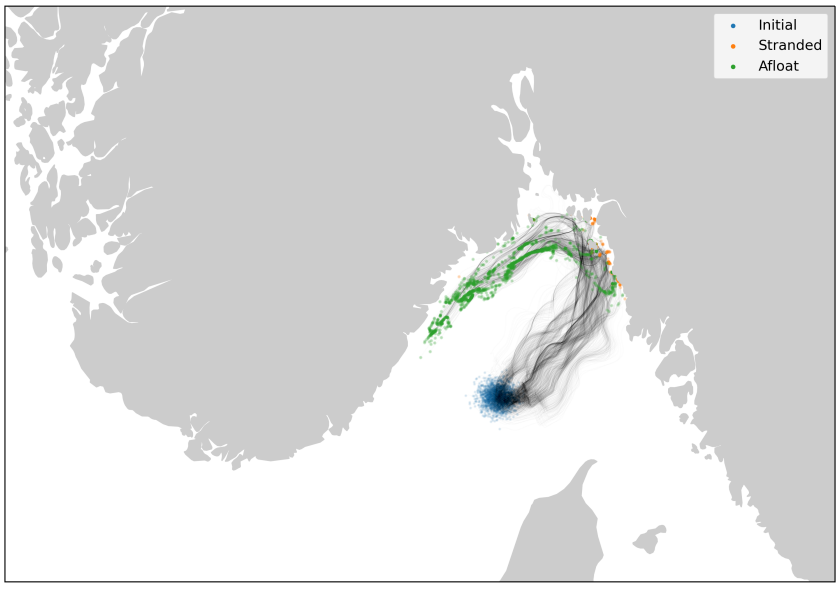
NTNU – Trondheim
Norwegian University of
Science and Technology

NTNU, Institutt for Fysikk

TMA4320 spring 2024 - Technical physics project

Tor Nordam

Particle transport with ocean currents



Project period: 03.04.24 - 16.04.25

Praktisk informasjon

Innleveringsfrist: Tirsdag 16.04.24, kl 23.59
Innleveringsplattform: Inspera (Se wikisiden til faget for mer info)
Språk: Dere kan velge om dere vil svare på engelsk eller norsk
Vurderingsansvarlig: Niels Henrik Aase, mail: `niels.h.aase@ntnu.no`
Innleveringsformat: Jupyter Notebook.
NB! Notebooken skal være kjørt ved innlevering. For å sjekke om den er kjørt kan dere laste ned den endelige notebooken dere skal levere og sjekke at figurene fortsatt er der.

1 Introduction

It is probably well known that the Norwegian Meteorological Institute produces a weather forecast, predicting the weather several days into the future. This weather forecast includes a vector field of air velocities at 10 m elevation above the ground, commonly known as "the wind"[1]. Less well known is that they also produce a forecast for the ocean, which includes a velocity field describing the ocean currents, as well as for example the temperature and salinity. In the event of an accident at sea, this information can be used to predict where spilled materials such as oil, chemicals or plastics will end up, which in turn can be used to direct response operations to try to minimise the damage. During for example the Deepwater Horizon oil spill in the Gulf of Mexico in 2010, numerical simulations were used daily to predict what would happen over the next few days.

In this project, we will study the transport of floating plastics particles at the surface of the ocean using a so-called Lagrangian particle method. You will see how to write programs that can read data which are provided by the Meteorological Institute, and how to interpolate these data and use them to calculate trajectories. You will also learn to plot positions and trajectories on a map.

2 Background and Theory

The point of this exercise is to look at how we can calculate the transport of plastics in the ocean. In our simulations, we will represent the plastics as numerical particles, also called "Lagrangian elements". The idea is that a numerical particle will represent a given mass of physical plastics particles. The numerical particles are not to be interpreted as one actual, physical particle, but simply as a numerical representation. If we have large numbers of numerical particles, they can be used to calculate things like probability of finding plastics at the beach in some location.

The particles will move due to the velocity of the water and the wind. We assume that the presence of the particles does not affect the motion of the ocean or the atmosphere, so we can take the velocities as given, and simply look up the current and the wind at some point, and use that in our transport equation.

