

Hands on *plasticparcels*

A ready-to-use tool for plastic dispersion simulations and parameterisation development

Michael Denes – postdoc in the *parcels* group

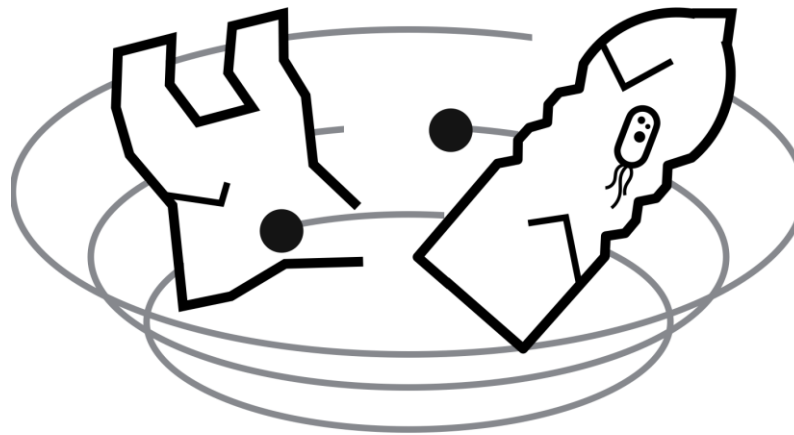
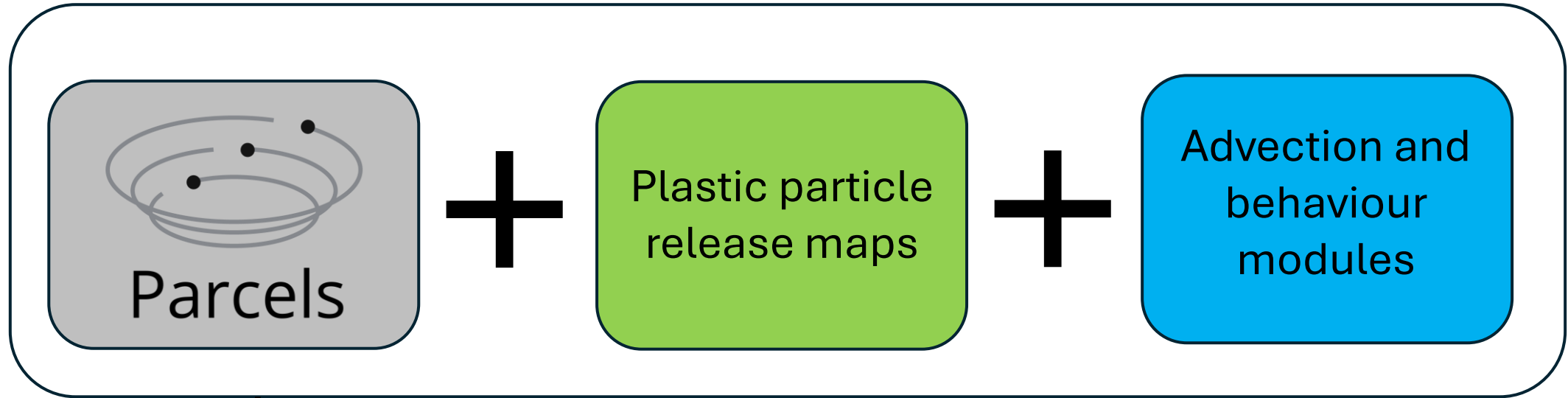
15-17 June, Santiago de Compostela



**Utrecht
University**



This project has received funding from Horizon Europe RIA under **Grant Number 101081273**



Plastic**Parcels**

<https://plastic.parcels-code.org/>

Summary

- A brief overview on numerical modelling and the *parcels** framework
 - Numerical modelling
 - Parcels
 - Fieldsets and fields (data)
 - Particlesets, particles, and variables
 - Kernels
- An overview and examples using *plasticparcels*
 - Release maps
 - Included kernels
 - Example using *copernicusmarine* toolkit

*v3 *parcels*

Numerical modelling of trajectories

The equation we are trying to integrate:

$$\frac{d\mathbf{x}}{dt} = \mathbf{v}_{\text{particle}}$$

Numerical modelling of trajectories

The equation we are trying to integrate:

$$\frac{d\mathbf{x}}{dt} = \mathbf{v}(\mathbf{x}(t), t) + \mathbf{p}(\mathbf{x}(t), t) + \mathbf{b}(\mathbf{x}(t), t)$$

Water velocity	Particle-dependent physics	Particle-dependent behaviour
	- Buoyancy forces - Wind-drag - “missing”/unresolved physics from ocean models	- Biofouling - Swimming - Diel vertical migration

With initial condition $\mathbf{x}(0) = \mathbf{x}_0$

Two **very** important things to consider:

- a) The ***integration scheme*** you choose (EE, RK2, RK4, etc.)
- b) How you ***interpret*** and ***interpolate*** your data

Numerical modelling of trajectories

The equation we are trying to integrate:

$$\frac{d\mathbf{x}}{dt} = \mathbf{v}(\mathbf{x}(t), t) + \mathbf{p}(\mathbf{x}(t), t) + \mathbf{b}(\mathbf{x}(t), t)$$

Water velocity	Particle-dependent physics	Particle-dependent behaviour
	- Buoyancy forces - Wind-drag - “missing”/unresolved physics from ocean models	- Biofouling - Swimming - Diel vertical migration

With initial condition $\mathbf{x}(0) = \mathbf{x}_0$

Explicit-Euler formation

$$\frac{d\mathbf{x}}{dt} = \mathbf{v}(\mathbf{x}(t), t)$$

Explicit-Euler formation

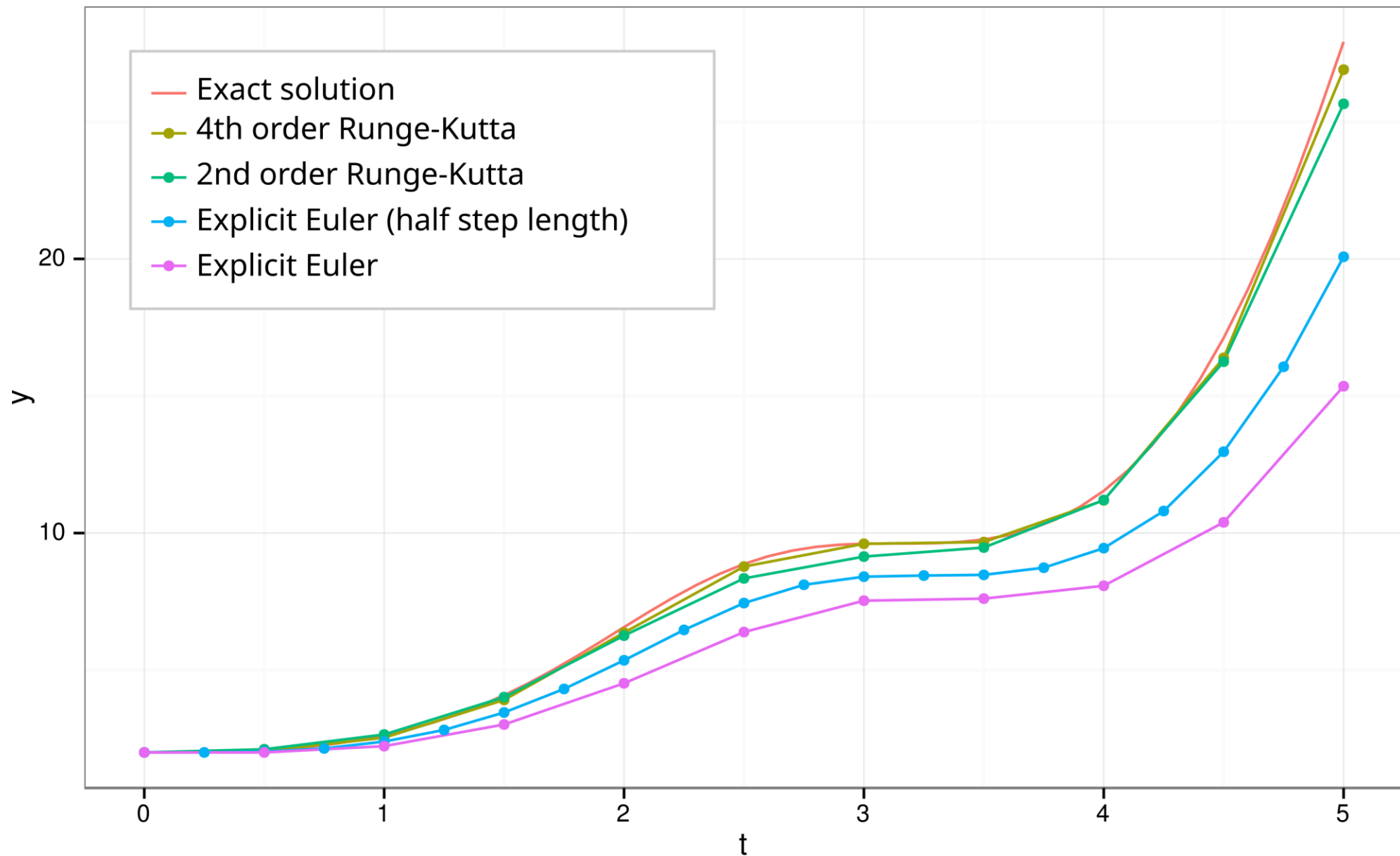
$$\frac{d\mathbf{x}}{dt} = \mathbf{v}(\mathbf{x}(t), t) \xrightarrow{\text{Discretising}} \frac{\Delta \mathbf{x}}{\Delta t} = \mathbf{v}(\mathbf{x}(t), t)$$

