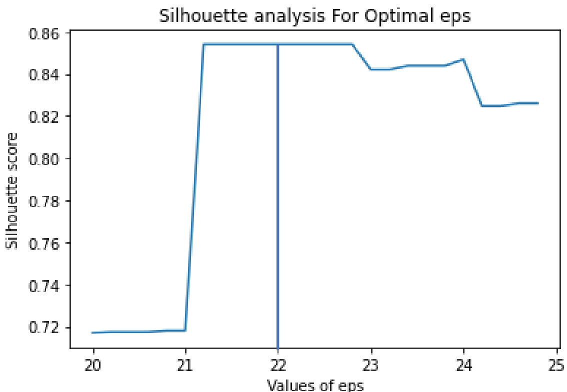


LABEL	count	colors
9	5	black
3	5	grey
19	5	sienna
5	6	plum
6	6	indigo
10	7	purple
1	7	violet
12	8	magenta
17	8	pink
4	10	blue
8	10	cyan
11	11	green
15	12	khaki
18	13	darkred
0	14	red
7	19	lightcoral
16	21	salmon
13	28	darkorange
2	36	orange
-1	78	white
14	1353	yellow

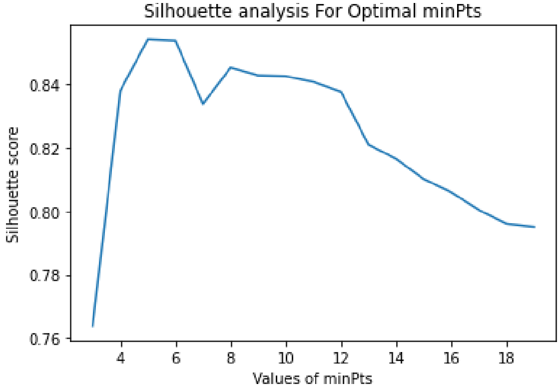
This algorithm is more arduous to tune; the parameters like the epsilon is not so intuitive to set, so it's more difficult to choose good initial parameter values for this algorithm.

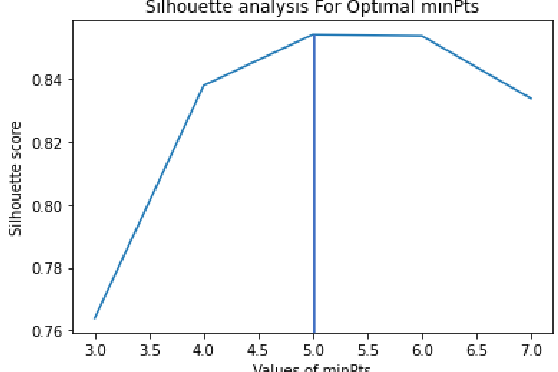
We tried to set the eps parameter computing the silhouette score as in the case of the algorithm K-means. The results are shown on the following plots:



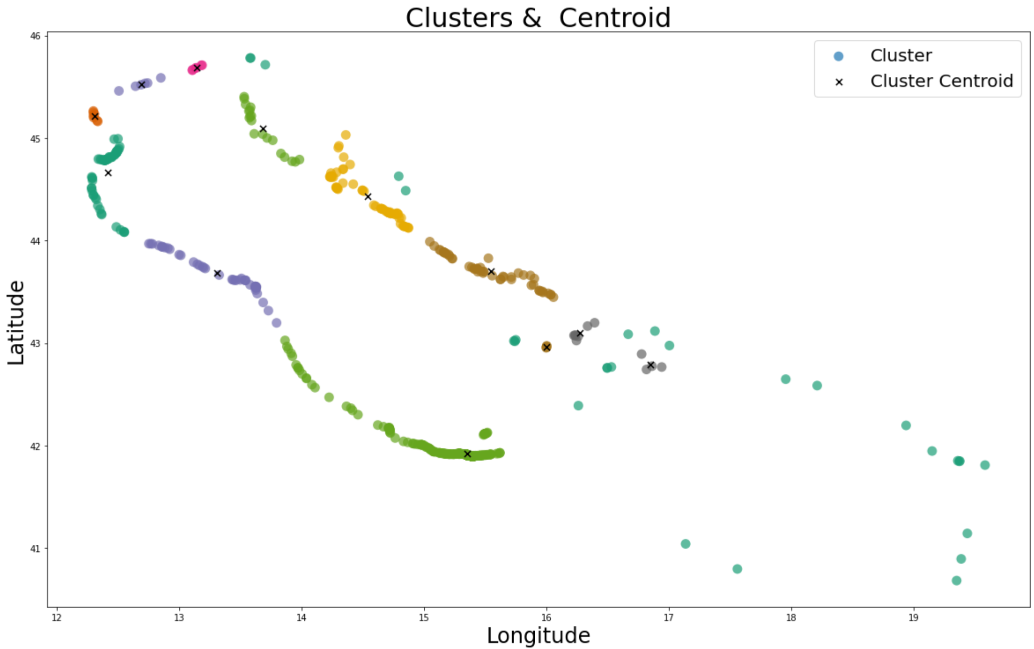


We obtained the maximum silhouette score with an eps of 22 km.
After that we decided to use the same procedure to find the best minPts parameters. The results are shown below:

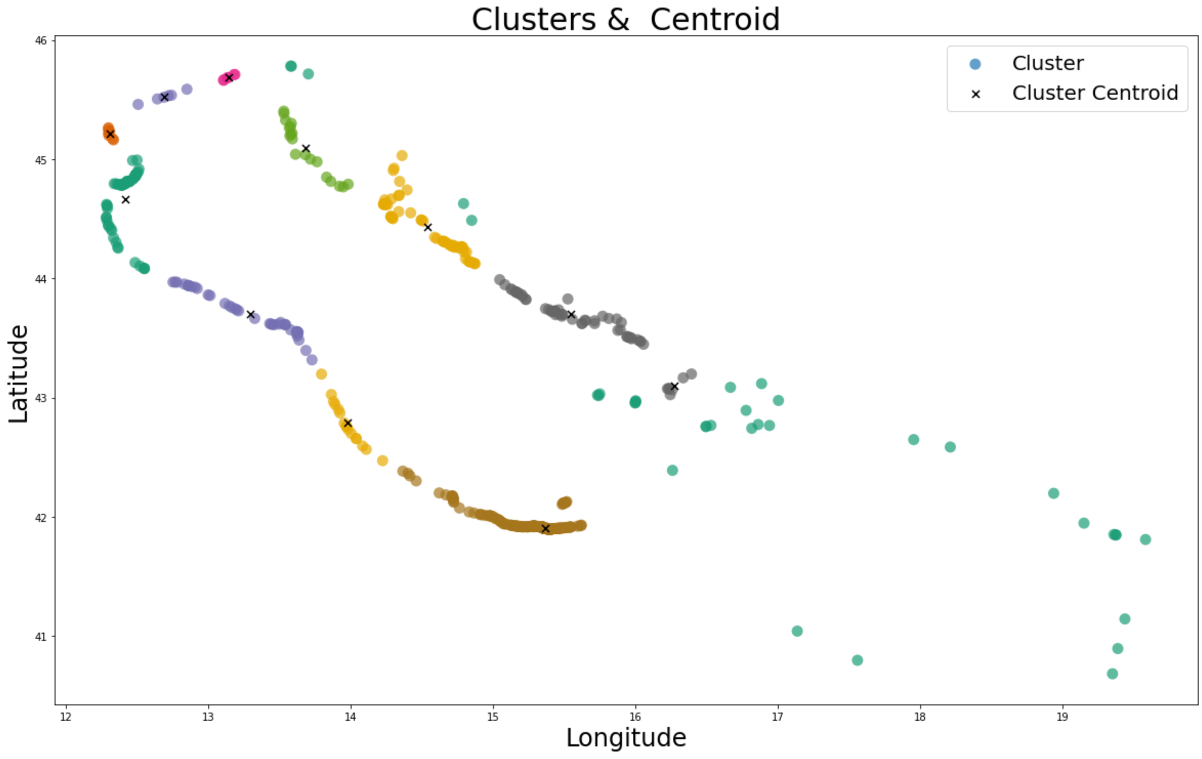




In the theory, it is suggested to set the minPts to a value that is 2 times the dimension of the dataset: 2*D, thus in this case, since the dataset is in 2D, the result is 4. With a minPts set to 4, we obtained the following clusterization result (12 clusters):



It can be noticed from the plot above, that the algorithm find a very large cluster (the green points that takes the area from Abruzzo and Puglia.
But setting this parameter to 5 as obtained calculating the silhouette score, we obtained 11 clusters and the above mentioned cluster divided into 2 parts, thing that we prefer.



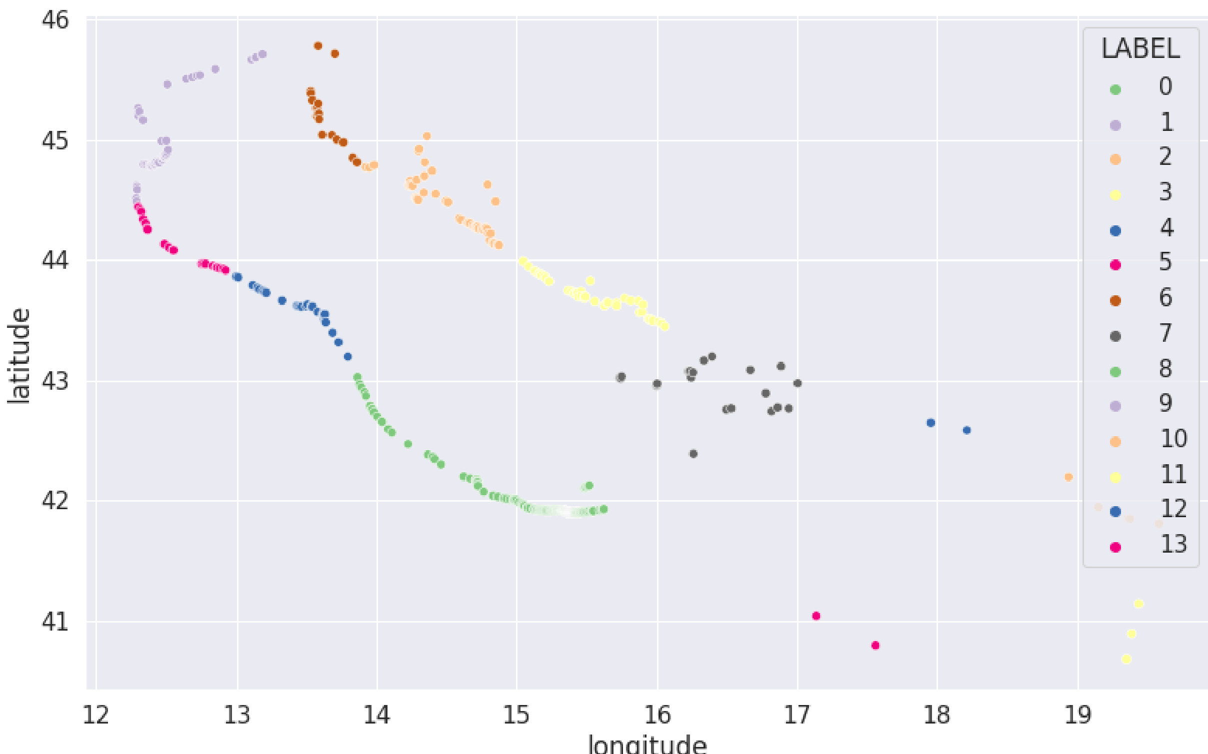
OPTICS (Ordering Points to Identify the Clustering Structure) algorithm

This algorithm can be seen as a generalization of DBSCAN that replaces the eps parameter with a maximum value that mostly affects performance. MinPts becomes the minimum cluster size to find. It produces a hierarchical clustering instead of the simple data partitioning that DBSCAN produces. It's a density-based algorithm similar to DBSCAN, but it's better because it can find meaningful clusters in data that varies in density. It does this by ordering the data points so that the closest points are neighbours in the ordering. This makes it easier to detect different density clusters.