[1] See https://www.yr.no/en/map/wind/.

As our transport model, we will simply assume that the particle moves with the velocity of the water, plus a contribution from the wind. If the position of a numerical particle is $\mathbf{x}$, then the transport model is described by the ODE

$$\dot{\mathbf{x}} = \mathbf{v}_c(\mathbf{x}, t) + f_w \mathbf{v}_w(\mathbf{x}, t), \tag{1}$$

where $\mathbf{v}_c(\mathbf{x}, t)$ is the velocity of the current, as a function of position and time, and $\mathbf{v}_w(\mathbf{x}, t)$ is the velocity of the wind, also a function of position and time. We will limit ourselves to looking at floating particles, that only move horizontally and stay at the ocean surface.

The factor f_w in Eq. (1) is called the windage factor, and depends on the type of plastics to be modelled. Generally, a large piece of plastics with low density, such as an empty bottle, will be more affected by the wind than a denser piece of plastics that is almost completely submerged.

2.1 Environmental data

In this project, we will use pre-calculated ocean current and wind data to tell us the velocity of the current, $\mathbf{v}_c$, and the wind, $\mathbf{v}_w$ as a function of $\mathbf{x}$ and t. The data are produced by the Meteorological Institute, and they are the results of numerical simulations that are run daily. It provides information about current velocities, water temperature, salinity and a few other parameters for an area covering the entire Norwegian coast (see Fig. 1), at 800 m $\times$ 800 m horizontal resolution and with a temporal resolution of 1 hour.

The data are available for download as NetCDF[2] files. NetCDF is a file format for storing data in arrays. It is quite well suited for medium to large amounts of data (up to 100s of gigabytes in a single file works fine), and it is very commonly used for geoscientific data such as ocean or atmosphere data. In order to access the data, we will use the python library `xarray`.

The data files contain the x and y components of the velocity field, stored in two variables named `u` and `v` (it is quite common to store only the horizontal components of the current velocity, as the vertical component is typically much smaller). These variables are stored as rank 4 arrays, which give their values as a function of time, depth, y position and x position (note that the order of the dimensions in the files is (t, z, y, x)). The coordinates of the grid points along these dimensions are also stored in the data files, in the variables `time`, `depth`, `Y`, and `X`. For simplicity, we will ignore the depth dimension, dealing only with movement in the horizontal plane at the surface of the ocean. In Fig. 1, temperature data for the surface water is shown, as an example to illustrate the extent of the available data. In Fig. 2, the same data are shown in the coordinate system used in the file, with the x coordinates on the horizontal axis, and the y coordinates on the vertical, and distances in meters.

The dimensions x and y, as shown in Fig. 2, are the coordinate axes in what is known as a polar stereographic projection of the Earth's surface onto a plane. For the purpose of this project, we will deal with motion in the xy plane, with coordinates as shown. As

[2]Network Common Data Form, see www.unidata.ucar.edu/software/netcdf/.

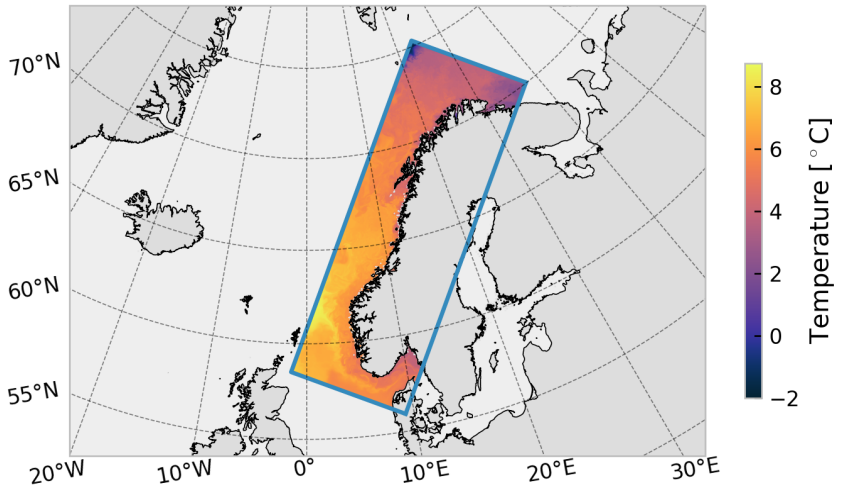


Figure 1: The domain of the NorKyst800m model, showing surface water temperatures on March 18, 2024.