Explicit-Euler formation

$$\frac{d\mathbf{x}}{dt} = \mathbf{v}(\mathbf{x}(t), t) \xrightarrow{\text{Discretising}} \frac{\Delta \mathbf{x}}{\Delta t} = \mathbf{v}(\mathbf{x}(t), t)$$

$$\Delta \mathbf{x} = \mathbf{v}(\mathbf{x}(t), t) \Delta t$$

Displacement (change in position) = velocity $\times$ timestep



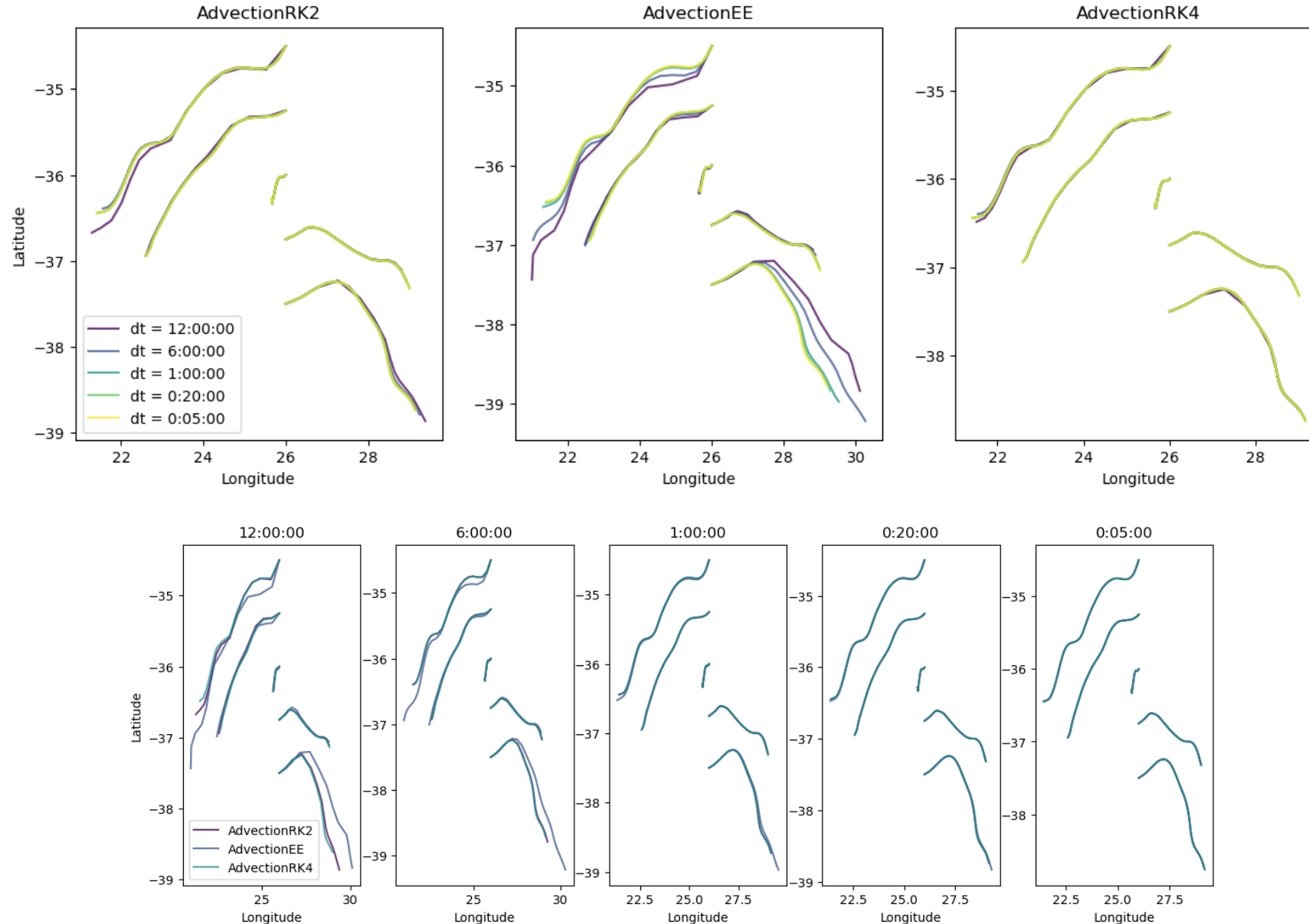
Note: Choice of integration scheme
and timestep will impact your results

From the Runge-Kutta Wikipedia page

The impact of the integration timestep

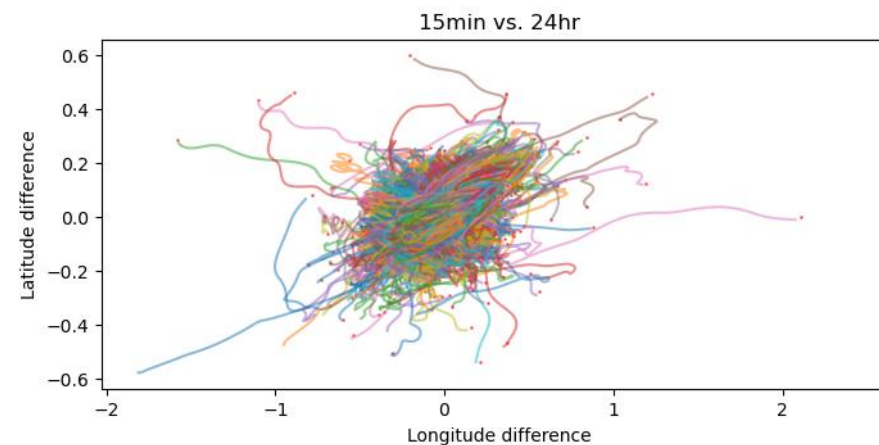
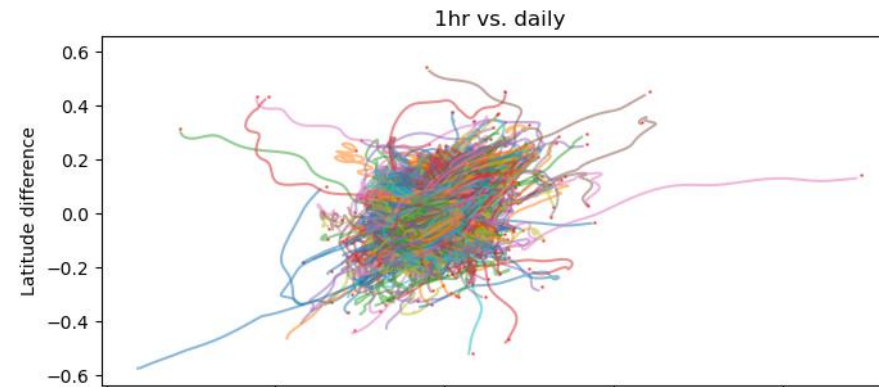
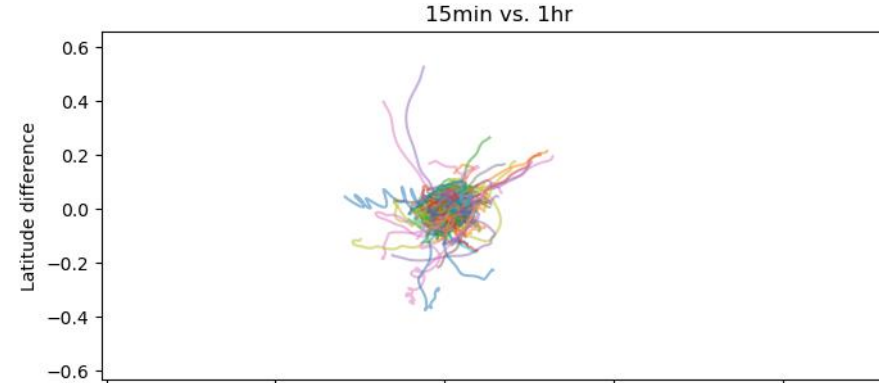
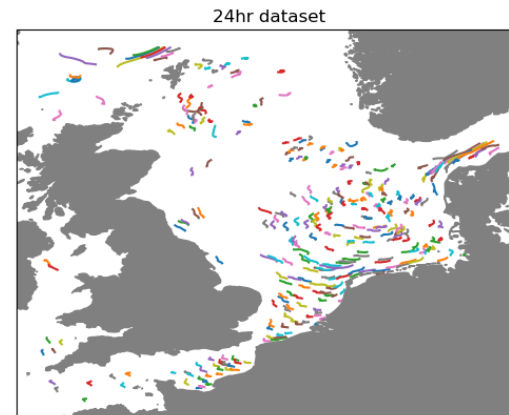
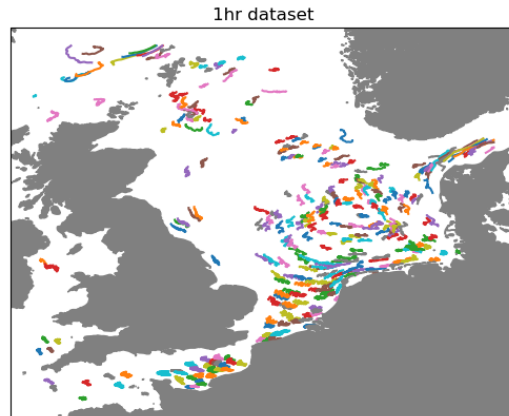
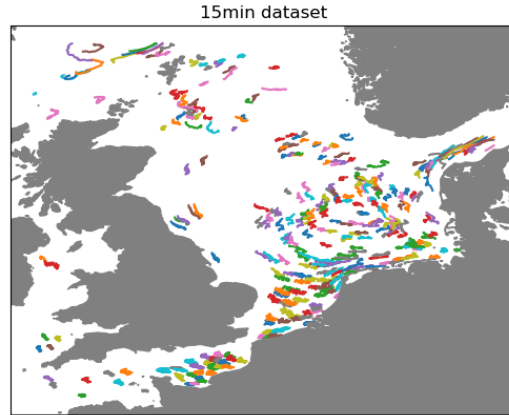
```
# Simulation settings
runtime = timedelta(days=7)
dt = timedelta(hours=12)
outputdt = dt

# Total simulation time
# Advection timestep
# Output timestep
```



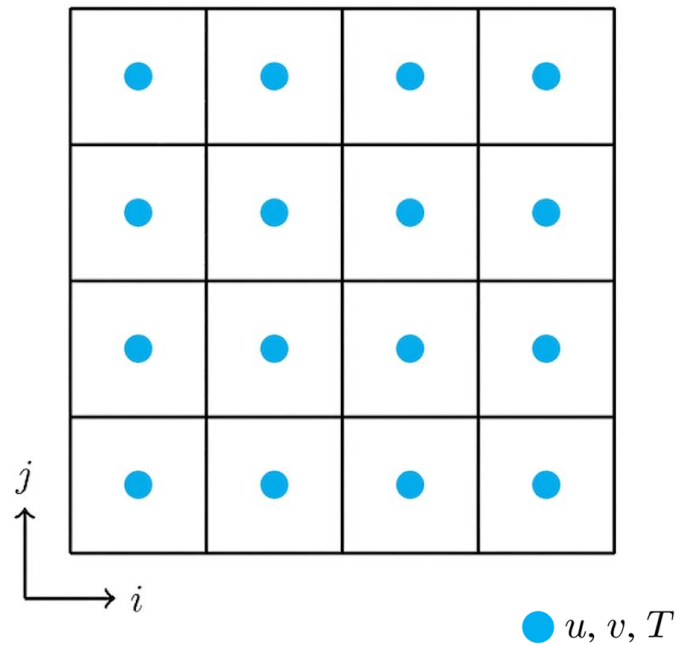
The impact of model temporal resolution

```
'startdate': datetime.strptime('2025-11-21-00:00:00', '%Y-%m-%d-%H:%M:%S')
'runtime': timedelta(days=5), # Runtime of simulation
'outputdt': timedelta(hours=1), # Timestep of output
'dt': timedelta(minutes=1), # Timestep of advection
```