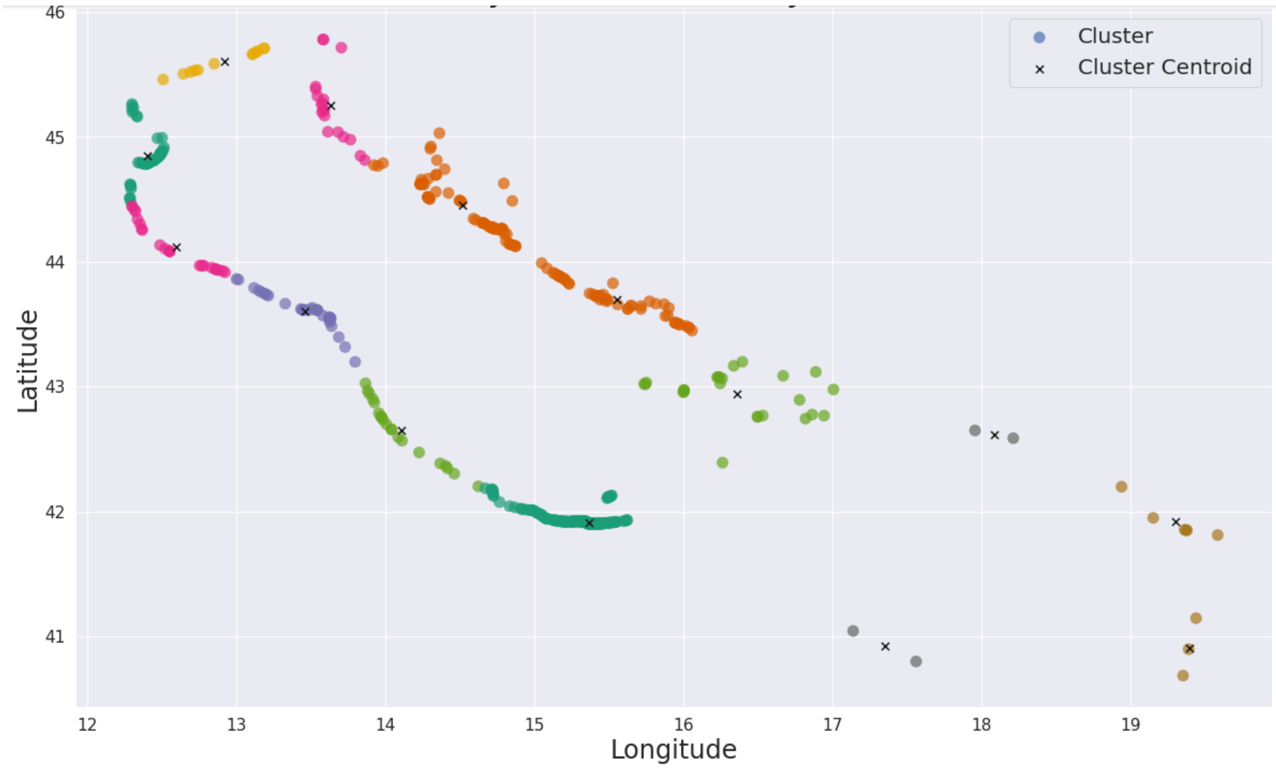
The problem of this method is that it keeps the number of elements inside the cluster closer to the minimum cluster size, so each clusters have nearly the same amount of data.

Mean-Shift clustering algorithm

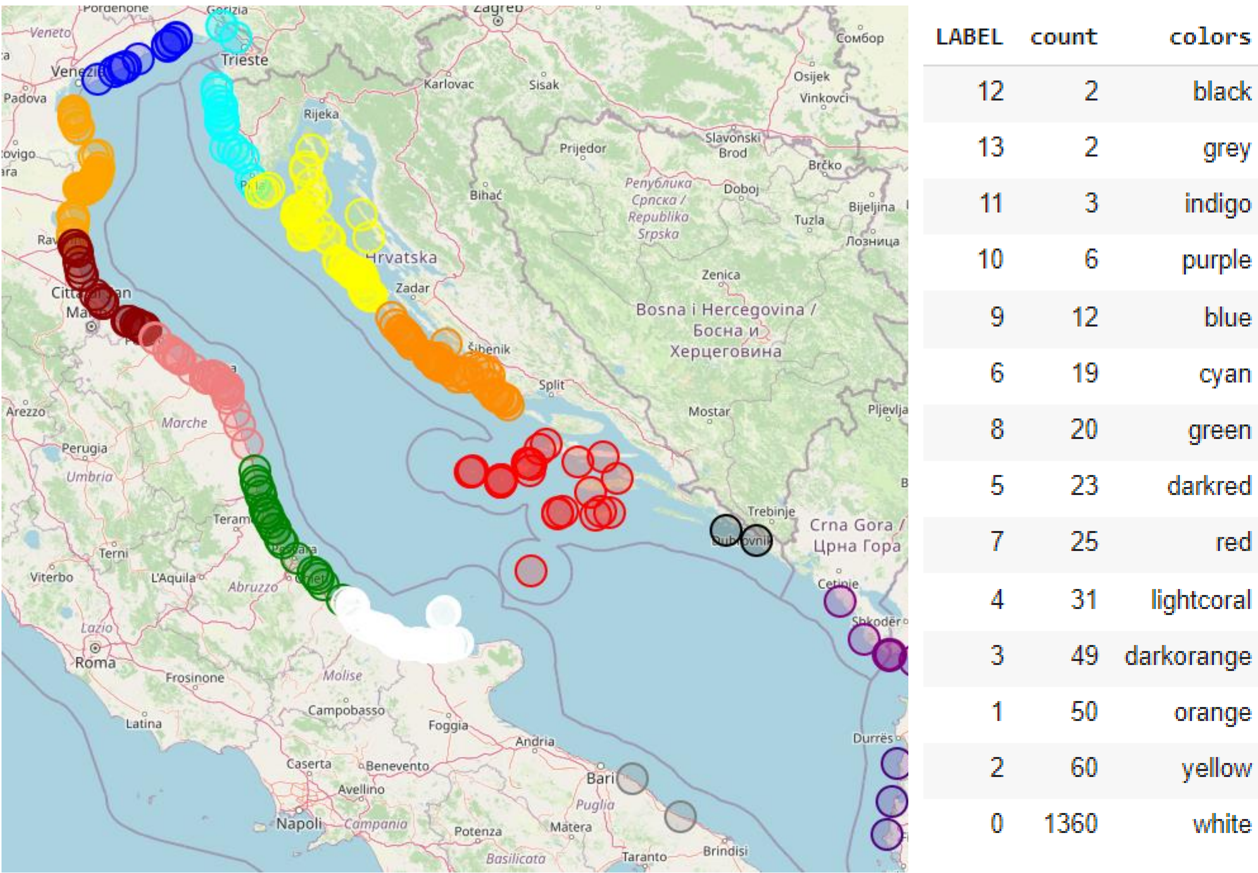
This algorithm is particularly useful for handling images and computer vision processing.
Mean-shift finds clusters without an initial number of clusters being set.
It is a non-parametric, density-based clustering algorithm, it is particularly useful for datasets where the clusters have arbitrary shapes and are not well-separated by linear boundaries.
The basic idea behind is to shift each data point towards the mode (i.e., the highest density) of the distribution of points within a certain radius. The algorithm iteratively performs these shifts until the points converge to a local maximum of the density function. These local maxima represent the clusters in the data.
Mean-shift builds upon the concept of kernel density estimation. It is a method to estimate the underlying distribution also called the probability density function for a set of data. It works by placing a kernel on each point in the data set. Adding up all of the individual kernels generates a probability surface example density function. Depending on the kernel bandwidth parameter used, the resultant density function will vary.
With our dataset it finds 14 clusters; on the following images you can see the number of clusters and the clusters with their centroids.



LABEL
0
1
2
3
4
5
6
7
8
9
10
11
12
13
latitude
longitude
46
45
44
43
42
41
12
13
14
15
16
17
18
19



On the following image the result is shown together with the table that shows the sources counts for each color.



LABEL	count	colors
12	2	black
13	2	grey
11	3	indigo
10	6	purple
9	12	blue
6	19	cyan
8	20	green
5	23	darkred
7	25	red
4	31	lightcoral
3	49	darkorange
1	50	orange
2	60	yellow
0	1360	white

Affinity Propagation clustering algorithm and other algorithms

With this algorithm, you have to set a parameter called damping factor [0.5,1.0). It represents the extent to which the current value is maintained relative to incoming values.
With values of this parameters ≤0.7 we obtained no clusters, with 0.8 148 clusters and with 0.9 159 clusters. Thus, this algorithm is not useful for our work.

Also the Agglomerative Hierarchy clustering algorithm and the Gaussian Mixture Model algorithm have been analysed but, they are not suitable for our case of study.

Results comparison

Considering polygons that identify some rivers firstly, and then some harbours and industries, a mask is performed in order to find if a particular cluster centroid falls inside a polygon, in such a way to link that cluster to a source of marine litter. This procedure is performed for the best three of the algorithms presented before: K-Means, DBSCAN and Mean-Shift.

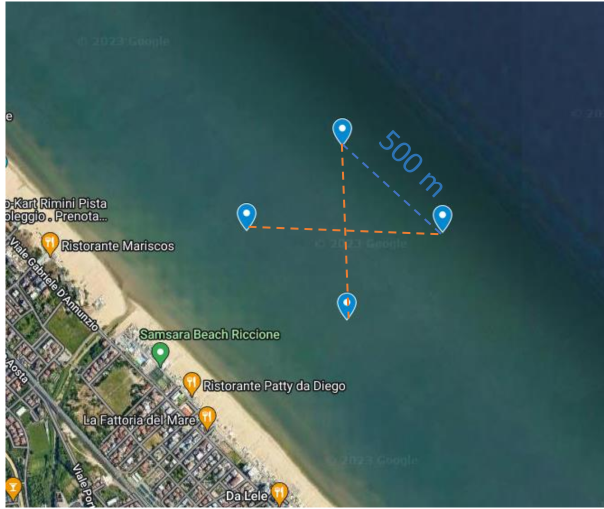
In the following tables, the results are shown: the centroid coordinates, the polygon name, the cluster label, the amount of element inside that cluster, the ranking position based on the count of elements considering the overall clusters.

Rivers					
DBSCAN (11 clusters)					
Centroid longitude	**Centroid latitude**	**Polygon name**	**Cluster label**	**Count of elements**	**Ranking position**
12.41389	44.66663	Reno river	0	56	2
12.30901	45.21414	Brenta-Adige rivers	1	7	9
12.69083	45.52348	Piave river	2	6	11
Mean-Shift (14 clusters)					
Centroid longitude	**Centroid latitude**	**Polygon name**	**Cluster label**	**Count of elements**	**Ranking position**
12.39117	44.78174	Po river	1	50	3
14.07829	42.67326	Vomano river	8	20	8
19.39282	40.90818	Semam river	11	3	12
K-Means (12 clusters)					
No linking					

Harbours and industries					
DBSCAN (11 clusters)					
Centroid longitude	**Centroid latitude**	**Polygon name**	**Cluster label**	**Count of elements**	**Ranking position**
13.68303	45.09470	Rovigno harbour	5	19	7
13.97743	42.78709	Giulianova harbour	6	16	8
Mean-Shift (14 clusters)					
Centroid longitude	**Centroid latitude**	**Polygon name**	**Cluster label**	**Count of elements**	**Ranking position**
13.45814	43.60549	Ancona harbour and rafinery	4	31	5
18.08349	42.61729	Ragusa harbour	12	2	13
K-Means (12 clusters)					
No linking					

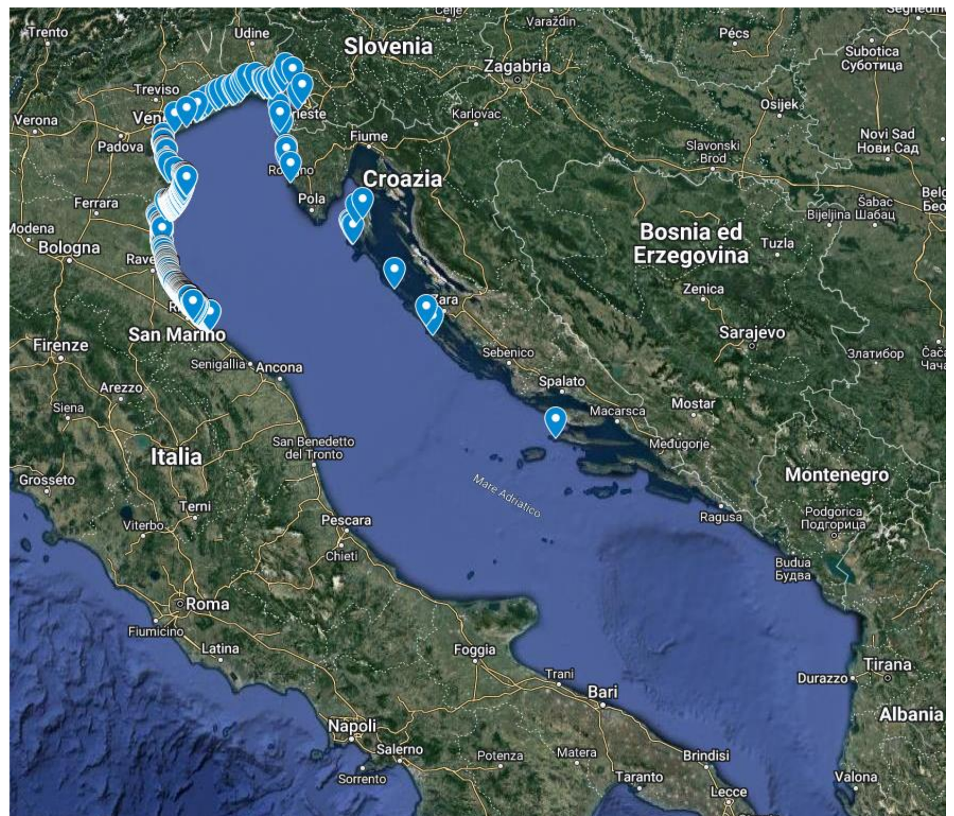
Tests: 334 days of simulation from the Riccione beach in Emilia Romagna

The same test above is repeated for the Riccione beach.