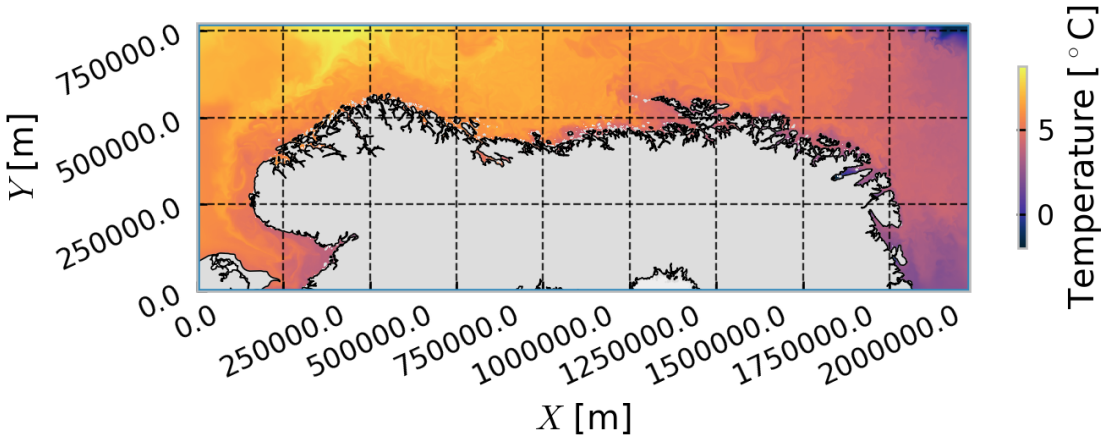


Figure 2: The domain of the NorKyst800m model, shown in the coordinate system used to store the data. The distances are in meters.

the vector components of the velocity field are aligned with these coordinate axes, we can use the components directly to calculate motion in the xy plane. This means that for the transport simulations, we will ignore the curvature of the Earth. In the end, we will see how to transform from xy coordinates to longitude and latitude, and plot the particle positions on a map.

2.2 Interpolation

The oceanographic data we will use here are provided on a grid of discrete positions and times, (t_n, z_l, y_j, x_i). As mentioned, we will consider motion in the xy plane only, *i.e.*, we will consider a constant depth $z = 0$. The available data then represents a time-variable