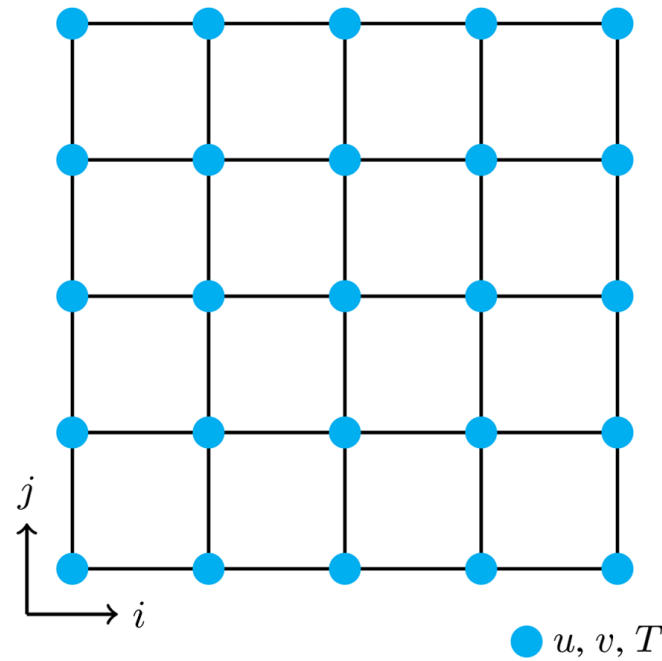


Data, grids, interpretation and interpolation scheme

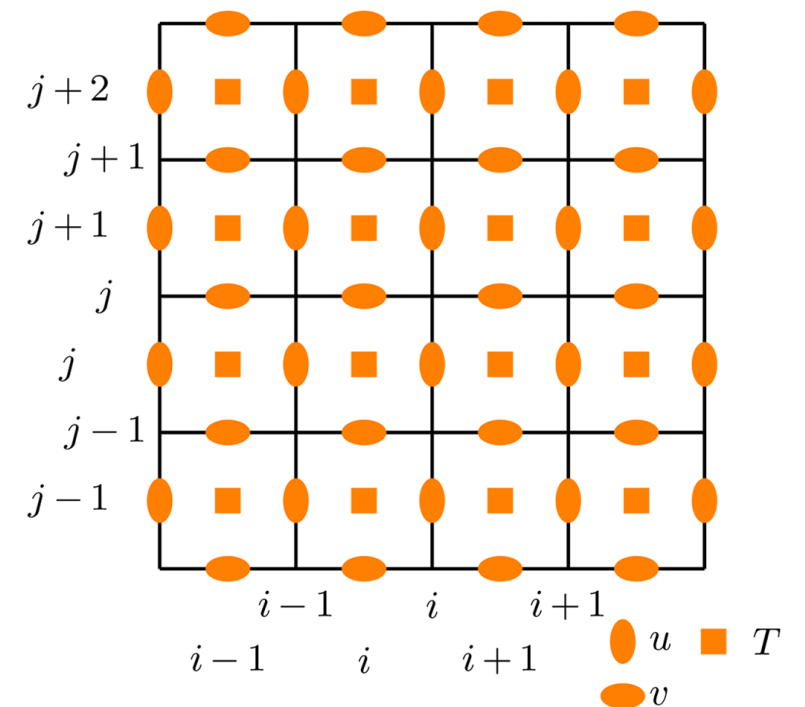
A grid



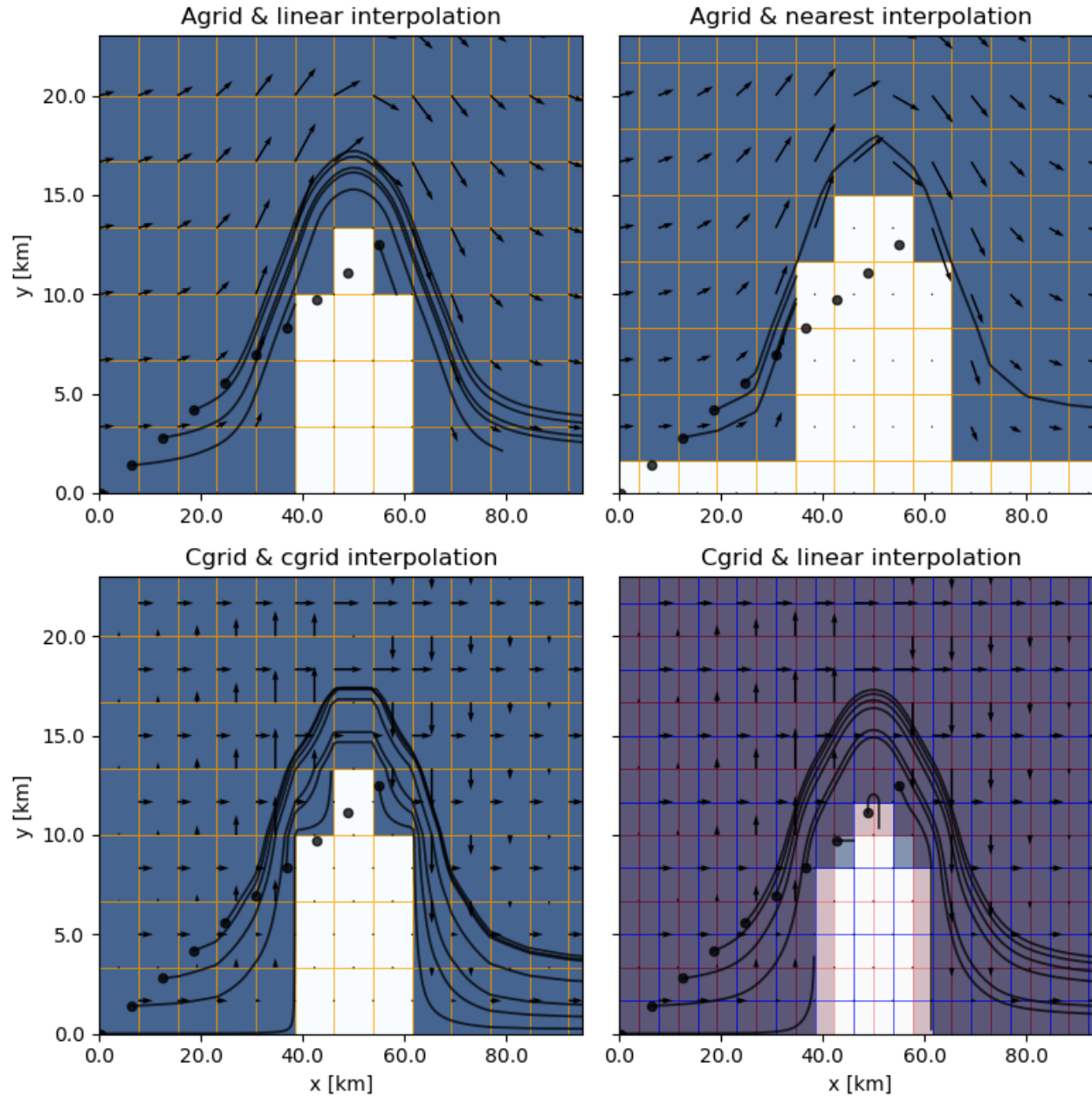
A grid



C grid

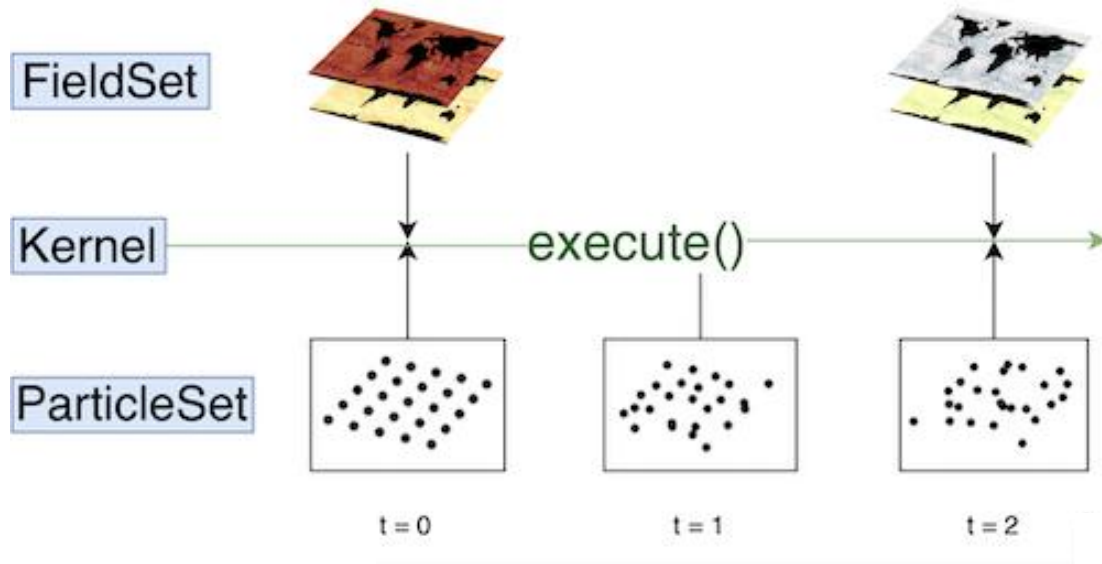


Why is this important?