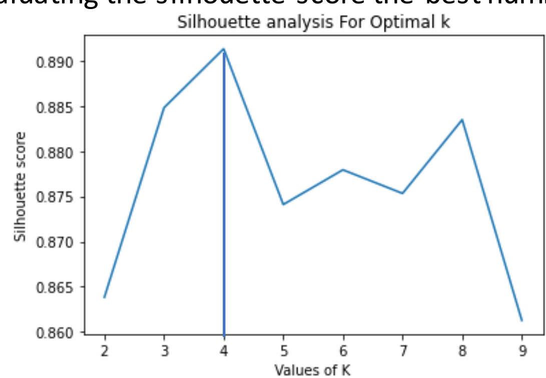
Here below the possible sources found after 334 days of simulation are shown.

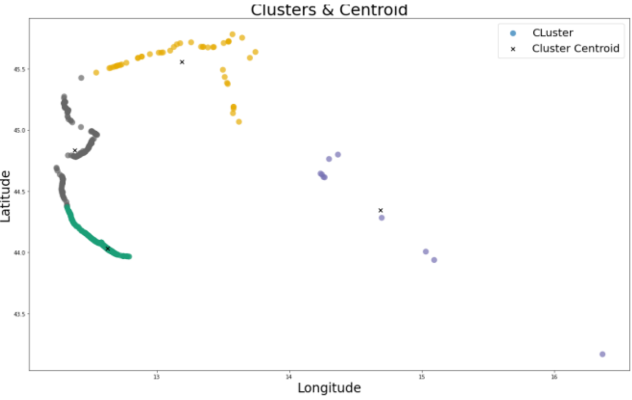
Interreg
Italy - Croatia
MARLESS
EUROPEAN UNION

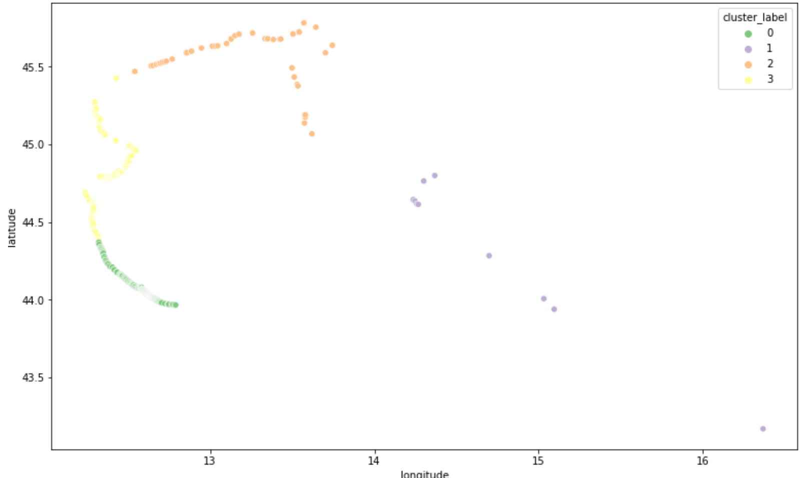

Trento
Udine
Slovenia
Varaždin
Pécs
Subotica
Суботица
Treviso
Zagabria
Verona
Karlovac
Osijek
Padova
Fiume
Slavonski Brod
Novi Sad
Нови Сад
Croazia
Ferrara
Pola
Šabac
Bijeljina
Bologna
Bosnia ed Erzegovina
Tuzla
Zenica
San Marino
Sarajevo
Firenze
Sebenico
Senigallia
Ancona
Spalato
Arezzo
Siena
Macarsca
Mostar
Italia
Međugorje
San Benedetto del Tronto
Grosseto
Mare Adriatico
Montenegro
Terni
Ragusa
Podgorica
Подгорица
Viterbo
Pescara
Chieti
Budua
Будва
Roma
Fiumicino
Latina
Foggia
Trani
Tirana
Durazzo
Bari
Albania
Napoli
Salerno
Potenza
Matera
Brindisi
Sorrento
Taranto
Lecce
Valona
Otranto

K-Means algorithm

Evaluating the silhouette score the best number of clusters found is k = 4.



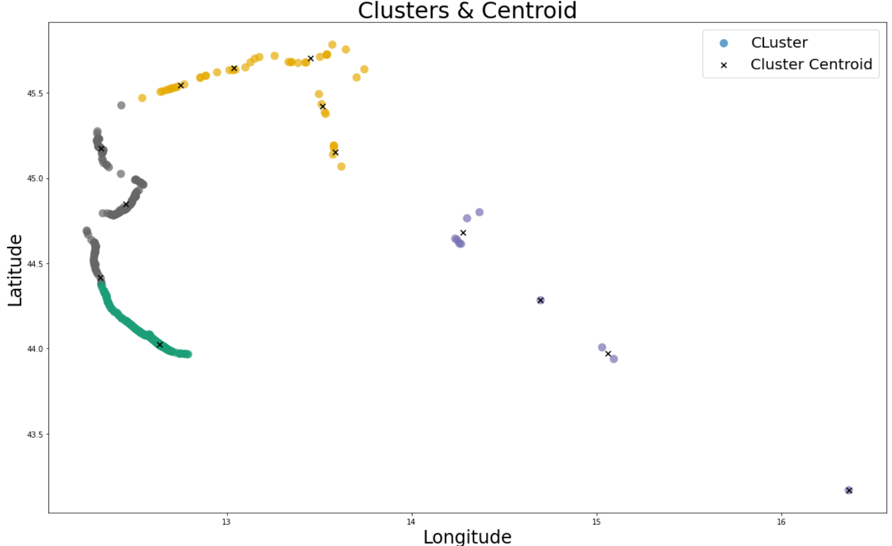


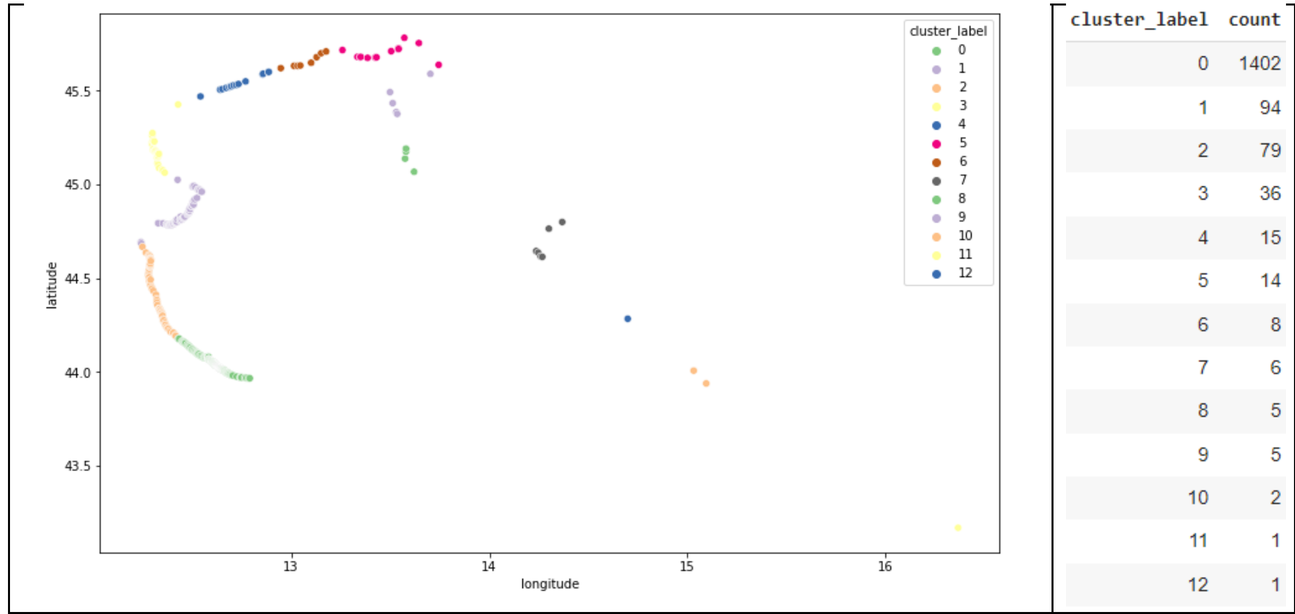


cluster_label	count
0	1435
3	176
2	47
1	10

Mean-Shift clustering algorithm

With the algorithm Mean-Shift 13 clusters are found.





cluster_label	count
0	1402
1	94
2	79
3	36
4	15
5	14
6	8
7	6
8	5
9	5
10	2
11	1
12	1

DBSCAN (Density-based spatial clustering of applications with noise) algorithm

For the DBSCAN algorithm, with the silhouette method, we found 2 possible values for the parameter eps: 10.5 km and 83 km. The second value has a higher silhouette score. Then the best minPts value for both these 2 eps values is searched.

For eps = 10.5 km, we found two values for the minPts parameter: 4 and 26 (with the latter with a higher value of silhouette score). As you can see from the plots below the best clusterization is performed with the minPts equal to 4.