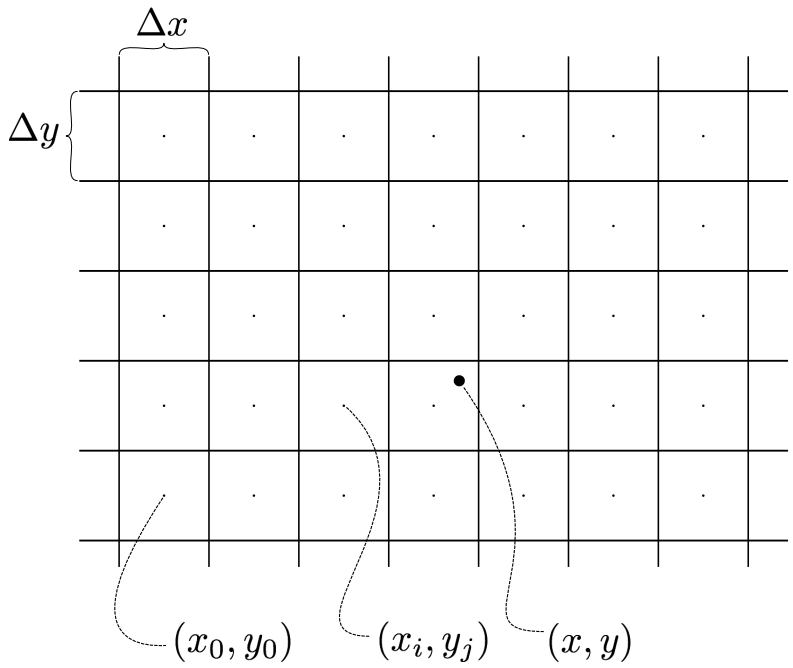


Figure 3: Velocity vector components are given only on discrete points, $(x_i, y_j) = (x_0 + i\Delta x, y_0 + j\Delta y)$. In order to calculate trajectories, we need to evaluate the velocity at arbitrary positions (x, y).

2D vector field of two-component vectors. The vectors of this field are given on a regular quadratic grid, with cells of size 800 m × 800 m. We say that the spatial resolution of the data is 800 m. Additionally, there is a temporal resolution: the data are provided at one-hourly intervals.

For each cell in the grid, and for each interval of one hour, there is one vector. The vector at time and location (t_n, y_j, x_i) gives the average velocity of water in an 800 m × 800 m cell centered at (x_i, y_j), and in a time interval of length 1 hour, centered at time t_n.

In order to calculate the trajectory of a particle that moves in the velocity field defined by these data, we will have to evaluate the vector field at arbitrary locations (see Fig. 3). To evaluate the velocity at a location (x, y), one possible option is to identify which cell that location is in, and use the vector defined at the center of that cell. This is sometimes known as nearest-neighbour interpolation. The advantages of this approach include simple implementation, and the fact that it makes a certain intuitive sense, as the vector is supposed to define the average velocity inside a cell. The major disadvantage is that the vector field is then discontinuous at the boundaries of cells, which is not realistic.

A more advanced approach is the technique known as spline interpolation, which produces piecewise polynomial results. The idea of spline interpolation is to ensure that not only the interpolated variable is continuous, but also its first $k-1$ derivatives, where k is the degree of the spline interpolation. An example of spline interpolation in one

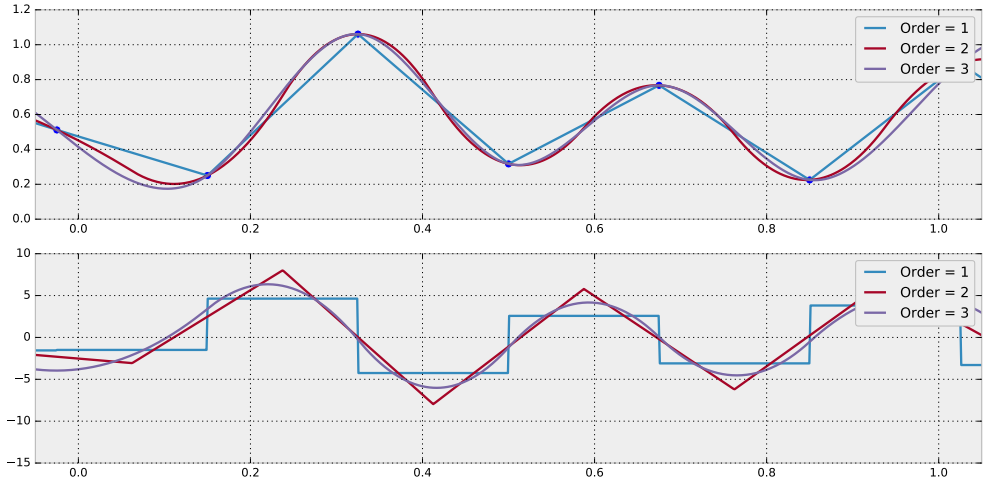


Figure 4: Example of one-dimensional spline interpolation (top) with derivatives (bottom).

dimension is shown in Fig. 4. We have some given data points, marked with blue dots, and the goal is to find a function that passes through all of these points. First order spline interpolation is equivalent to simple linear interpolation (the blue line in the top panel). In this case, the interpolated values are continuous, but the derivative (the blue line in the bottom panel) will change discontinuously at each datapoint. For second order splines, the function is now continuous and smooth, the derivative is also continuous, but piecewise linear, which means the second derivative will be discontinuous. For third order splines, also known as cubic splines, both the function and the derivative are continuous and smooth. The same ideas also extend to two and more dimensions.