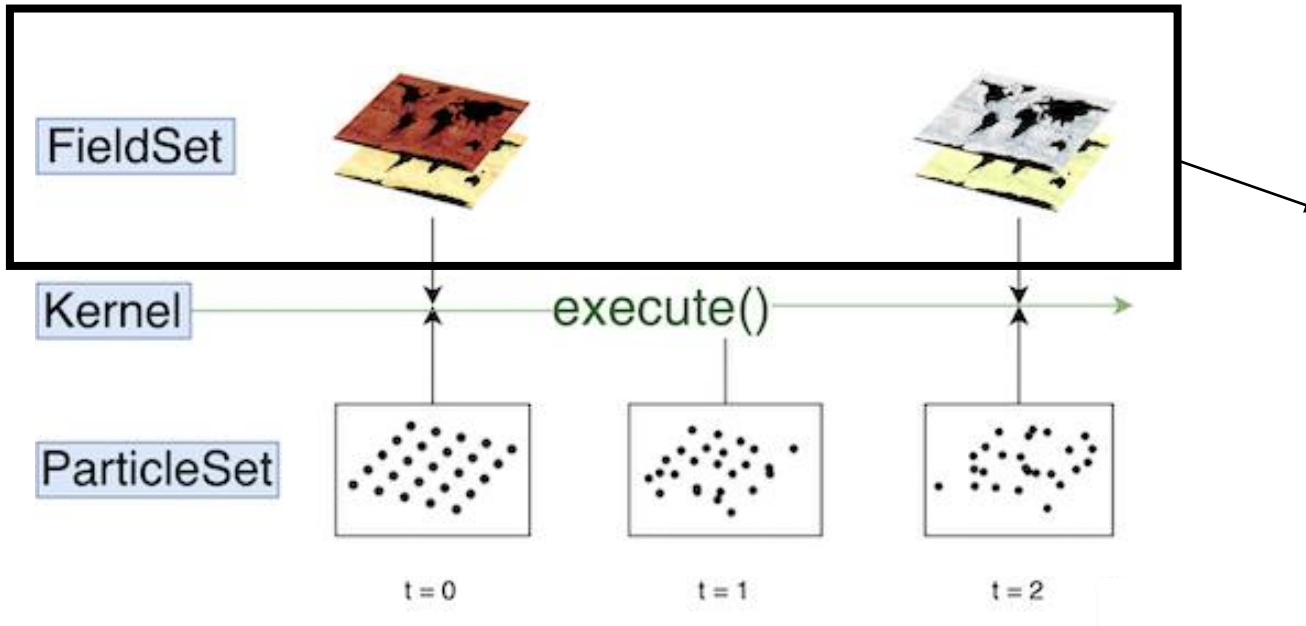
Parcels - structure



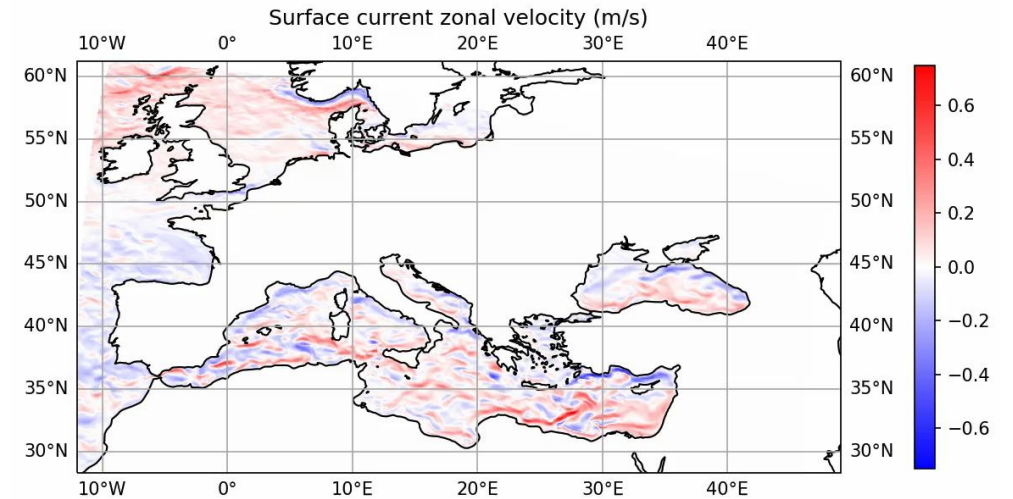


Parcels

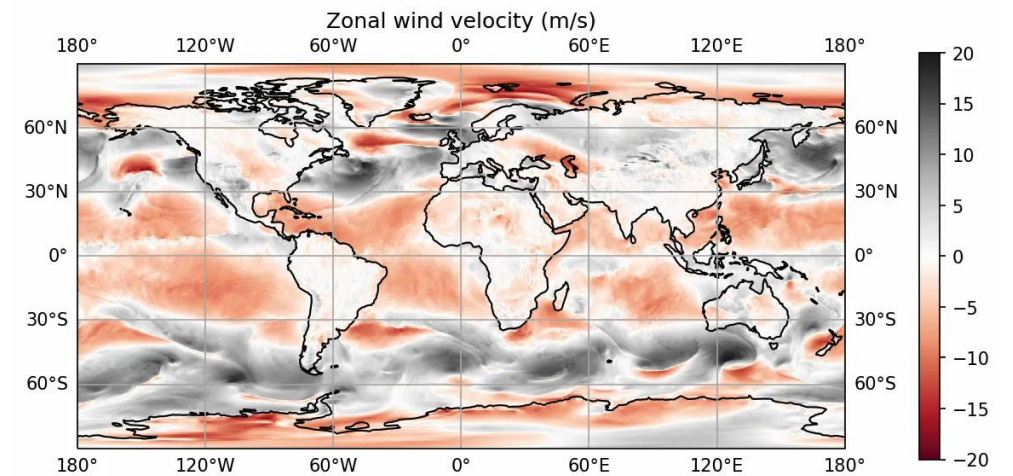
- structure



Ocean currents



Surface winds

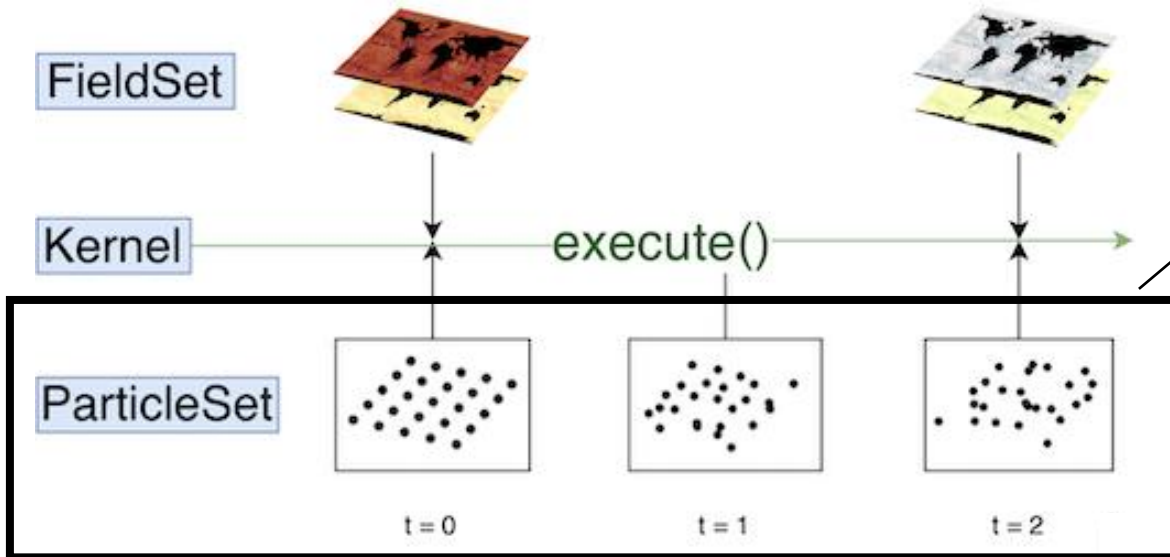


Your data outputs may be on different grids at different spatiotemporal resolutions



Parcels

- structure



NOTE: If lon and lat are in units of degrees, then `particle_dlon` and `particle_dlat` must compute displacements in units of degrees!

A particle has (at least) the following variables:

- `particle.lon` (longitude in degrees, or x position in m)
- `particle.lat` (latitude in degrees, or y position in m)
- `particle.depth` (depth in m)

You can create additional variables that can modify a particles position (or keep track of something, like temperature, salinity, biofilm algal amount).

At the beginning of each timestep, three “displacement / difference” variables are initialised to zero:

$$\begin{aligned}\text{particle_dlon} &= \Delta x = 0, \\ \text{particle_dlat} &= \Delta y = 0, \\ \text{particle_ddepth} &= \Delta z = 0\end{aligned}$$

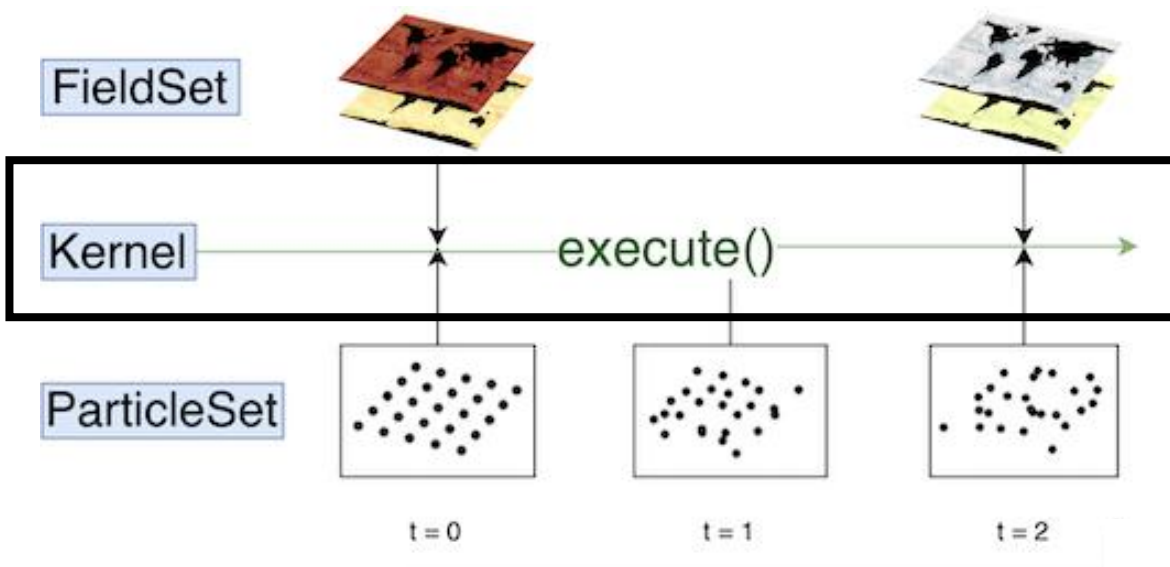
We loop through our kernels, using `+=` or `-=` to update these three variables, and at the end of the loop, our positions will be updated automatically

(e.g. `particle.lon += particle_dlon` - **DON'T DO THIS YOURSELF**).