Eps 10.5 km; minPts 4; 8 clusters found

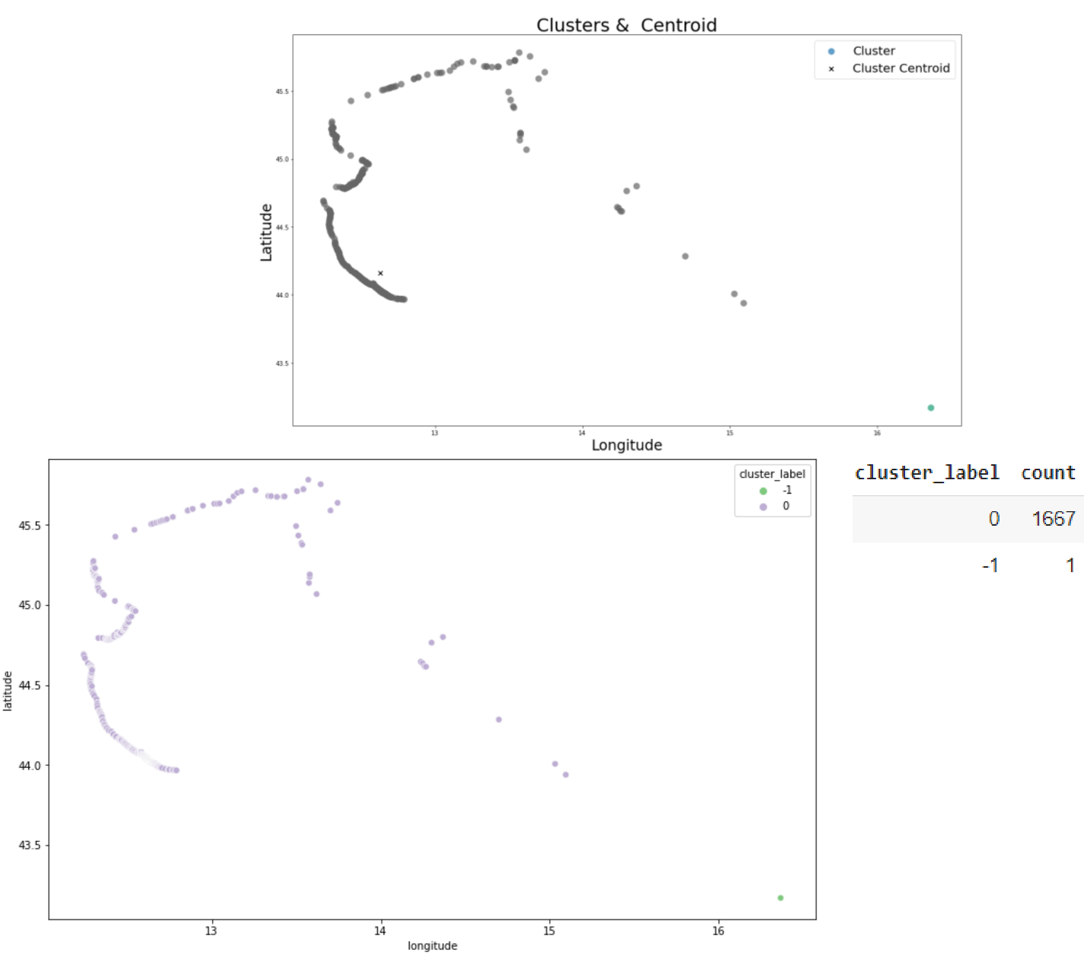
Clusters & Centroid

cluster_label	count
0	1483
1	127
2	23
3	12
-1	10
5	5
4	4
6	4

Eps 10.5 km; minPts 26; 5 clusters found

cluster_label	count
2	1438
3	91
-1	59
0	45
1	35

For the value eps = 83 km, we found the highest silhouette score at minPts equal to 8.
From the images below, it can be noticed that the algorithm find a unique cluster, so the best pair of parameter to be chosen are eps = 10.5 km and minPts = 4.



cluster_label	count
0	1667
-1	1

Results comparison

Considering polygons that identify some rivers firstly, and then some harbours and industries, a mask is performed in order to find if a particular cluster centroid falls inside a polygon, in such a way to link that cluster to a source of marine litter. This procedure is performed for the three algorithms presented before: K-Means, DBSCAN and Mean-Shift.

In the following tables, the results are shown: the centroid coordinates, the polygon name, the cluster label, the amount of element inside that cluster, the ranking position based on the count of elements considering the overall clusters.

Rivers					
DBSCAN (8 clusters)					
Centroid longitude	**Centroid latitude**	**Polygon name**	**Cluster label**	**Count of elements**	**Ranking position**
12.41736	44.93931	Po river	1	127	2
12.88441	45.59113	Livenza river	2	23	3
Mean-Shift (13 clusters)					
Centroid longitude	**Centroid latitude**	**Polygon name**	**Cluster label**	**Count of elements**	**Ranking position**
12.45553	44.84761	Po river	1	94	2
12.31784	45.17529	Brenta-Adige rivers	3	36	4
12.74859	45.54436	Piave river	4	15	5
K-Means (4 clusters)					
Centroid longitude	**Centroid latitude**	**Polygon name**	**Cluster label**	**Count of elements**	**Ranking position**
12.38222	44.83537	Po river	3	176	2

Harbours and industries					
DBSCAN (8 clusters)					
Centroid longitude	**Centroid latitude**	**Polygon name**	**Cluster label**	**Count of elements**	**Ranking position**
13.58695	45.15219	Rovigno harbour	5	5	6
Mean-Shift (13 clusters)					
Centroid longitude	**Centroid latitude**	**Polygon name**	**Cluster label**	**Count of elements**	**Ranking position**
12.31697	44.41464	Marina Ravenna harbour	2	79	3
15.06275	43.97263	Zara harbour	10	2	11
13.58695	45.15219	Rovigno harbour	8	5	9
K-Means (4 clusters)					
No linking					

Conclusions

Considering the results obtained with all the considered clustering algorithms, Mean-Shift is the most suitable for our study. Since we have to perform massive analyses on the C3HPC cluster, this algorithm allows to have a non-subjective intervention of the users to choose the right value of clusters. Moreover, considering the tables that compare the polygon masks between all the algorithms, there is a good intersection of results between DBSCAN and Mean-Shift and with the latter we have more matches between cluster centroids and possible sources.

Test considering all the sources used in the deliverable D3.3.2

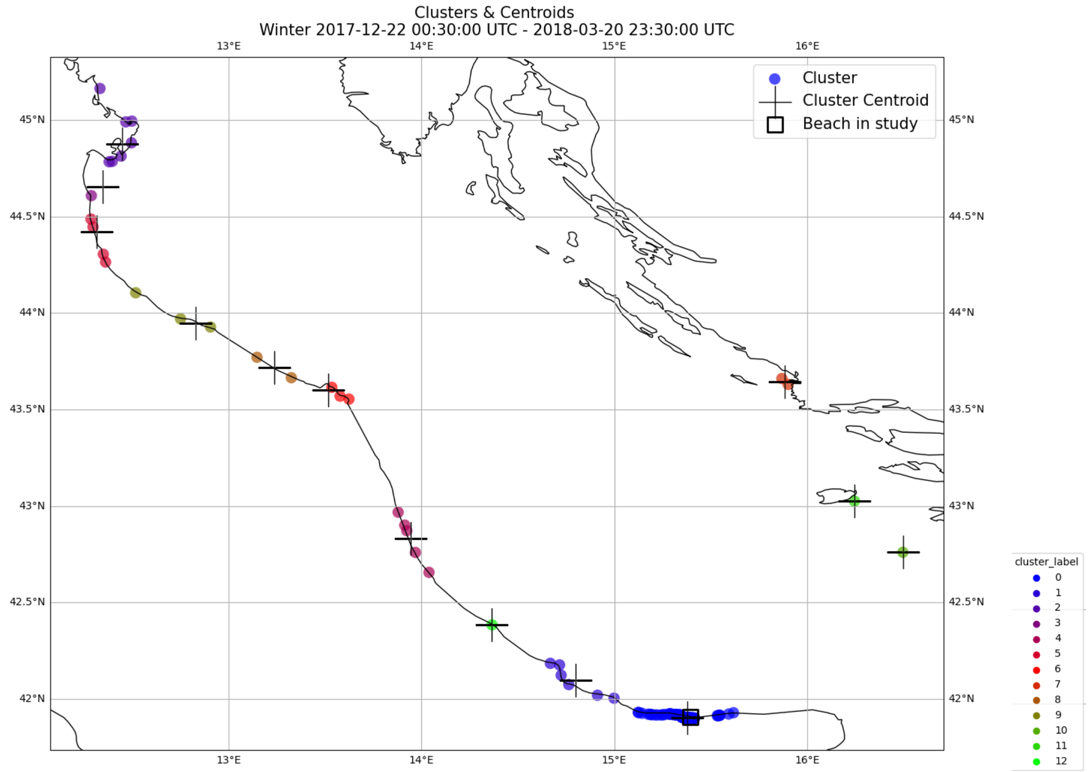
Now, considering the Mean-Shift algorithm we have performed the polygon masks with all the sources considered in the deliverable D3.3.2 for the results obtained for the beaches of Lesina and Riccione.

The polygon masks analysis is performed for 5 period of time: one for each season (the autumn season is shorter because of wind and current limitation of data) and one considering a full year (in our case 334 days).

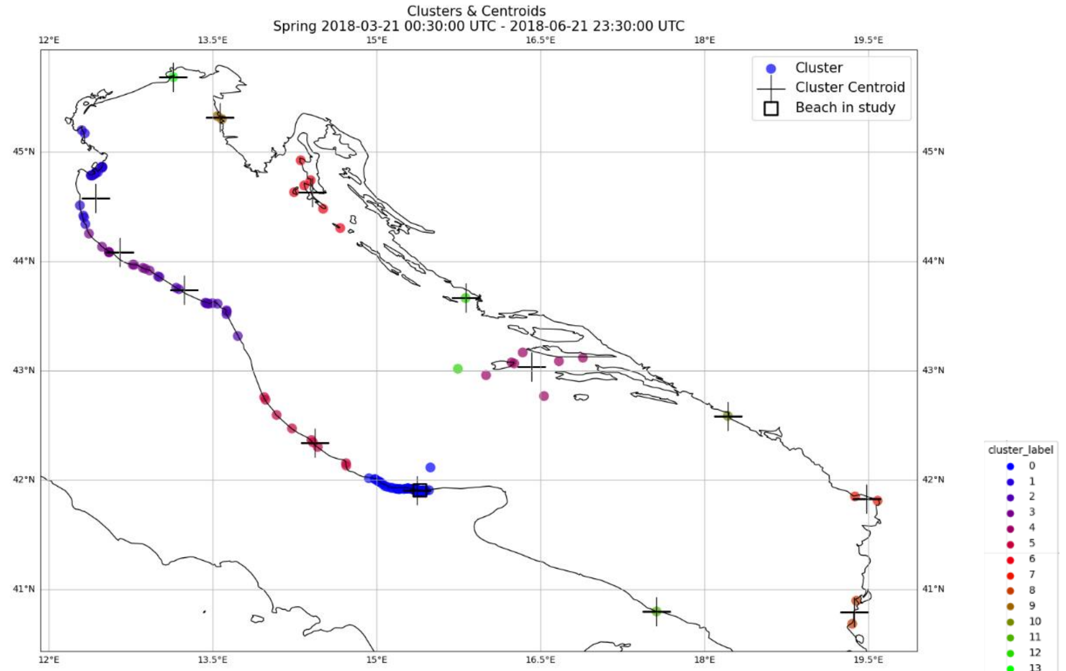
In the following sections this study is presented for each beach.

Lesina beach

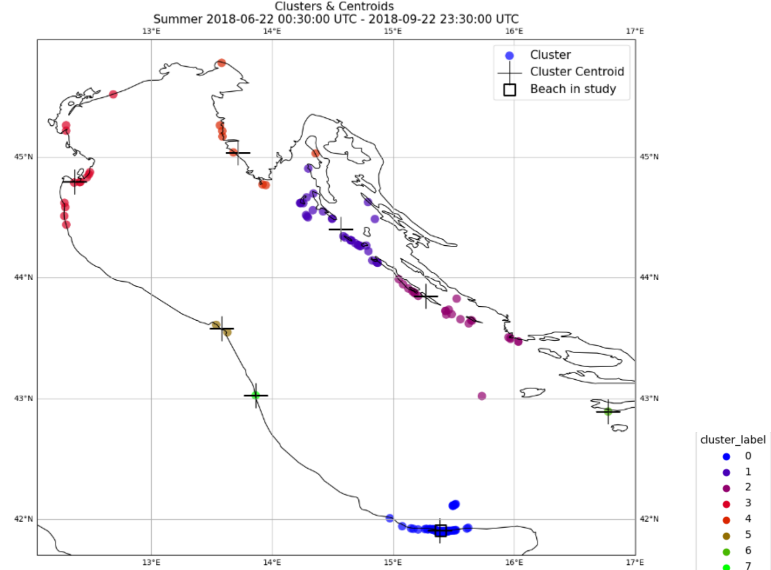
WINTER				
Polygons mask				
Longitude	**Latitude**	**Polygon name**	**Cluster label**	**Counts**
15.3785	41.9038	Lesina and Varano lakes	0	408
12.4477	44.8740	Po river	2	7
14.7974	42.0972	Trigno river	1	6
13.9451	42.8311		4	5
12.3176	44.4212	Marina Ravenna harbour	5	4
12.8291	43.9481	Foglia river	9	3
13.5161	43.6003	Ancona harbour and rafinery	6	3
13.2369	43.7173	Cesano-Misa rivers and Senigallia harbour	8	2
15.8848	43.6450		7	2
12.3494	44.6532	Reno river	3	1
16.2446	43.0251	Spalato harbour and Cettina-Jedro rivers	11	1
14.3660	42.3841	Ortona harbour	12	1
16.4956	42.7599		10	1