In this project, we will use the class called `RectBivariateSpline` from the library `scipy.interpolate` to carry out the interpolation to arbitrary positions in the xy plane. The procedure is as follows:

- For each timestep, get the rank 2 arrays holding the vector components of the wind and the current for the surface layer ($z = 0$), for that time.
- For each of the four vector component arrays (two for wind, two for currents), pass them along with the coordinate arrays `Y` and `X` to `RectBivariateSpline`. This returns an interpolation object for each component, which can be used to obtain the interpolated value of the vector component at any point.
- Use these to calculate the motion of the particles, step forward in time, and repeat.

The steps above are implemented in the `Interpolator` class in the provided example code.

2.3 Integration

In this project, we will be using a numerical ODE integrator called Heun's method. This is a secord-order explicit method, where the position at time $t_{i+1} = t_i + h$ is given by

$$\begin{aligned} \mathbf{k}_1 &= f(\mathbf{x}_i, t_i) \\ \mathbf{k}_2 &= f(\mathbf{x}_i + \mathbf{k}_1 h, t_i + h) \\ \mathbf{x}_{i+1} &= \mathbf{x}_i + \frac{h}{2}(\mathbf{k}_1 + \mathbf{k}_2). \end{aligned} \tag{2}$$

An advantage of this method is that if we only know $f(\mathbf{x}, t)$ at discrete times, t_n, we can choose h to match such that we don't have to worry about interpolation in time. This will be useful in Problems 2 and 3, when we are using pre-computed velocity field which are available only at hourly intervals.

3 Plotting on maps with python

When moving particles around with the water velocity, we will use the coordinate system shown in Fig. 2. For plotting trajectories, it is straightforward to just use those coordinates directly, which will show distances in meters. However, if we want to show the trajectories on a map, we need to convert from the xy coordinate system of the polar stereographic projection, to longitude and latitude. In this project, we will use the library `cartopy`, which has features for dealing with map projections and plotting map features such as coastlines.

To use the provided example code, you will need to install these libraries:

- `numpy`
- `scipy`
- `xarray`
- `netCDF4`
- `cartopy`

4 Further reading

If you are interested in reading or seeing more about how particle transport methods like the ones described here are used in environmental research, some suggestions for further reading could be to look at OceanParcels or OpenDrift. They are two widely used open-source libraries for marine transport modelling, suitable for many applications.

- https://oceanparcels.org/
- https://opendrift.github.io/

There is also a quite good review paper by van Sebille et al.: https://doi.org/10.1016/j.ocemod.2017.11.008

5 Problems

The exercises follow below. For your convenience, you may refer directly to specific questions (such as e.g. **1a**) in your text, but still try to write a coherent text. **In particular, include** an introduction of the project which does not overly rely on the project description. Also write a conclusion for the project. In the rest of your Notebook, you can assume that the reader has read the project description.

As in project 1, we recommend structuring the notebook by integrating code, results, and discussion in a seamless manner. All plots should be discussed, and they must have physical units when necessary. Your code should be well-documented and structured.

Problem 1 - Simple test case

In this problem, we will start simple. Instead of using actual wind and current data, we will use an analytical expression to represent a velocity field, and consider particles moving in this field. The position of a particle, $\mathbf{x} = [x, y]$, is governed by the ODE

$$\begin{aligned} \dot{\mathbf{x}} &= \mathbf{f}(\mathbf{x}, t) \\ &= [v_x(\mathbf{x}, t), v_y(\mathbf{x}, t)]. \end{aligned} \tag{3}$$