Parcels

- structure



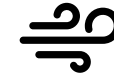
Custom kernels to simulate bio/chem/physical behaviour



Stokes Drift



Tuna Swimming



Windage



ARGO Floats



Biofouling



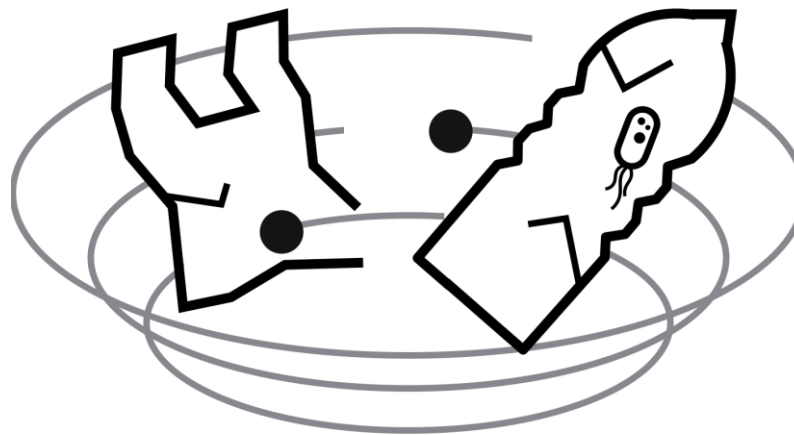
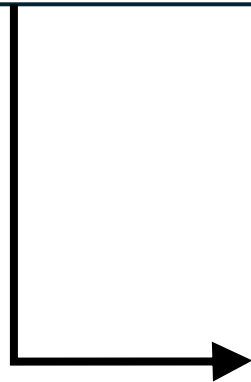
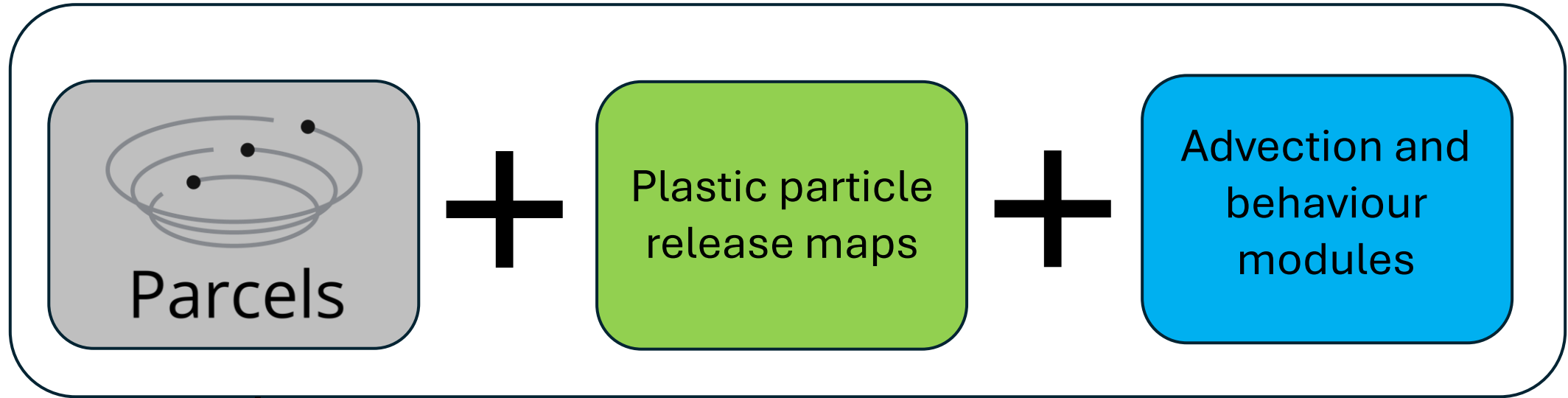
Icebergs



Sea-ice Capture



Turtle Drift



Plastic**Parcels**



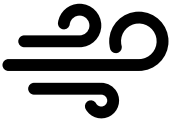
Plastic**Parcels** built-in kernels

Included particle dependent physics and behaviours

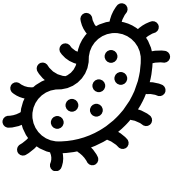


Stokes Drift

Breivik et al. (2016)



Wind-induced Drift



Biofouling

Kooi et al. (2017)



Sea-ice Capture*



Vertical Turbulent Mixing

Onink et al. (2022)

```
def Stokes_drift(particle, fieldset, time):  
    # Sample the U / V components of Stokes drift  
    stokes_U = fieldset.Stokes_U[time, particle.depth, particle.lat, particle.lon]  
    stokes_V = fieldset.Stokes_V[time, particle.depth, particle.lat, particle.lon]  
  
    # Sample the peak wave period  
    T_p = fieldset.wave_Tp[time, particle.depth, particle.lat, particle.lon]  
  
    # Only compute displacements if the peak wave period is large enough  
    if T_p > 1E-14:  
        # Peak wave frequency  
        omega_p = 2. * math.pi / T_p  
  
        # Peak wave number  
        k_p = (omega_p ** 2) / fieldset.G  
  
        # Repeated inner term of Eq. (19) – note depth is negative in this formulation  
        kp_z_2 = 2. * k_p * particle.depth  
  
        # Decay factor in Eq. (19) -- Where beta=1 for the Phillips spectrum  
        decay = math.exp(-kp_z_2) - math.sqrt(math.pi * kp_z_2) * math.erfc(math.sqrt(kp_z_2))  
  
        # Apply Eq. (19) and compute particle displacement  
        particle_dlon += stokes_U * decay * particle.dt  
        particle_dlat += stokes_V * decay * particle.dt
```



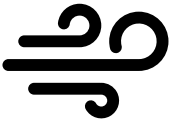
Plastic**Parcels** builtin kernels

Included particle dependent physics and behaviours

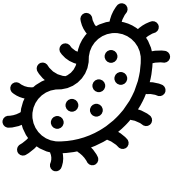


Stokes Drift

Breivik et al. (2016)



Wind-induced Drift



Biofouling

Kooi et al. (2017)



Sea-ice Capture*



Vertical Turbulent Mixing

Onink et al. (2022)

```
def WindageDrift(particle, fieldset, time):  
  
    # Sample ocean velocities  
    (ocean_U, ocean_V) = fieldset.UV[time, particle.depth, particle.lat, particle.lon]  
  
    # Sample the U / V components of wind  
    wind_U = fieldset.Wind_U[time, particle.depth, particle.lat, particle.lon]  
    wind_V = fieldset.Wind_V[time, particle.depth, particle.lat, particle.lon]  
  
    # Compute particle displacement using relative wind in an Euler-forward fashion  
    particle_dlon += particle.wind_coefficient * (wind_U - ocean_U) * particle.dt  
    particle_dlat += particle.wind_coefficient * (wind_V - ocean_V) * particle.dt
```

```

def Biofouling(particle, fieldset, time):
    # seawater_density = particle.seawater_density # [kg m-3]
    temperature = fieldset.conservative_temperature[time, particle.depth]
    seawater_salinity = fieldset.absolute_salinity[time, particle.depth]
    particle_radius = 0.5 * particle.plastic_diameter
    # particle_density = particle.plastic_density
    initial_settling_velocity = particle.settling_velocity # m/s

    # Compute the seawater dynamic viscosity and kinematic viscosity
    water_dynamic_viscosity = 4.2844E-5 + (1. / ((0.156 * temperature) +
    A = 1.541 + 1.998E-2 * temperature - 9.52E-5 * temperature**2
    B = 7.974 - 7.561E-2 * temperature + 4.724E-4 * temperature**2
    seawater_dynamic_viscosity = water_dynamic_viscosity * (1 + B * seawater_salinity)
    seawater_kinematic_viscosity = seawater_dynamic_viscosity / seawater_density

    # Compute the algal growth component
    # Sample fields
    mol_concentration_diatoms = fieldset.bio_diatom[time, particle.depth]
    mol_concentration_nanophytoplankton = fieldset.bio_nanophytoplankton[time, particle.depth]
    total_primary_production_of_phyto = fieldset.pp_phyto[time, particle.depth]
    median_mg_carbon_per_cell = 2726e-9 # Median mg of Carbon per cell
    # carbon_molecular_weight = fieldset.carbon_molecular_weight

    # Compute concentration numbers
    number_concentration_diatoms = mol_concentration_diatoms * median_mg_carbon_per_cell
    number_concentration_diatoms = max(number_concentration_diatoms, 1)
    number_concentration_nanophytoplankton = mol_concentration_nanophytoplankton * median_mg_carbon_per_cell
    number_concentration_nanophytoplankton = max(number_concentration_nanophytoplankton, 1)
    number_concentration_total = number_concentration_diatoms + number_concentration_nanophytoplankton

    # Compute primary production
    primary_production_per_cell = total_primary_production_of_phyto / number_concentration_total
    primary_production_numcell_per_cell = primary_production_per_cell * number_concentration_total
    primary_production_numcell_per_cell = max(primary_production_numcell_per_cell, 0)

    # Compute growth rates
    max_growth_rate = 1.85 # Maximum growth rate (per day), from literature
    mu_a = min(primary_production_numcell_per_cell, max_growth_rate)

    # Compute the algal growth of the algae already on the particle
    algae_growth = mu_a * particle.algae_amount # productivity

    # Compute the radius, surface area, volume and thickness of the particle
    particle_volume = (4. / 3.) * math.pi * particle_radius**3
    particle_surface_area = 4. * math.pi * particle_radius**2
    particle_thickness = particle_volume / particle_surface_area

    # Compute the dimensionless settling velocity w_*
    if dimensionless_diameter > 5E9: # "The boundary layer around the sphere becomes fully turbulent, causing a reduction in settling velocity"
        dimensionless_velocity = 265000. # Set a maximum dimensionless settling velocity
    elif dimensionless_diameter < 0.05: # "At values of D_* less than 0.05, (9) deviates significantly ... from Stokes' law"
        dimensionless_velocity = (dimensionless_diameter**2.) / 5832. # Using Eq. (8) in [1]
    else:
        dimensionless_velocity = 10. ** (-3.76715 + (1.92944 * math.log10(dimensionless_diameter)) - (0.09815 * math.log10(dimensionless_diameter)**2) + (0.00575 * math.log10(dimensionless_diameter)**3) + (0.00056 * math.log10(dimensionless_diameter)**4))

    # Compute the settling velocity of the particle using Eq. (5) from [1] (solving for the settling velocity)
    sign_of_density_difference = math.copysign(1., normalised_density_difference)
    settling_velocity = sign_of_density_difference * (fieldset.G * seawater_kinematic_viscosity * dimensionless_velocity)**0.5

    # Update the settling velocity
    particle.settling_velocity = settling_velocity

    # Update particle depth
    particle_ddepth += particle.settling_velocity * particle.dt # noqa

    # Compute the algal decay due to respiration
    a_respiration = fieldset.algae_respiration_f * (fieldset.Q10**((temperature - 20.) / 10.)) * fieldset.R20 * particle.algae_amount

    # Compute the algal decay due to grazing
    a_grazing = fieldset.algae_mortality_rate * particle.algae_amount

    # Compute the final algal amount
    algae_amount_change = (a_collision + algae_growth - a_grazing - a_respiration) * particle.dt
    if particle.algae_amount + algae_amount_change < 0.:
        particle.algae_amount = 0.
    else:
        particle.algae_amount += algae_amount_change

    # Compute the new settling velocity
    particle_diameter = 2. * (total_radius) # equivalent spherical diameter [m], calculated from Dietrich (1982) from A = pi/4 * D^2 * thickness

    # Compute the density difference of the particle
    normalised_density_difference = (total_density - particle.seawater_density) / particle.seawater_density # normalised difference in density

    # Compute the dimensionless particle diameter D_* using Eq. (4) from [2]
    dimensionless_diameter = (math.fabs(total_density - particle.seawater_density) * fieldset.G * particle_diameter**3.) / (4. * math.pi * particle_thickness * particle_seawater_density**2.)