Clusters & Centroids
Winter 2017-12-22 00:30:00 UTC - 2018-03-20 23:30:00 UTC
13°E
14°E
15°E
16°E
45°N
44.5°N
44°N
43.5°N
43°N
42.5°N
42°N
Cluster
Cluster Centroid
Beach in study
cluster_label
0
1
2
3
4
5
6
7
8
9
10
11
12

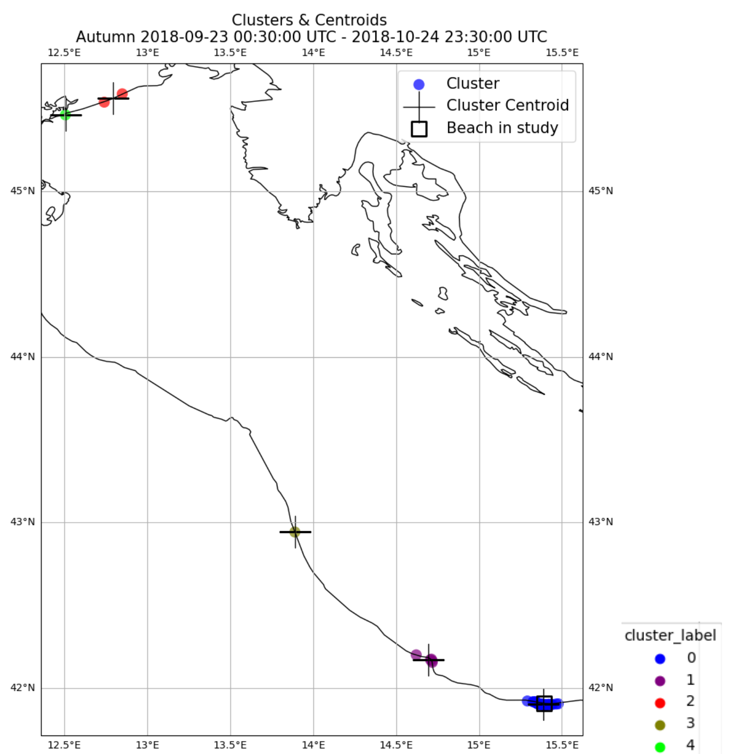
SPRING				
Polygons mask				
Longitude	Latitude	Polygon name	Cluster label	Counts
15.3691	41.9058	Lesina and Varano lakes	0	365
12.4290	44.5789	Reno river	1	18
13.2378	43.7362	Cesano-Misa rivers and Senigallia harbour	2	14
12.6461	44.0830	Cesenatico-Rimini habrours and Rubicone-Uso-Marecchia rivers	3	10
14.4320	42.3377	Ortona harbour	5	9
16.4160	43.0341		4	7
14.4091	44.6296	Fiume harbour	6	6
19.3699	40.7901	Semam river	8	2
13.5625	45.3142	Quieto river	9	2
19.4800	41.8302	Ishem-Mat-Drin rivers and San Giovanni Medua harbour	7	2
15.8130	43.6642		12	2
13.1375	45.6853	Marano-Grado lagoon	13	1
18.2119	42.5862		10	1
17.5599	40.7975		11	1


Clusters & Centroids
Spring 2018-03-21 00:30:00 UTC - 2018-06-21 23:30:00 UTC
Cluster
Cluster Centroid
Beach in study
cluster_label
0
1
2
3
4
5
6
7
8
9
10
11
12
13
12°E
13.5°E
15°E
16.5°E
18°E
19.5°E
45°N
44°N
43°N
42°N
41°N

SUMMER				
Polygons mask				
Longitude	**Latitude**	**Polygon name**	**Cluster label**	**Counts**
15.3857	41.9078	Lesina and Varano lakes	0	338
14.5669	44.4053		1	32
15.2698	43.8501	Zara harbour and Zermagna river	2	23
12.3664	44.7985	Po river	3	13
13.7162	45.0384	Rovigno harbour	4	8
13.5843	43.5804	Ancona harbour and rafinery	5	2
16.7774	42.8933	Narenta river	6	1
13.8641	43.0266		7	1

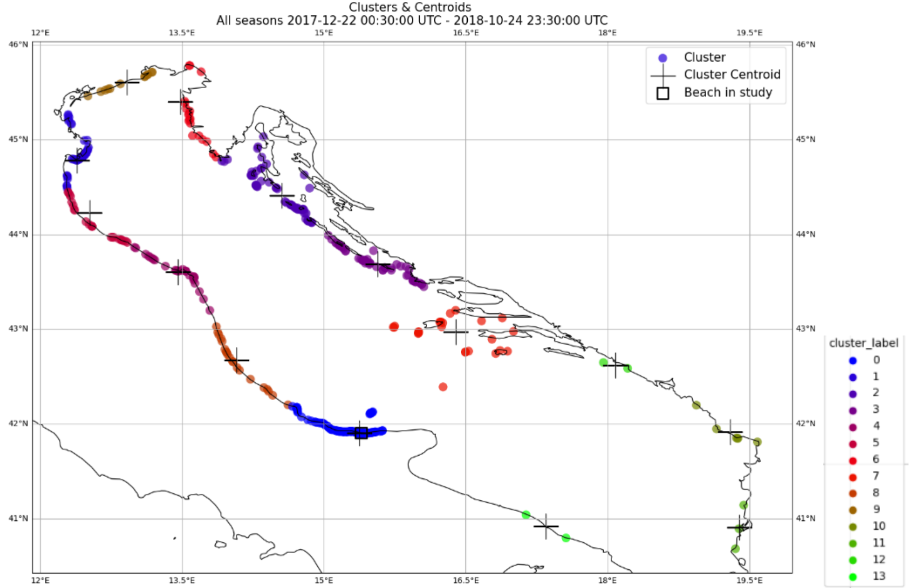


AUTUMN				
Polygons mask				
Longitude	Latitude	Polygon name	Cluster label	Counts
15.3871	41.9028	Lesina and Varano lakes	0	94
14.6957	42.1737	Vasto harbour	1	5
12.7950	45.5618		2	2
12.5066	45.4597	Venice lagoon	4	1
13.8901	42.9425	San Benedetto del Tronto harbour	3	1



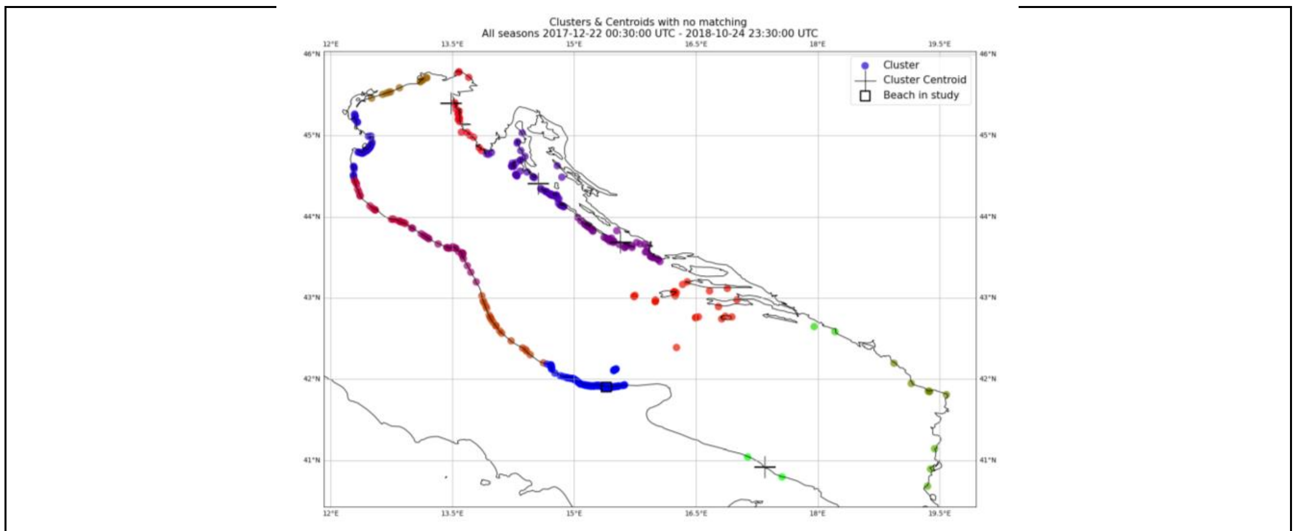
Clusters & Centroids
Autumn 2018-09-23 00:30:00 UTC - 2018-10-24 23:30:00 UTC
Cluster
Cluster Centroid
Beach in study
cluster_label
0
1
2
3
4
12.5°E
13°E
13.5°E
14°E
14.5°E
15°E
15.5°E
45°N
44°N
43°N
42°N

ALL SEASONS				
Polygons mask				
Longitude	**Latitude**	**Polygon name**	**Cluster label**	**Counts**
15.3738	41.9063	Lesina and Varano lakes	0	1360
14.5586	44.4113		2	60
12.3912	44.7817	Po river	1	50
15.5692	43.6891		3	49
13.4581	43.6055	Ancona harbour and refinery	4	31
16.3927	42.9706	Narenta river	7	25
12.5242	44.2250	Cesenatico-Rimini habrours and Rubicone-Uso-Marecchia rivers	5	23
14.0783	42.6733	Vomano river	8	20
13.4849	45.3998		6	19
12.9165	45.6047	Livenza river	9	12
19.2973	41.9177	Boiana river and Port Milena harbour	10	6
19.3928	40.9082	Semam river	11	3
18.0835	42.6173	Ragusa harbour	12	2
17.3485	40.9199		13	2



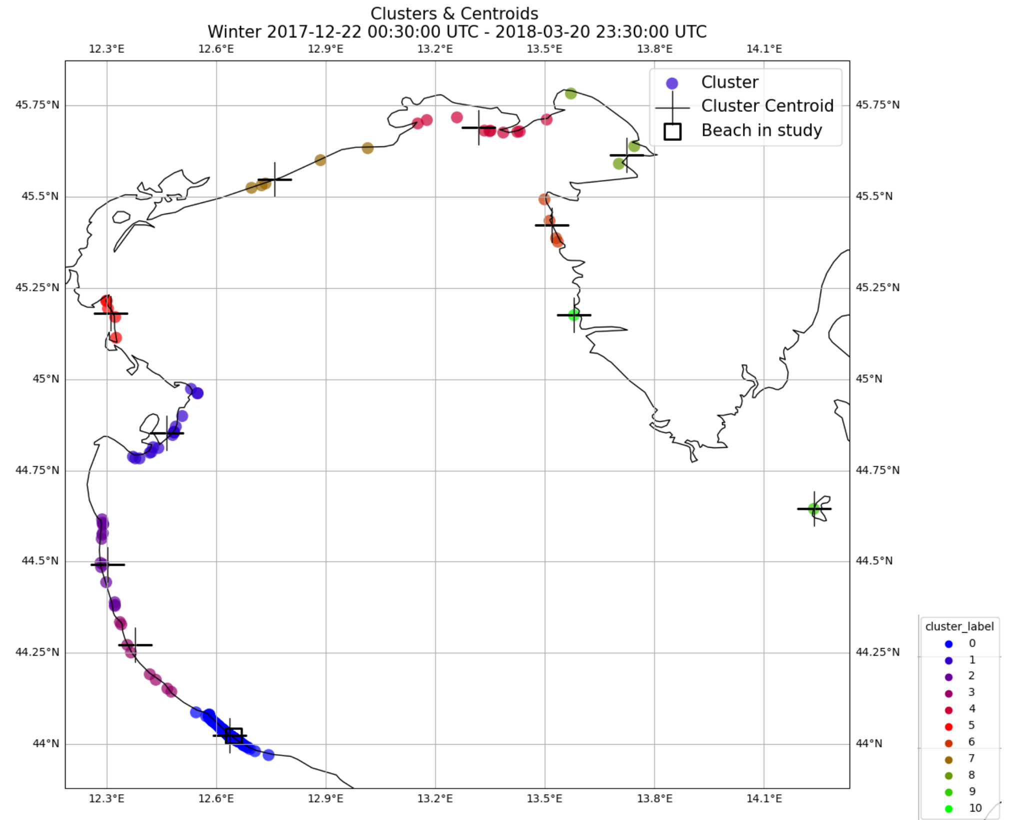
As you can notice there are some clusters that didn't have an associated polygon. In this case the matching is performed by hand looking where the centroid falls in the satellite maps.
On the following images the centroids with no matching are shown:

Clusters & Centroids with no matching Winter 2017-12-22 00:30:00 UTC - 2018-03-20 23:30:00 UTC	Clusters & Centroids with no matching Spring 2018-03-21 00:30:00 UTC - 2018-06-21 23:30:00 UTC
Clusters & Centroids with no matching Summer 2018-06-22 00:30:00 UTC - 2018-09-22 23:30:00 UTC	Clusters & Centroids with no matching Autumn 2018-09-23 00:30:00 UTC - 2018-10-24 23:30:00 UTC


Clusters & Centroids with no matching
All seasons 2017-12-22 00:30:00 UTC - 2018-10-24 23:30:00 UTC
Cluster
Cluster Centroid
Beach in study
12°E
13.5°E
15°E
16.5°E
18°E
19.5°E
46°N
45°N
44°N
43°N
42°N
41°N

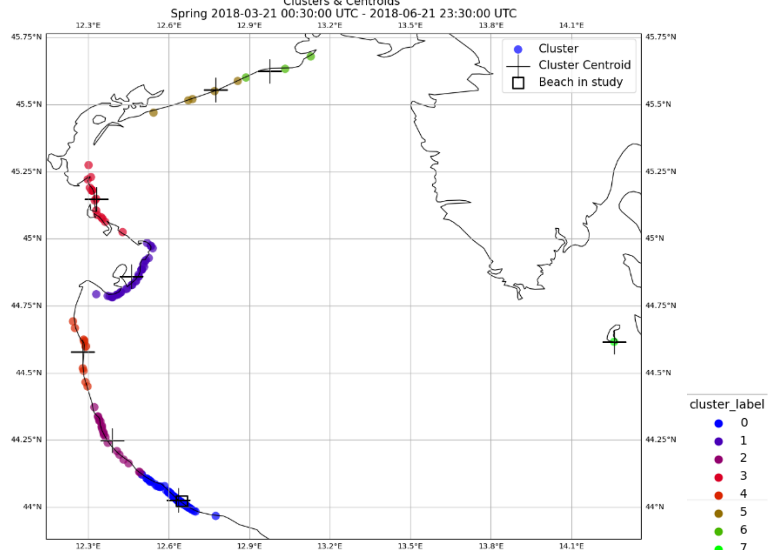
Riccione beach

WINTER				
Polygons mask				
Longitude	**Latitude**	**Polygon name**	**Cluster label**	**Counts**
12.6364	44.0254		0	369
12.4631	44.8533	Po river	1	16
12.3014	44.4921	Marina Ravenna harbour	2	14
13.3182	45.6887	Marano-Grado lagoon	4	10
12.3784	44.2727	Cesenatico-Rimini habrours and Rubicone-Uso-Marecchia rivers	3	8
12.3104	45.1811	Brenta Adige rivers	5	5
12.7603	45.5476	Piave river	7	5
13.5195	45.4222	Pirano-Isola-Koper harbours	6	4
13.7238	45.6140	Trieste harbour	8	3
14.2365	44.6447	Fiume harbour	9	1
13.5796	45.1751	Rovigno harbour	10	1

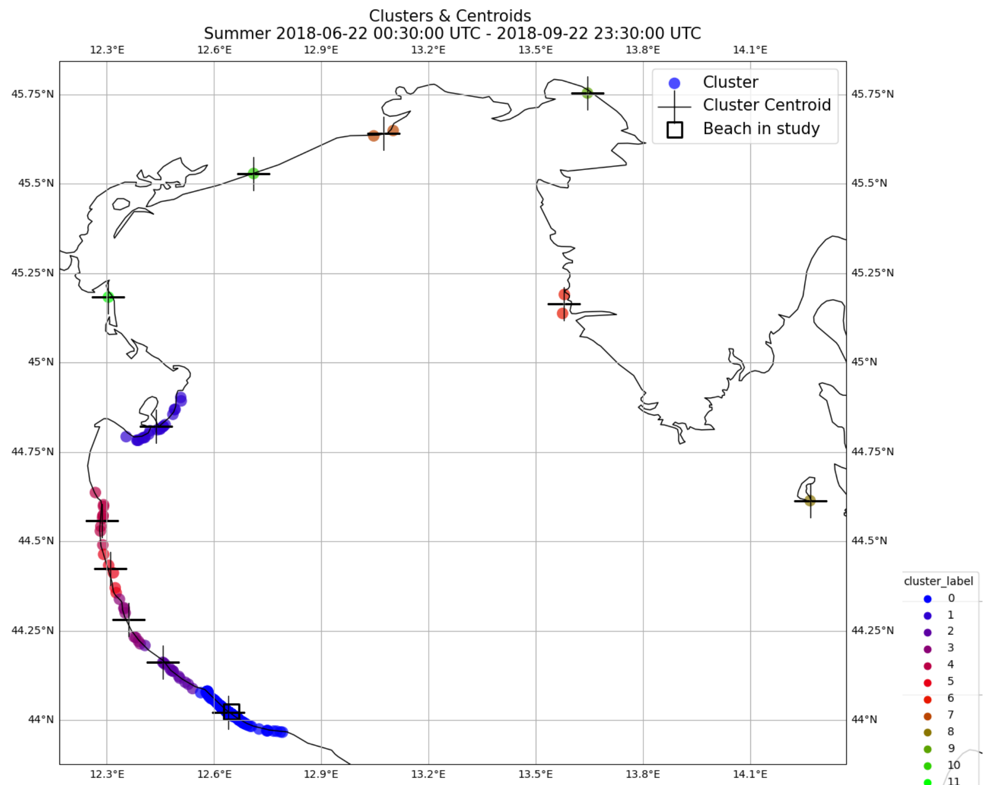


Clusters & Centroids
Winter 2017-12-22 00:30:00 UTC - 2018-03-20 23:30:00 UTC
Cluster
Cluster Centroid
Beach in study
12.3°E
12.6°E
12.9°E
13.2°E
13.5°E
13.8°E
14.1°E
45.75°N
45.5°N
45.25°N
45°N
44.75°N
44.5°N
44.25°N
44°N
cluster_label
0
1
2
3
4
5
6
7
8
9
10

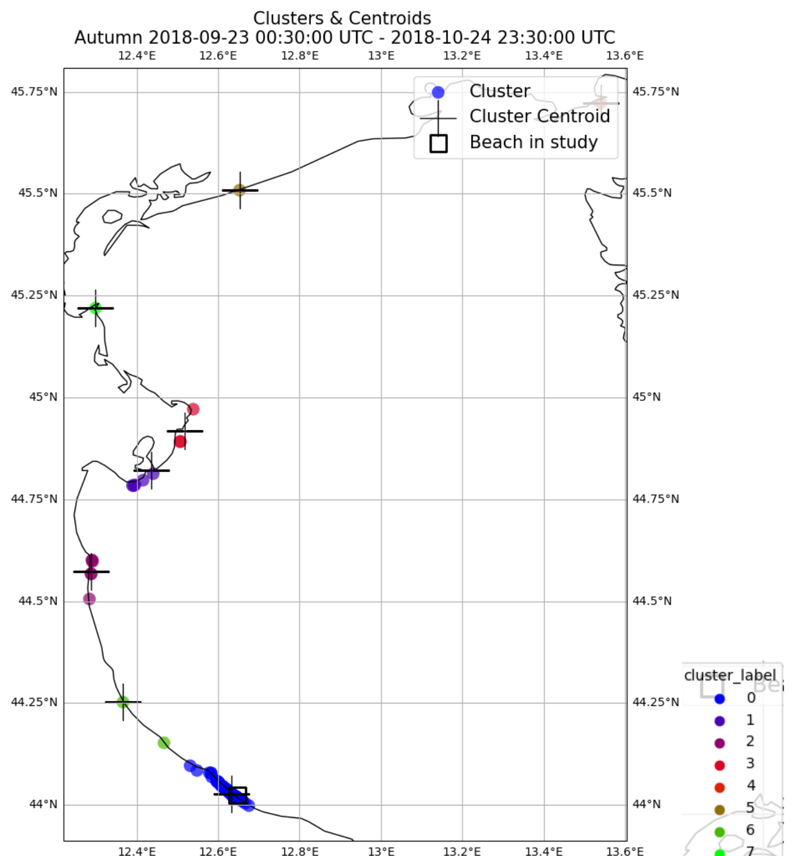
SPRING				
Polygons mask				
Longitude	**Latitude**	**Polygon name**	**Cluster label**	**Counts**
12.6350	44.0258		0	353
12.4603	44.8595	Po river	1	35
12.3893	44.2495	Cesenatico-Rimini habrours and Rubicone-Uso-Marecchia rivers	2	21
12.3298	45.1476	Brenta Adige rivers	3	16
12.2798	44.5780	Reno river	4	11
12.7752	45.5546	Piave river	5	5
12.9766	45.6249		6	3
14.2590	44.6162	Fiume harbour	7	1



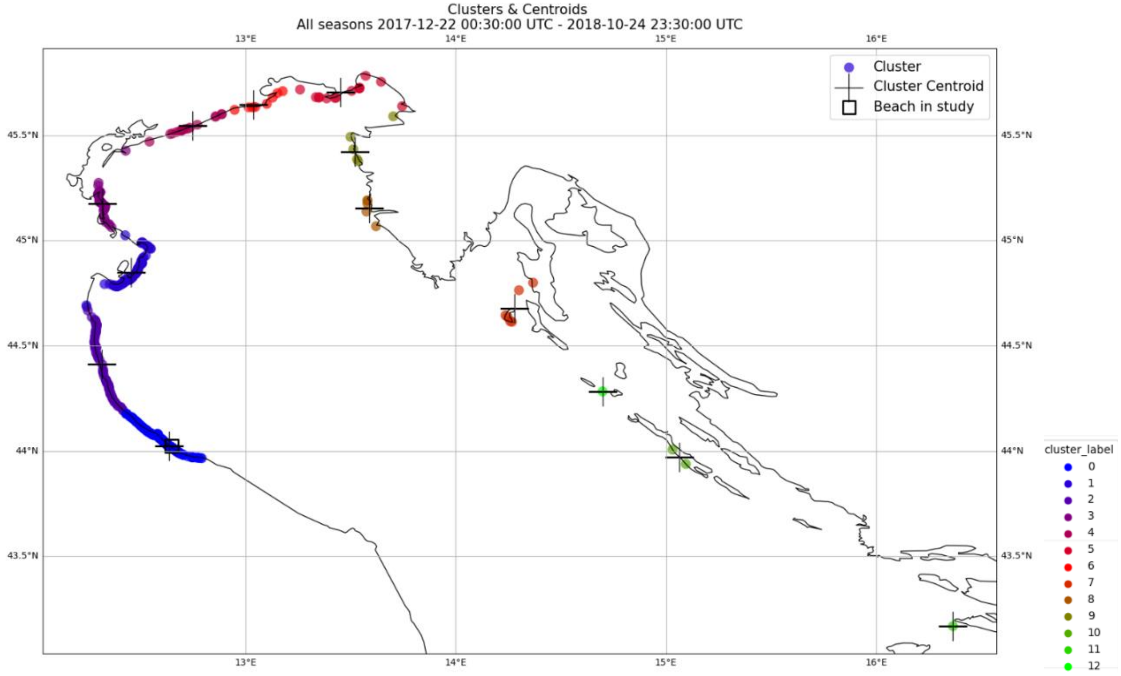
SUMMER				
Polygons mask				
Longitude	Latitude	Polygon name	Cluster label	Counts
12.6406	44.0222		0	384
12.4381	44.8210	Po river	1	20
12.4566	44.1616	Cesenatico-Rimini habrours and Rubicone-Uso-Marecchia rivers	2	13
12.2862	44.5569		4	9
12.3619	44.2814	Cesenatico-Rimini habrours and Rubicone-Uso-Marecchia rivers	3	8
12.3089	44.4239	Marina Ravenna harbour	5	5
13.0740	45.6418	Tagliamento river	7	2
13.5776	45.1642	Rovigno harbour	6	2
12.3044	45.1832	Brenta Adige rivers	11	1
13.6446	45.7542	Isonzo river - Monfalcone harbour	9	1
12.7106	45.5290	Piave river	10	1
14.2673	44.6139	Fiume harbour	8	1