This velocity field is defined in the xy-plane, in the region $x \in [0, 2]$, $y \in [0, 1]$, and the velocity components are given by:

$$\begin{aligned} v_x &= -\pi A \sin(\pi f(x,t)) \cos(\pi y), \\ v_y &= \pi A \cos(\pi f(x,t)) \sin(\pi y) \frac{\partial f(x,t)}{\partial x}, \end{aligned} \tag{4a}$$

where

$$\begin{aligned} f(x,t) &= a(t)x^2 + b(t)x, \\ a(t) &= \epsilon \sin(\omega t), \\ b(t) &= 1 - 2\epsilon \sin(\omega t), \end{aligned} \tag{4b}$$

where the parameters A, ϵ and ω are chosen to adjust the properties of the system. Here, we will use the values $A = 0.1$, $\epsilon = 0.25$ and $\omega = 1$.

a. We will first consider a single particle starting from an initial position $\mathbf{x}_0 = [1.05, 0.5]$. Let the trajectory, $\mathbf{x}(t)$, of the particle be controlled by Eqs. (3) and (4). Calculate the trajectory for the time interval $t \in [0, 50]$, using Heun's method.

   Try a few different timesteps, and compare the results by plotting the trajectories for different timesteps in the same plot. Make up your mind about what seems a reasonably short timestep. What happens if you double the integration time, such that $t \in [0, 100]$, is the same timestep still a good choice?

b. We will next calculate many trajectories at once, where the trajectories represent individual particles. Choose a grid of $N_p = 100$ initial conditions, for example spanning a square of size 0.1×0.1. Calculate the trajectory for each particle (i.e., for each initial condition), for the interval $t \in [0, 10]$, using Heun's method. In the same plot, show scatter plots of the initial and final positions, and plot the trajectories that connect them. You might want to use for example

```
plt.plot(x, y, alpha=0.1, lw=0.1)
```

to get thin and transparent lines, otherwise the plot can look very messy.

c. You should make sure your implementation is reasonably efficient. A key point is to avoid looping over the particles, but rather writing your implementation of Heun's method such that it operates on an array of particle positions. Test your implementation by measuring the time it takes to run for different numbers of particles, N_p. Make a plot of run time as a function of N_p. An example of such a plot is shown in Fig. 5. Why do you think the run time is approximately constant for small N_p?

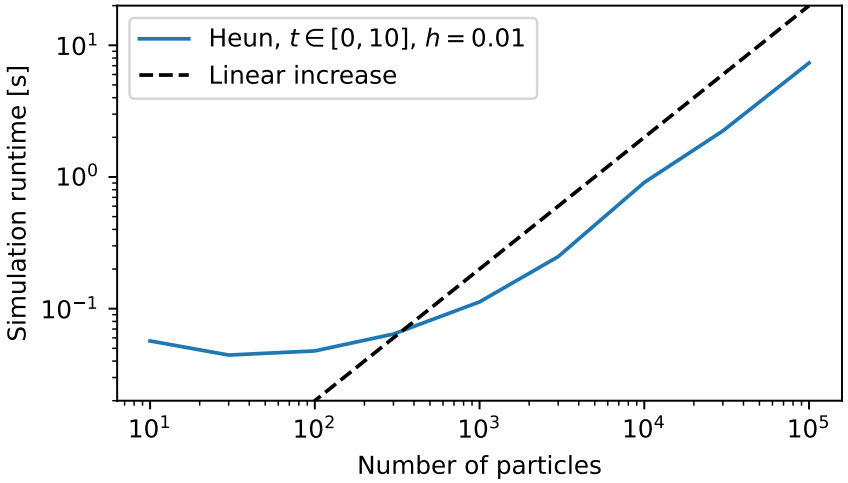


Figure 5: Simulation runtime as a function of particle number, for integration time $t \in [0, 10]$ and timestep $h = 0.01$.

2 - Transport of a particle with ocean currents

We will now turn to using model data of winds and ocean currents to calculate particle trajectories, as described in Section 2. The particle position is governed by the ODE

$$\dot{\mathbf{x}} = \mathbf{v}_c(\mathbf{x}, t) + f_w \mathbf{v}_w(\mathbf{x}, t), \tag{5}$$

where $\mathbf{v}_c(\mathbf{x}, t)$ is the velocity of the current, as a function of position and time, and $\mathbf{v}_w(\mathbf{x}, t)$ is the velocity of the wind, also a function of position and time. The factor f_w we called the windage factor (see Section 2). Use $f_w = 0.03$ unless otherwise specified.