```

```

def Biofouling(particle, fieldset, time):
    # seawater_density = particle.seawater_density # [kg m-3]
    temperature = fieldset.conservative_temperature[time, particle.depth]
    seawater_salinity = fieldset.absolute_salinity[time, particle.depth]
    particle_radius = 0.5 * particle.plastic_diameter
    # particle_density = particle.plastic_density
    initial_settling_velocity = particle.settling_velocity #

    # Compute the seawater dynamic viscosity and kinematic viscosity
    water_dynamic_viscosity = 4.2844E-5 + (1. / ((0.156 * (temperature - 15.41) + 1.998E-2 * temperature - 9.52E-5 * temperature**2) + 7.974 - 7.561E-2 * temperature + 4.724E-4 * temperature**2))
    seawater_dynamic_viscosity = water_dynamic_viscosity * (1. - 0.0001 * seawater_salinity)
    seawater_kinematic_viscosity = seawater_dynamic_viscosity / seawater_density

    # Compute the algal growth component
    # Sample fields
    mol_concentration_diatoms = fieldset.bio_diatom[time, particle.depth]
    mol_concentration_nanophytoplankton = fieldset.bio_nanophytoplankton[time, particle.depth]
    total_primary_production_of_phyto = fieldset.pp_phyto[time, particle.depth]
    median_mg_carbon_per_cell = 2726e-9 # Median mg of Carbon per cell
    # carbon_molecular_weight = fieldset.carbon_molecular_weight

    # Compute concentration numbers
    number_concentration_diatoms = mol_concentration_diatoms * 1e21 / median_mg_carbon_per_cell
    number_concentration_diatoms = max(number_concentration_diatoms, 0)
    number_concentration_nanophytoplankton = mol_concentration_nanophytoplankton * 1e21 / median_mg_carbon_per_cell
    number_concentration_nanophytoplankton = max(number_concentration_nanophytoplankton, 0)
    number_concentration_total = number_concentration_diatoms + number_concentration_nanophytoplankton

    # Compute the algal growth of the algae already present
    primary_production_numcell_per_cell = total_primary_production_of_phyto / number_concentration_total
    mu_a = min(primary_production_numcell_per_cell, max_growth_rate)

    # Compute the algal growth of the algae already present
    algae_growth = mu_a * particle.algae_amount

    # Compute the radius, surface area, volume and thickness
    particle_volume = (4. / 3.) * math.pi * particle_radius**3
    particle_surface_area = 4. * math.pi * particle_radius**2
    particle_thickness = particle_volume / particle_surface_area

    # Compute the dimensionless settling velocity w_*
    if dimensionless_diameter > 5E9: # "The boundary layer around the sphere becomes fully turbulent, causing a reduction in settling velocity"
        dimensionless_velocity = 265000. # Set a maximum dimensionless settling velocity
    elif dimensionless_diameter < 0.05: # "At values of D_* less than 0.05, (9) deviates significantly ... from Stokes' law"
        dimensionless_velocity = (dimensionless_diameter**2.) / 5832. # Using Eq. (8) in [1]
    else:
        dimensionless_velocity = 10. ** (-3.76715 + (1.92944 * math.log10(dimensionless_diameter)) - (0.09815 * math.log10(dimensionless_diameter)**2) - (0.00575 * math.log10(dimensionless_diameter)**3) + (0.00056 * math.log10(dimensionless_diameter)**4))

    # Compute the settling velocity of the particle using Eq. (5) from [1] (solving for the settling velocity)
    sign_of_density_difference = math.copysign(1., normalised_density_difference)
    settling_velocity = sign_of_density_difference * (fieldset.G * seawater_kinematic_viscosity * dimensionless_velocity)

    # Update the settling velocity
    particle.settling_velocity = settling_velocity

    # Update particle depth
    particle_ddepth += particle.settling_velocity * particle.dt # noqa

    # Compute the algal decay due to respiration
    a_respiration = fieldset.algae_respiration_f * (fieldset.Q10**((temperature - 20.) / 10.)) * fieldset.R20 * particle.algae_amount

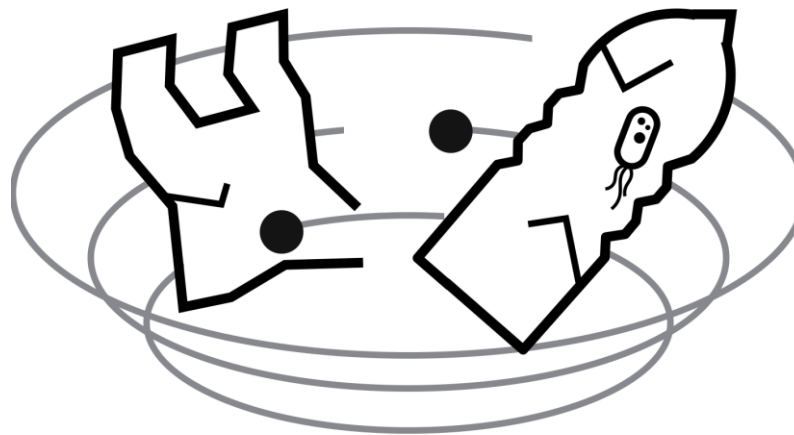
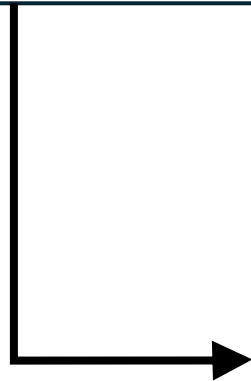
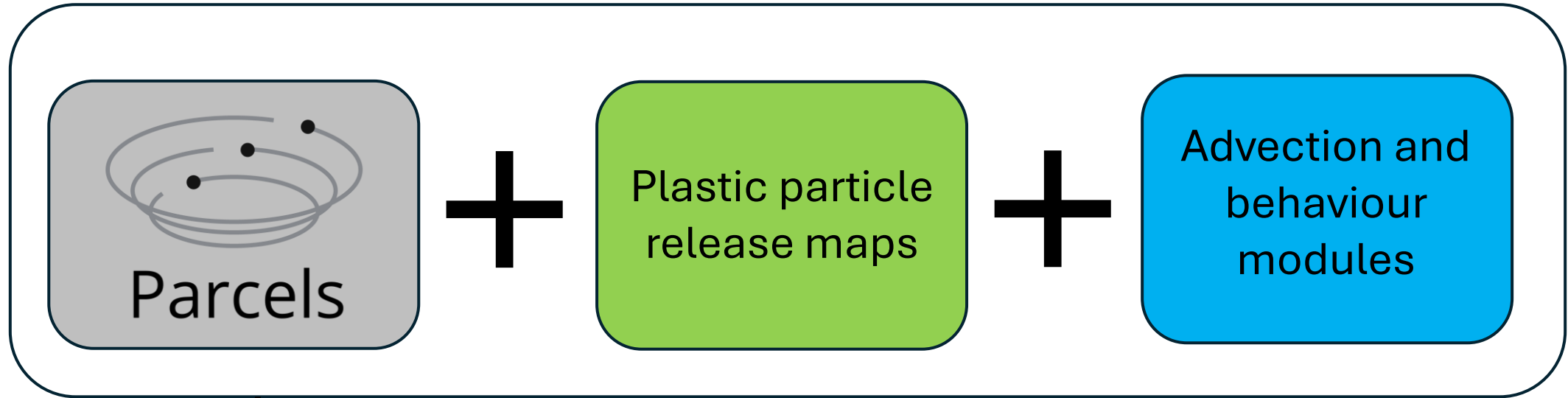
    # Compute the algal decay due to grazing
    a_grazing = fieldset.algae_mortality_rate * particle.algae_amount

    # Compute the final algal amount
    algae_amount_change = (a_collision + algae_growth - a_grazing - a_respiration) * particle.dt
    particle.algae_amount += algae_amount_change

    # Compute the new settling velocity
    particle_diameter = 2. * (total_radius) # equivalent spherical diameter [m], calculated from Dietrich (1982) from A = pi/4 * D^2
    dimensionless_diameter = (math.fabs(total_density - particle.seawater_density) * fieldset.G * particle_diameter**3.) / (4. * math.pi * particle_density * particle_kinematic_viscosity)
    particle.settling_velocity = settling_velocity

```

All this code just for a boring Euler-forward scheme....