Clusters & Centroids
Summer 2018-06-22 00:30:00 UTC - 2018-09-22 23:30:00 UTC
Cluster
Cluster Centroid
Beach in study
cluster_label
0
1
2
3
4
5
6
7
8
9
10
11
12.3°E
12.6°E
12.9°E
13.2°E
13.5°E
13.8°E
14.1°E
45.75°N
45.5°N
45.25°N
45°N
44.75°N
44.5°N
44.25°N
44°N

AUTUMN				
Polygons mask				
Longitude	**Latitude**	**Polygon name**	**Cluster label**	**Counts**
12.6334	44.0274		0	132
12.2889	44.5729	Reno river	2	6
12.4357	44.8207	Po river	1	5
12.5179	44.9181	Po river	3	3
12.3666	44.2521	Cesenatico-Rimini habrours and Rubicone-Uso-Marecchia rivers	6	2
12.6529	45.5086	Venice lagoon	5	1
12.2990	45.2193	Brenta Adige rivers	7	1
13.5402	45.7225	Isonzo river - Monfalcone harbour	4	1


Clusters & Centroids
Autumn 2018-09-23 00:30:00 UTC - 2018-10-24 23:30:00 UTC
Cluster
Cluster Centroid
Beach in study
cluster_label
0
1
2
3
4
5
6
7
12.4°E
12.6°E
12.8°E
13°E
13.2°E
13.4°E
13.6°E
45.75°N
45.5°N
45.25°N
45°N
44.75°N
44.5°N
44.25°N
44°N

ALL SEASONS				
Polygons mask				
Longitude	**Latitude**	**Polygon name**	**Cluster label**	**Counts**
12.6369	44.0249		0	1402
12.4555	44.8476	Po river	1	94
12.3170	44.4146	Marina Ravenna harbour	2	79
12.3178	45.1753	Brenta Adige rivers	3	36
12.7486	45.5444	Piave river	4	15
13.4525	45.7033	Marano-Grado lagoon	5	14
13.0375	45.6458	Tagliamento river	6	8
14.2797	44.6791	Fiume harbour	7	6
13.5869	45.1522	Rovigno harbour	8	5
13.5195	45.4222	Pirano-Isola-Koper harbours	9	5
15.0628	43.9726	Zara harbour and Zermagna river	10	2
16.3650	43.1685	Spalato harbour and Cettina-Jedro rivers	11	1
14.6983	44.2834	Zara harbour and Zermagna river	12	1



The centroids with no matching for each period of time are shown in the following images, for all of these unmatched centroids we are going to choose the best sources to associate to them:

Clusters & Centroids with no matching
Winter 2017-12-22 00:30:00 UTC - 2018-03-20 23:30:00 UTC

Cluster
Cluster Centroid
Beach in study

Clusters & Centroids with no matching
Spring 2018-03-21 00:30:00 UTC - 2018-06-21 23:30:00 UTC

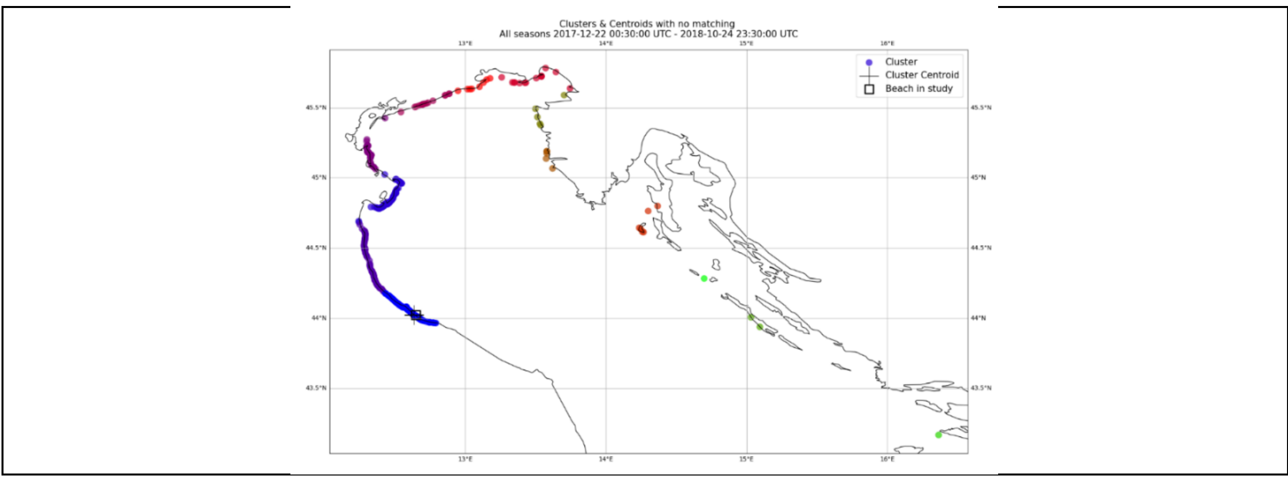
Cluster
Cluster Centroid
Beach in study

Clusters & Centroids with no matching
Summer 2018-06-22 00:30:00 UTC - 2018-09-22 23:30:00 UTC

Cluster
Cluster Centroid
Beach in study

Clusters & Centroids with no matching
Autumn 2018-09-23 00:30:00 UTC - 2018-10-24 23:30:00 UTC

Cluster
Cluster Centroid
Beach in study



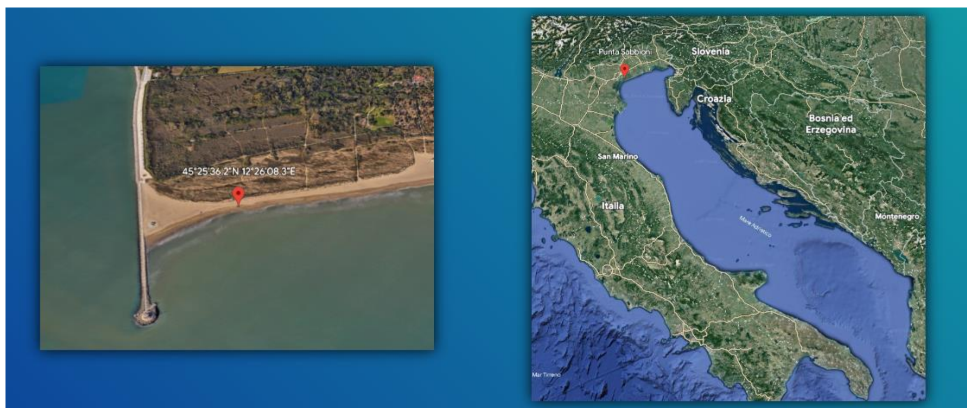
Simulation results and summaries

After having defined the method, the backtrajectories simulations have been performed for 330 days with a continuous release of 3 particles per hour. The beaches chosen are: Grado Pineta, Punta Sabbioni, Riccione, Bosco Isola Lesina, Fasano and Zambratija. For each of them the statistical analysis and the clusterization is performed for different periods of time: annual (2017/12/22 - 2018/10/24), winter (2017/12/22 - 2018/03/20), spring (2018/03/21 - 2018/06/21), summer (2018/96/22 - 2018/09/22) and autumn (2018/09/23 - 2018/10/24). For the autumn we have less days of simulation since with the C3HPC limits of computational time we were able to perform only 330 days of simulation.

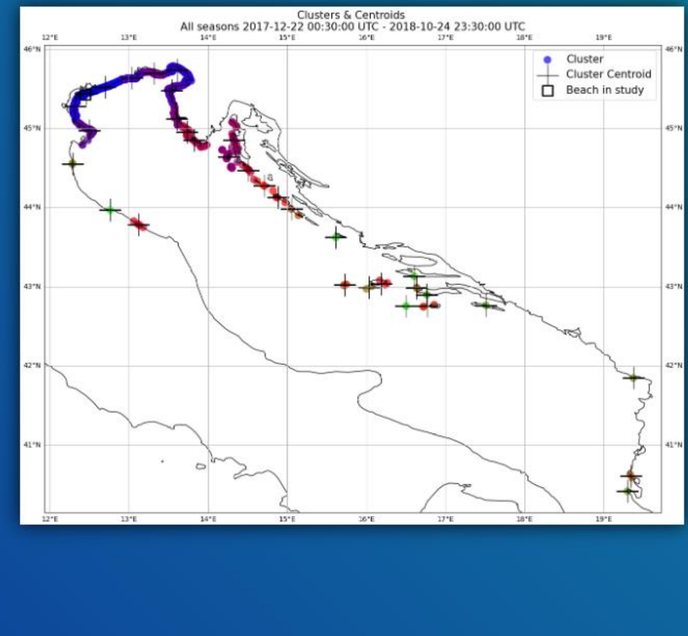
The polygons considered for matching the centroid clusters to the sources are shown below:

Here below an example of the analysis is shown, the results are of the Punta Sabbioni beach. The tables for each period of time show: the counts of beached elements found in that time period, the probability of the source to be the pollutant for that beach in study, and the partial % that indicates the probability not considering the source with more counts elements. For each time period the figures with the cluster and centroids are shown on the right. For example, considering the annual time period, the Venice lagoon source has about the 70% of probability to pollute the Punta Sabbioni beach.

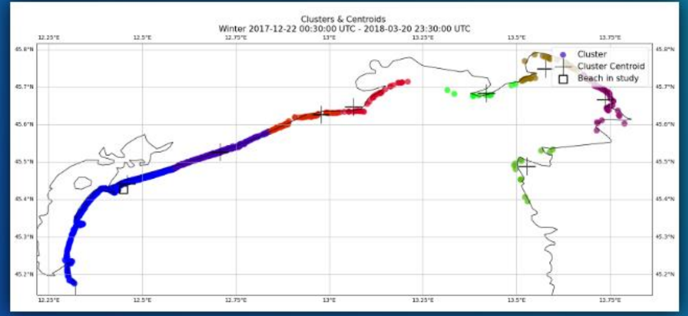
Interreg
Italy - Croatia
MARLESS
EUROPEAN UNION


45°25'36.2"N 12°26'08.3"E
Punta Sabbioni
Slovenia
Croazia
Bosnia ed
Erzegovina
San Marino
Italia
Montenegro
Mare Adriatico
Mar Tirreno

Annual (2017/12/22 - 2018/10/24)			
Possible sources from backtrajectories	Counts	%	Partial %
Venice lagoon	16338	68.893	
Piave river	3412	14.388	46.252
Brenta Adige rivers	2460	10.373	33.347
Tagliamento river	539	2.273	7.306
Isonzo river - Monfalcone harbour	468	1.973	6.344
Po river	176	0.742	2.386
Pirano-Isola-Koper harbours	89	0.375	1.206
Marano-Grado lagoon	74	0.312	1.003
Fiume harbour	65	0.274	0.881
Rovigno harbour	31	0.131	0.420
Pola harbour	24	0.101	0.325
Zara harbour and Zermagna river	10	0.042	0.136
Narenta river	9	0.038	0.122
Spalato harbour and Cettina-Jedro rivers	8	0.034	0.108
Cesano-Misa rivers and Senigallia harbour	5	0.021	0.068
Marina Ravenna harbour	2	0.008	0.027
Voiussa river	2	0.008	0.027
Foglia river	1	0.004	0.014
Valona harbour	1	0.004	0.014
Boiana river adn Port Milena harbour	1	0.004	0.014



Winter (2017/12/22 - 2018/03/20)			
Possible sources from backtrajectories	Counts	%	Partial %
Venice lagoon	5256	83.5	
Piave river	764	12.1	73.7
Tagliamento river	171	2.7	16.5
Trieste harbour	50	0.8	4.8
Isonzo river - Monfalcone harbour	28	0.4	2.7
Pirano-Isola-Koper harbours	12	0.2	1.2
Marano-Grado lagoon	11	0.2	1.1



Spring (2018/03/21 - 2018/06/21)			
Possible sources from backtrajectories	**Counts**	**%**	**Partial %**
Venice lagoon	4132	64.3	
Piave river	1317	20.5	57.4
Brenta Adige rivers	343	5.3	15.0
Tagliamento river	257	4.0	11.2
Isonzo river - Monfalcone harbour	248	3.9	10.8
Pirano-Isola-Koper harbours	42	0.7	1.8
Marano-Grado lagoon	39	0.6	1.7
Po river	38	0.6	1.7
Rovigno harbour	10	0.2	0.4