We have prepared a datafile containing 6 days of wind and current data spanning from March 20 to March 25 2024. In the datafile, the x and y dimensions are in units meters, and the coordinate axes are the same as in Fig. 2. The time dimension is measured in seconds from the start of the file (00:00, March 20, 2024).

The datafile (size 1.3 GB) is available for download here:

- `http://folk.ntnu.no/nordam/data/NorKyst-800m.nc`

In this problem you will also use Heun's method, with a timestep $h = 3600$ s. Since this integrator evaluates the velocity field only at integer multiples of h, and since the data are provided at intervals of 1 hour, no interpolation in time is needed when we choose $h = 3600$ s, and $t_0 = 0$ (or some other multiple of 3600). In the provided examples, you will find a class called `Interpolator`. This class contains an implementation of $f(\mathbf{x}, t)$ as defined by Eq. (1), including the code to read and interpolate data from the files. See the examples for how to create and use instances of this class.

a. Create a set of initial conditions by choosing N_p Gaussian random x and y values, using mean values $\bar{x} = 790\,000$ m, $\bar{y} = 490\,000$ m, and standard deviation 10 000 m (same for x and y). This corresponds to a position somewhere to the west of Trondheim (see Fig. 2). Choose a suitable value of N_p, so that the time to run (and plot) isn't too long. Transport the particles with the current and wind for a time of 5 days, using $t_0 = 0$. Plot the trajectories directly in x-y-coordinates.

b. Use the attached example code to plot your trajectories on a map (see also comment about plotting in problem 1b). Using Figs. 1 and 2 as a guide, try selecting a few other initial positions [3], and repeat the simulation.

3 - Stranding of plastics

In Problem 2, we only considered plastics drifting at sea. However, we are often interested in finding out where the plastics will end up on the shoreline. The provided `Interpolator` class contains a function called `on_land()`, which returns True for those particles that are on land. Use this to modify the behaviour of the particles such that they stop if they reach land.

a. Create a set of initial conditions by choosing N_p Gaussian random x and y values, using mean values $\bar{x} = 250\,000$ m, $\bar{y} = 460\,000$ m, and standard deviation 10 000 m (same for x and y). Use a suitable value for N_p and justify your choice. Run the particle simulation for a time of 3 days. Make a plot that shows the distribution of stranded plastics particles, for example using a scatter plot to show final positions, with stranded particles in a different color than those that are still at sea. Also

[3] You could for example choose a position near your hometown.

include a plot showing the percentage of all particles that are stranded as a function of time.

b. With the same initial conditions as above, try a few different windage factors, in the range from 0 to 0.15, and show how this affects the results. You can for example look into the percentage of particles that become stranded after three days as a function of f_w. Note that the results may of course depend on where (and when) you start the particle trajectories, and for some initial conditions no particles will reach land at all.

If you want to run simulations for different time periods than covered by the provided data from March 2024, it is possible to run with data directly from the server at MET Norway without downloading the files first. Though be aware that it can be slower to run simulations in this way. To try this, you can create an `xarray` dataset like this:

```
d = xr.open_mfdataset([
    'https://thredds.met.no/thredds/dodsC/fou-hi/norkyst800m-1h/NorKyst-800m_ZDEPTHS_his.an.2024010100.nc',
    'https://thredds.met.no/thredds/dodsC/fou-hi/norkyst800m-1h/NorKyst-800m_ZDEPTHS_his.an.2024010200.nc',
    'https://thredds.met.no/thredds/dodsC/fou-hi/norkyst800m-1h/NorKyst-800m_ZDEPTHS_his.an.2024010300.nc',
    ])
```

and then pass that dataset when creating the Interpolator object.

Each of the urls in the example list points to a NetCDF file with 24 hours of data. Change the dates at the end of the urls to match the dates you want to look at, and add more files if needed. This particular set of data files goes back to 2019, but some files are missing. See here for a list of files: https://thredds.met.no/thredds/catalog/fou-hi/norkyst800m-1h/catalog.html