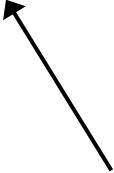


Plastic**Parcels**

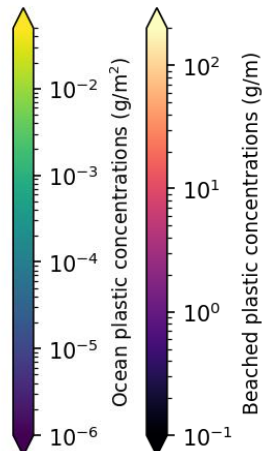
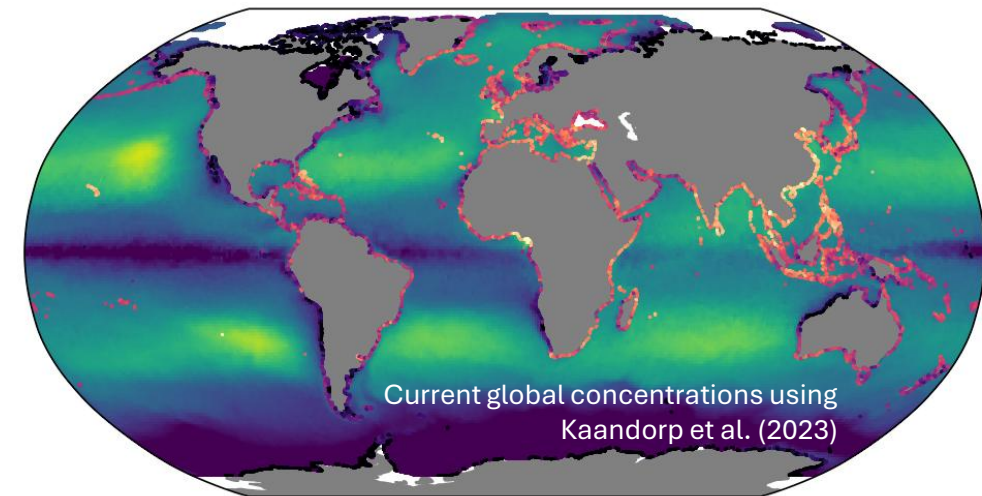
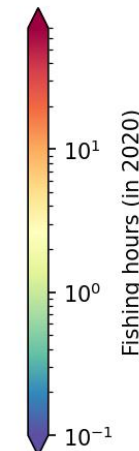
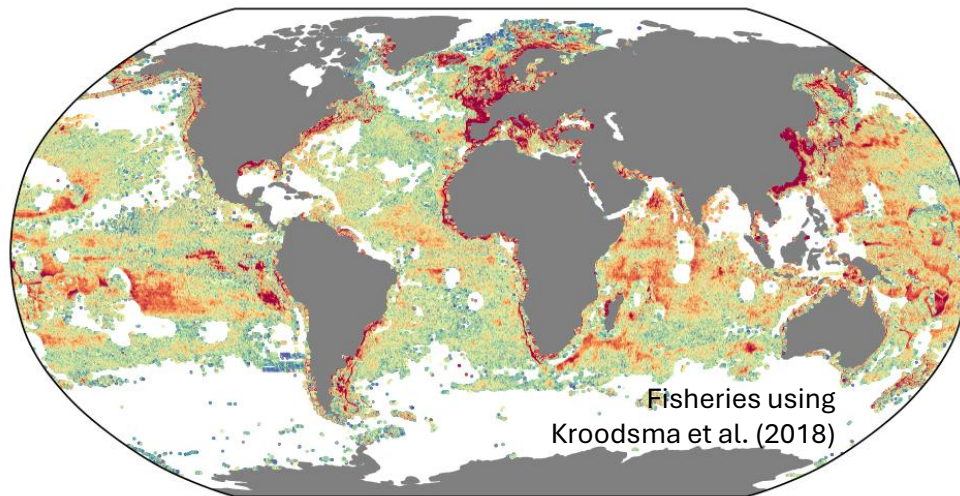
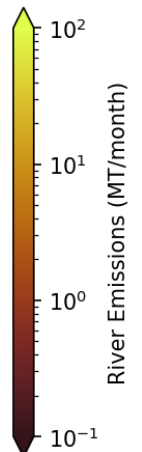
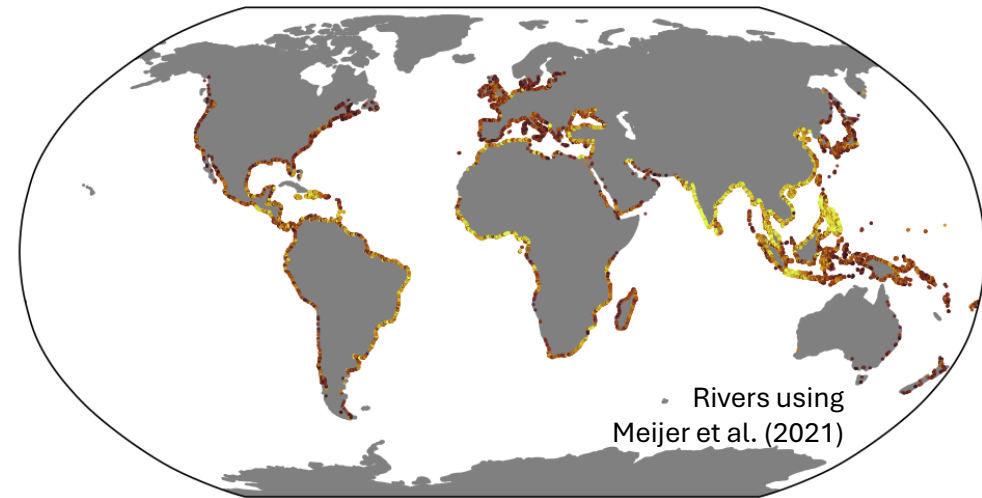
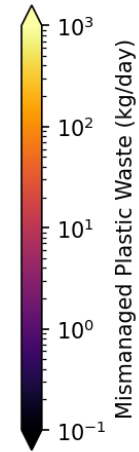
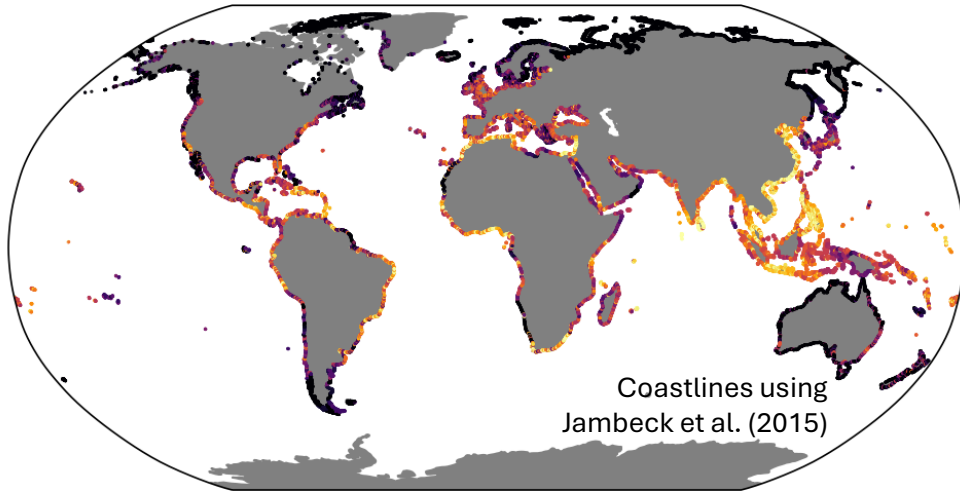
Where to release particles?

$$\mathbf{x}(t) = \mathbf{x}(0) + \int_0^t \mathbf{v}(\mathbf{x}(s), s) + \mathbf{B}(\mathbf{x}(s), s) ds$$

An Initial Value Problem



Plastic pollution release maps





Required Data

`plasticparcels` has been developed for use with data from the Copernicus Marine service and requires the following data to run:

- Hydrodynamic model data: [MOI GLO12 \(psy4v3r1\)](#)
- Biogeochemical model data: [MOI BIO4 \(biomer4v2r1\)](#)
- Wave data: [ECMWF ERA5 Wave](#) (specifically the variables `mean_wave_period`, `peak_wave_period`, `u_component_stokes_drift`, and `v_component_stokes_drift`.)
- Wind data: [ECMWF ERA5 Wind](#) (specifically the variables `10m_u_component_of_wind` and `10m_v_component_of_wind`)

For downloading the wind and wave data, we recommend using the [CDS API](#).

To run the examples, you will need to update the data directories in the relevant settings `.json` file.

Outside of the `parcels` framework, `plasticparcels` can be adapted to use other hydrodynamic, biogeochemical, wave, and atmospheric models. If you require assistance, please contact us through the [Discussions page on GitHub](#).

New functionality – can create fieldsets using the *copernicusmarine* toolkit*

*current working branch - https://github.com/Parcels-code/plasticparcels/tree/copernicus_marine

Example using *copernicusmarine**

```
# Load or create a copernicus settings dictionary
settings_file = 'copernicus_marine.json'
settings = pp.utils.load_settings(settings_file)
```

```
"use_3D": false,
"allow_time_extrapolation": false,
"verbose_delete": false,
"use_mixing": false,
"use_biofouling": false,
"use_stokes": true,
"use_wind": false,
```

```
{ } copernicus_marine.json M X
docs > examples > { } copernicus_marine.json > ...
8  "use_wind": false,
9  "ocean": {
10     "dataset_id": {"U": "cmems_mod_glo_phy-cur_anfc_0.083deg_P1D-m",
11                  "V": "cmems_mod_glo_phy-cur_anfc_0.083deg_P1D-m",
12                  "W": "cmems_mod_glo_phy-wcur_anfc_0.083deg_P1D-m",
13                  "conservative_temperature": "cmems_mod_glo_phy-thetao_anfc_0.083deg_P1D-m",
14                  "absolute_salinity": "cmems_mod_glo_phy-so_anfc_0.083deg_P1D-m",
15                  "bathymetry": "cmems_mod_glo_phy_anfc_0.083deg_static"},
16     "variables": {"U": "uo",
17                  "V": "vo",
18                  "W": "wo",
19                  "conservative_temperature": "thetao",
20                  "absolute_salinity": "so"},
21     "dimensions": {"lon": "longitude",
22                   "lat": "latitude",
23                   "depth": "depth",
24                   "time": "time"},
25     "bathymetry_variables": {"bathymetry": "deptho"},
26     "bathymetry_dimensions": {"lon": "longitude",
27                               "lat": "latitude"}
28 },
29 "bgc": {
30     "dataset_id": {"pp_phyto": "cmems_mod_glo_bgc-bio_anfc_0.25deg_P1D-m",
31                  "bio_nanophy": "cmems_mod_glo_bgc-pft_anfc_0.25deg_P1D-m",
32                  "bio_diatom": "cmems_mod_glo_bgc-plankton_anfc_0.25deg_P1D-m"},
33     "variables": {
34         "pp_phyto": "nppv",
35         "bio_nanophy": "phyc",
36         "bio_diatom": "zooc"
37     },
38     "dimensions": {"lon": "longitude",
39                   "lat": "latitude",
40                   "depth": "depth",
41                   "time": "time"},
42 }
```

*current working branch - https://github.com/Parcels-code/plasticparcels/tree/copernicus_marine

Setup the simulation

```
# Set the simulation settings
settings['simulation'] = {
    # Start date of simulation
    'startdate': datetime.strptime('2025-01-01-00:00:00', '%Y-%m-%d-%H:%M:%S'),
    'runtime': timedelta(days=10),          # Runtime of simulation
    'outputdt': timedelta(hours=6),         # Timestep of output
    'dt': timedelta(minutes=10),           # Timestep of advection

    # Specific to copernicusmarine datasets, provide a bounding box and depth range for your simulation
    'boundingbox': [-5.0, 30.0, 30.0, 45.0], # Bounding box of simulation [lonmin, lonmax, latmin, latmax]}
    'depth_range': [0, 5],                 # Depth range of simulation [depthmin, depthmax] in meters
}

# Create the particle release settings
settings['release'] = {
    'initialisation_type': 'fisheries',
    'country': 'Italy',
}

# Create the plastic type settings
settings['plastictype'] = {
    'wind_coefficient' : 0.0, # Percentage of wind to apply to particles
    'plastic_diameter' : 0.001, # Plastic particle diameter (m)
    'plastic_density' : 1028., # Plastic particle density (kg/m^3)
}
```

Construct the model and execute

```
# Create the fieldset, particleset, and kernels for the simulation
fieldset = pp.constructors.create_fieldset(settings)
pset = pp.constructors.create_particleset_from_map(fieldset, settings)
kernels = pp.constructors.create_kernel(fieldset)

runtime = settings['simulation']['runtime']
dt = settings['simulation']['dt']
outputdt = settings['simulation']['outputdt']

# Create the particle file where output will be stored
pfile = pp.ParticleFile('example_copernicusmarine.zarr', pset, settings=settings, outputdt=outputdt)

# Execute the simulation
pset.execute(kernels, runtime=runtime, dt=dt, output_file=pfile)
```