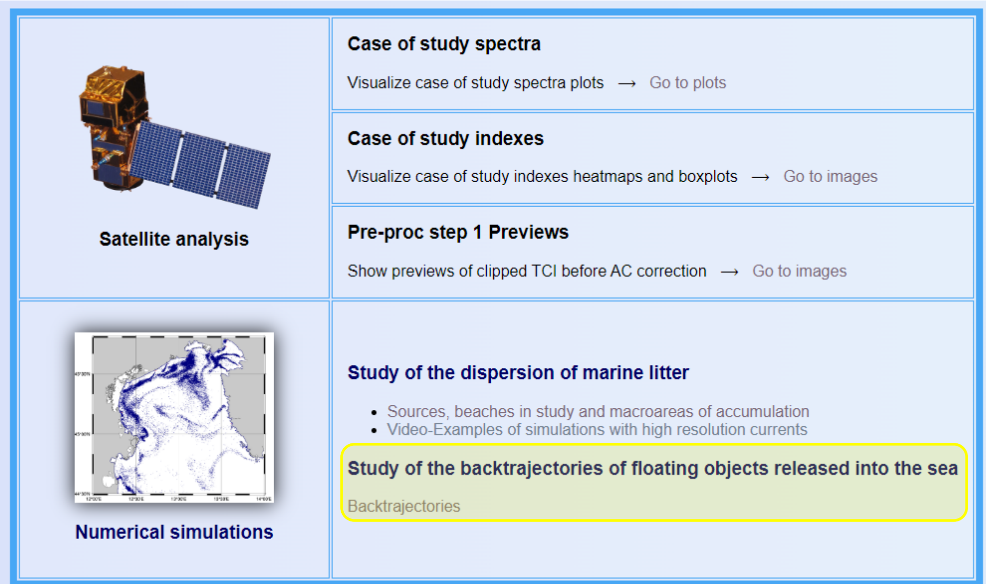
Summer (2018/96/22 - 2018/09/22)			
Possible sources from backtrajectories	**Counts**	**%**	**Partial %**
Venice lagoon	4104	65.40	
Brenta Adige rivers	1360	21.67	62.64
Piave river	560	8.92	25.79
Po river	84	1.34	3.87
Tagliamento river	78	1.24	3.59
Isonzo river - Monfalcone harbour	57	0.91	2.63
Pirano-Isola-Koper harbours	28	0.45	1.29
Pola harbour	2	0.03	0.09
Rovigno harbour	1	0.02	0.05
Fiume harbour	1	0.02	0.05

Autumn (2018/09/23 - 2018/10/24)			
Possible sources from backtrajectories	**Counts**	**%**	**Partial %**
Venice lagoon	1808	80.97	
Piave river	380	17.02	89.41
Livenza river	44	1.97	10.35
Marano-Grado lagoon	1	0.04	0.24

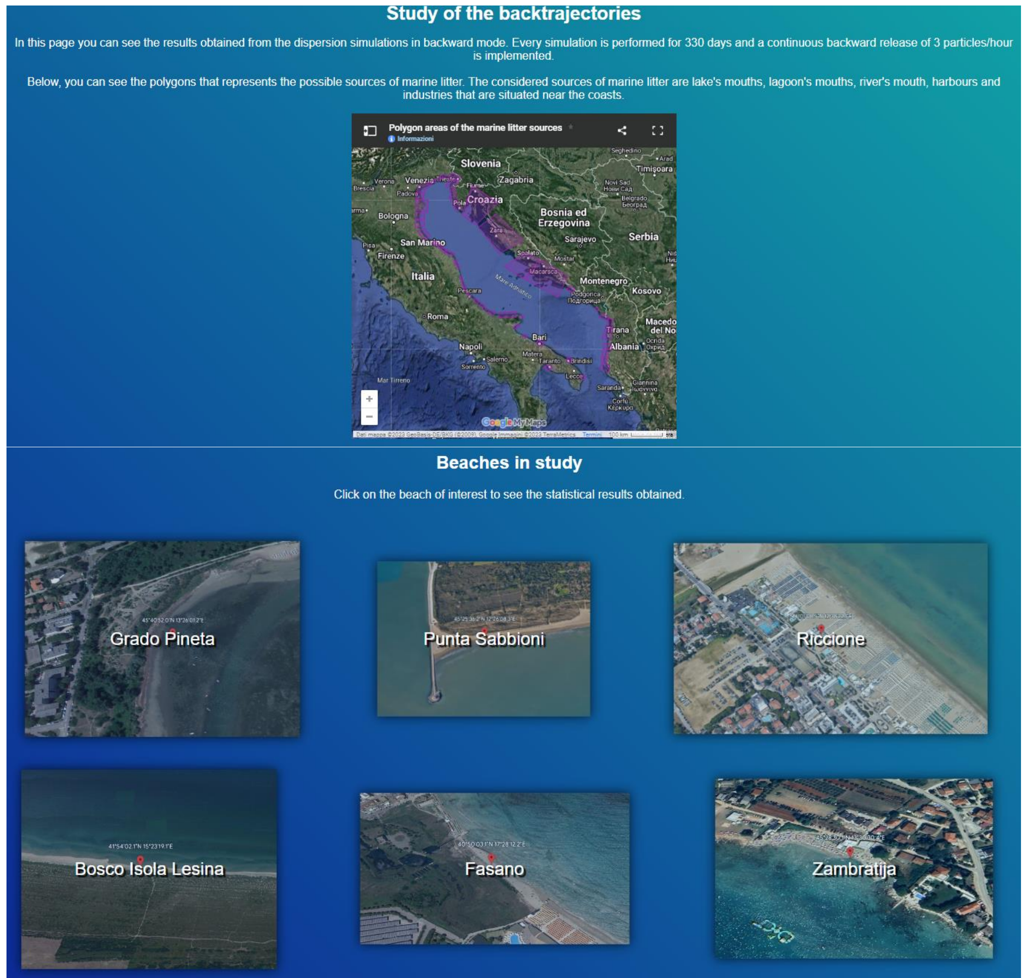
Conclusions

In conclusion, the main results are organized in digital form. In the working progress they are available in the following website: http://interreg.c3hpc.exact-lab.it/MARLESS/

The following image shows the homepage of the website; the part relative to this work is the 'Study of the backtrajectories of floating objects release into the sea' in the section relative to 'Numerical simulations'.

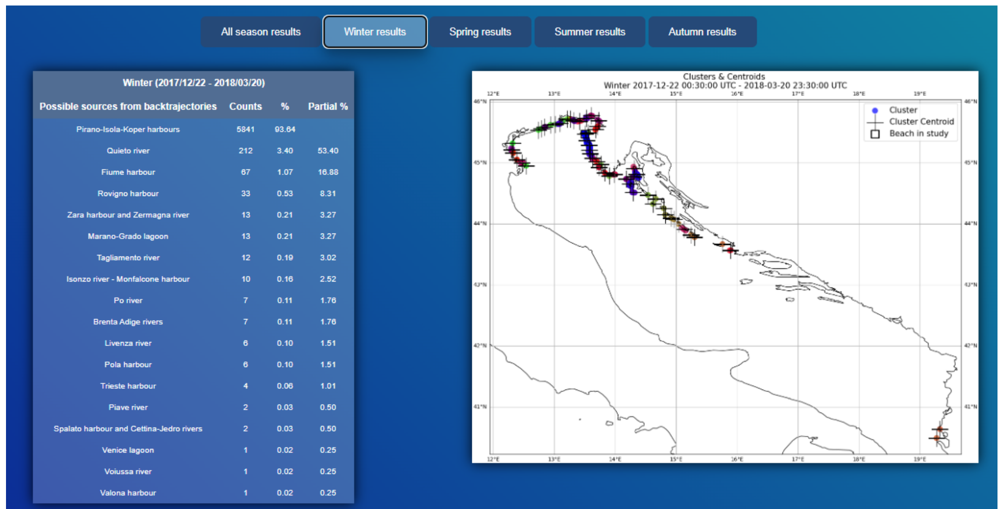


On the link 'Backtrajectories' you can find the maps with the polygons that represents the possible sources of marine litter and the list of beach studied.



Then, if you click on a beach you will see the results obtained:

Clicking on the time period of interest you can see the respective result obtained:



Winter (2017/12/22 - 2018/03/20)			
Possible sources from backtrajectories	Counts	%	Partial %
Pirano-Isola-Koper harbours	5841	93.64	
Quieto river	212	3.40	53.40
Fiume harbour	67	1.07	16.88
Rovigno harbour	33	0.53	8.31
Zara harbour and Zermagna river	13	0.21	3.27
Marano-Grado lagoon	13	0.21	3.27
Tagliamento river	12	0.19	3.02
Isonzo river - Monfalcone harbour	10	0.16	2.52
Po river	7	0.11	1.76
Brenta Adige rivers	7	0.11	1.76
Livenza river	6	0.10	1.51
Pola harbour	6	0.10	1.51
Trieste harbour	4	0.06	1.01
Piave river	2	0.03	0.50
Spalato harbour and Cettina-Jedro rivers	2	0.03	0.50
Venice lagoon	1	0.02	0.25
Voiussa river	1	0.02	0.25
Valona harbour	1	0.02	0.25

These html pages are also available in the attached folder *D3.3.4.tar*.

Appendix

Jupyther code for diffusion test

```
%matplotlib inline
from parcels import FieldSet, ParticleSet, Variable, JITParticle, AdvectionRK4, plotTrajectoriesFile, DiffusionUniformKh
import numpy as np
import math
from datetime import timedelta
from operator import attrgetter

fieldset = FieldSet.from_parcels("MovingEddies_data/moving_eddies")
kh_zonal = 10  #in m^2/s
kh_meridional = 10 #in m^2/s both values are converted to degrees/s under the hood since the mesh is spherical and not flat
```

```
fieldset.add_constant_field("Kh_zonal", kh_zonal)
fieldset.add_constant_field("Kh_meridional", kh_meridional)

pset = ParticleSet.from_list(fieldset=fieldset,  # the fields on which the particles are advected
                        pclass=JITParticle,  # the type of particles (JITParticle or ScipyParticle)
                        lon=[3.3e5,  3.3e5], # a vector of release longitudes
                        lat=[1e5, 2.8e5])    # a vector of release latitudes


output_file = pset.ParticleFile(name="EddyParticles.nc", outputdt=timedelta(hours=1)) # the file name and the time step of the outputs
pset.execute(pset.Kernel(AdvectionRK4)  + pset.Kernel( DiffusionUniformKh),             # the kernel (which defines how particles move)
          runtime=timedelta(days=6),    # the total length of the run
          dt=timedelta(minutes=5),      # the timestep of the kernel
          output_file=output_file)



rsync -vlch  -e "ssh -i C:\Users\705138\Documents\MobaXterm_Portable_v20.6\Chiavi/
Marless\id_rsa_C3HPC_farrisc" prova_prossimità_adriatico.nc  farrisc@login.c3hpc.exact-lab.it:/lustre/arpa/scratch/MARLESS/
```

References

1. [Online]. Available: https://nbviewer.jupyter.org/github/OceanParcels/parcels/blob/master/parcels/examples/tutorial_parcels_structure.ipynb.
2. [Online]. Available: https://nbviewer.jupyter.org/github/OceanParcels/parcels/blob/master/parcels/examples/documentation_indexing.ipynb.
3. P. Delandmeter e E. van Sebille, *The Parcels v2.0 Lagrangian framework: new field interpolation schemes,* Geoscientific Model Development, 2019.
4. K. Döös, B. Jönsson e J. Kjellss, *Evaluation of oceanic and atmospheric trajectory schemes in the TRACMASS trajectory model v6.0,* Geoscientific Model Development, 2016.
5. [Online]. Available: https://nbviewer.jupyter.org/github/OceanParcels/parcels/blob/master/parcels/examples/tutorial_analyticaladvection.ipynb.
6. E. van Sebille e et al., *Lagrangian ocean analysis: Fundamentals and practices,* vol. 121, Elsevier, A cura di, Ocean Modelling, 2018, pp. 49-75.
7. [Online]. Available: https://nbviewer.jupyter.org/github/OceanParcels/parcels/blob/master/parcels/examples/tutorial_diffusion.ipynb.
8. N. Scheijen, *Plastic litter in the ocean. Modeling of the vertical transport of micro plastics in the ocean.*
9. V. Onink e et al., *Global simulations of marine plastic transport show plastic,* vol. 16, Environmental Research Letter, 2021.