Output

```
# Output
ds = xr.open_zarr('example_copernicusmarine.zarr')
ds
✓ 0.2s
```

xarray.Dataset

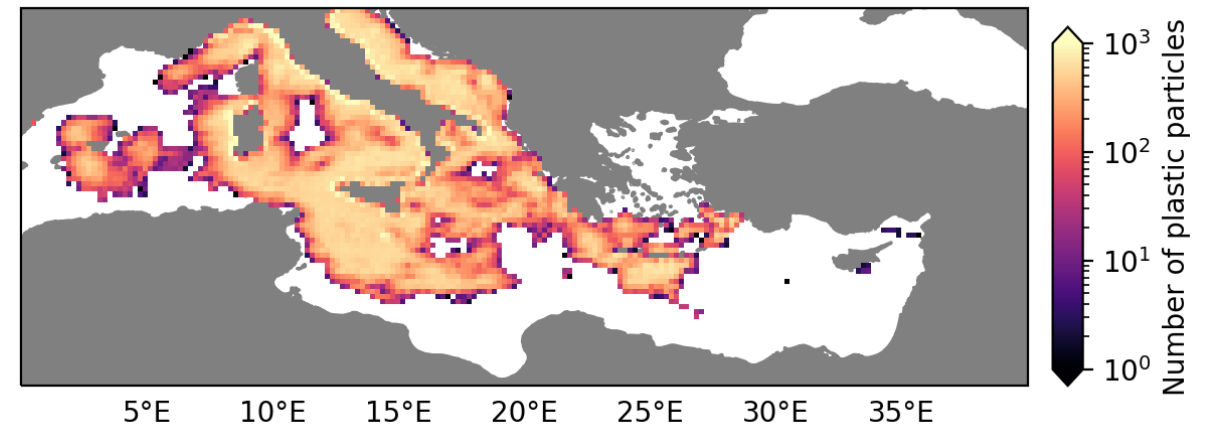
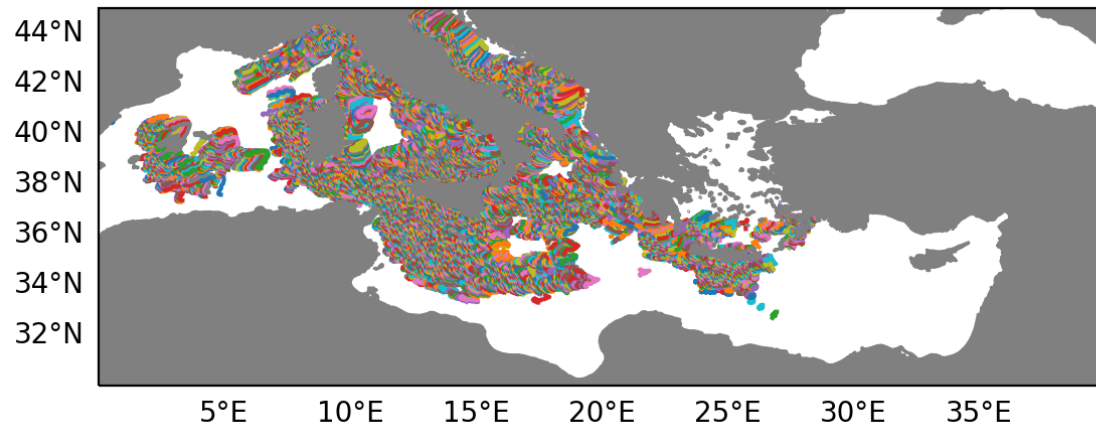
► Dimensions: (trajectory: 13298, obs: 40)

▼ Coordinates:

obs	(obs)	int32	0 1 2 3 4 5 6 ... 34 35 36 37 38 39		
trajectory	(trajectory)	int64	13298 13299 13300 ... 26379 26380		

▼ Data variables:

lat	(trajectory, obs)	float32	dask.array<chunksize=(730, 1), meta=np.ndarray>		
lon	(trajectory, obs)	float32	dask.array<chunksize=(730, 1), meta=np.ndarray>		
plastic_amount	(trajectory, obs)	float32	dask.array<chunksize=(730, 1), meta=np.ndarray>		
time	(trajectory, obs)	datetime64[ns]	dask.array<chunksize=(730, 1), meta=np.ndarray>		
z	(trajectory, obs)	float32	dask.array<chunksize=(730, 1), meta=np.ndarray>		



Another example (Italian coastlines)

```

set
set
# Cr
set
# Execute the simulation
pset.execute(kernels, runtime=runtime, dt=dt_timestep, output_file=pfile)

```

```

'runtime': timed
'dt_write': time
'dt_timestep': t
}

```

```

# Overwrite some set
settings['use_3D'] =
settings['use_biofou
settings['use_stokes
settings['use_wind']

```

```

# Create the particl
settings['release']
'initialisation_
'country': 'Ital
}

```

```

# Create the plastic
settings['plastictyp
'wind_coeficien
'plastic_diamete
'plastic_density
}

```

```

# Download the mask
settings = pp.utils.

```

```

# Create the fieldse
fieldset = pp.constr

```

```

# Create the particl
pset = pp.constructo

```

```

# Create the applica
kernels = pp.constru

```

```

# Define the runtime
runtime = settings['
dt_timestep = settin
dt_write = settings[

```

```

# Create the particl
pfile = pp.ParticleF

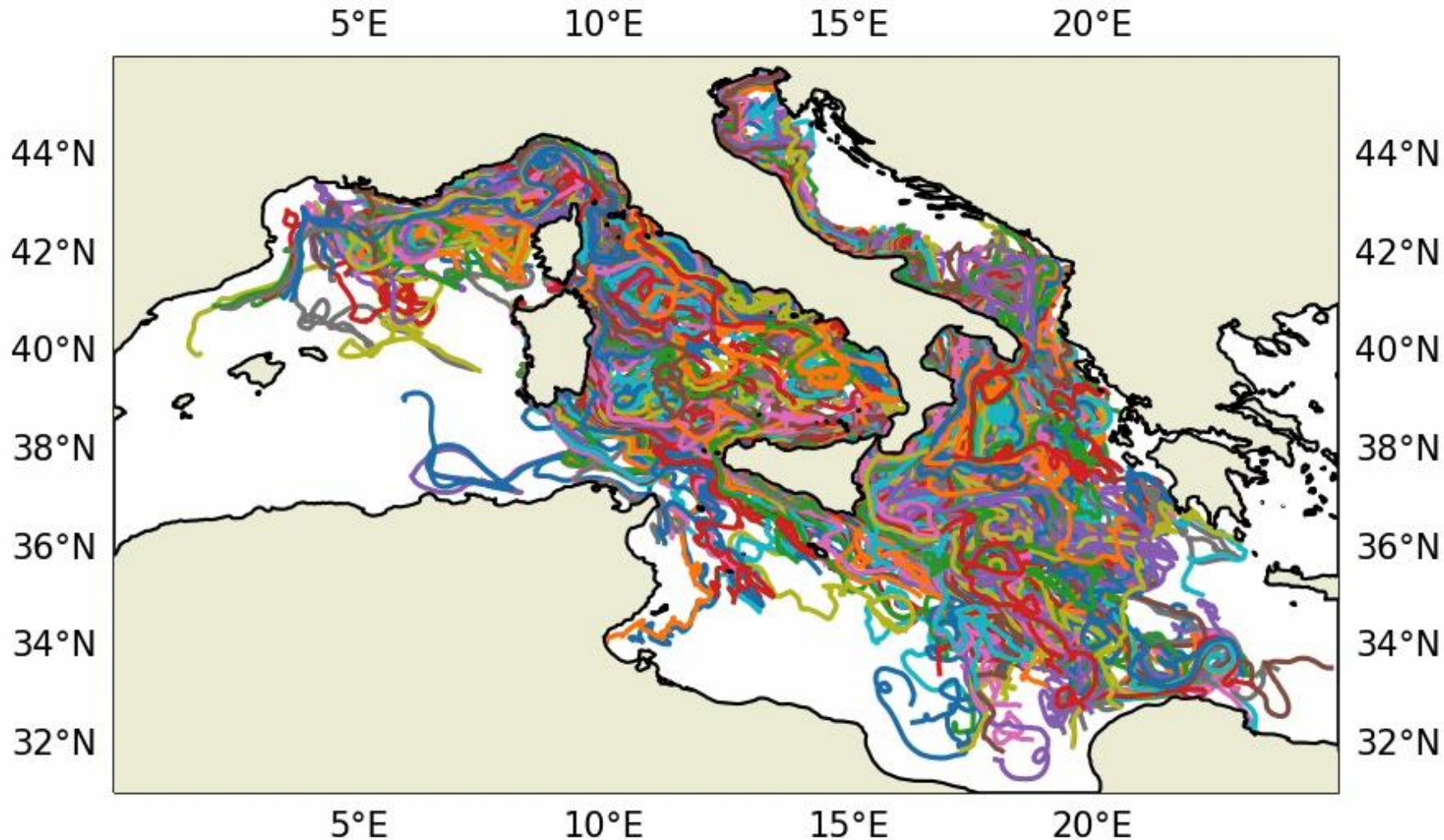
```

```

# Execute the simula
pset.execute(kernels

```

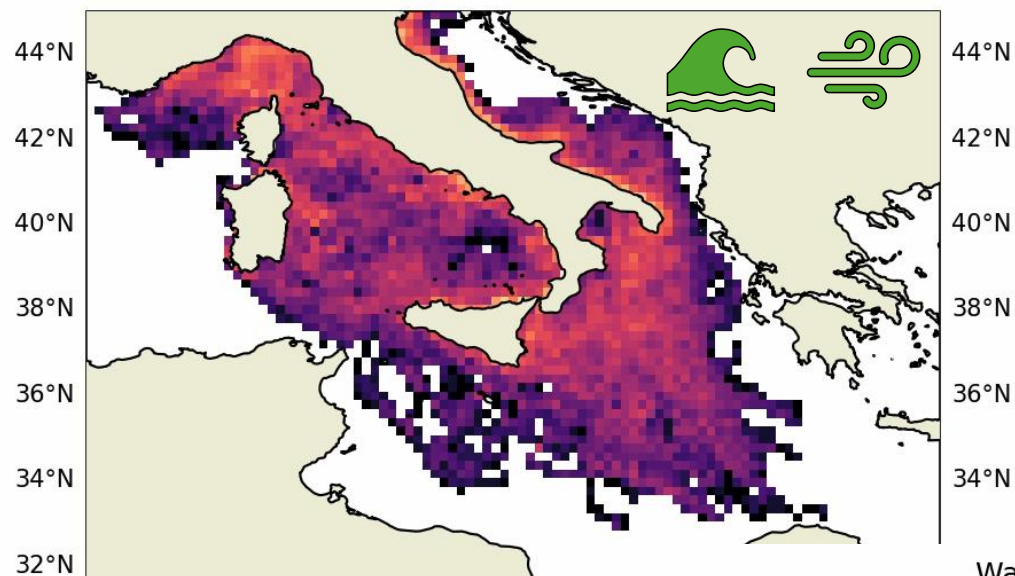
Output timestep: 715



The effect of turning on/off different kernels

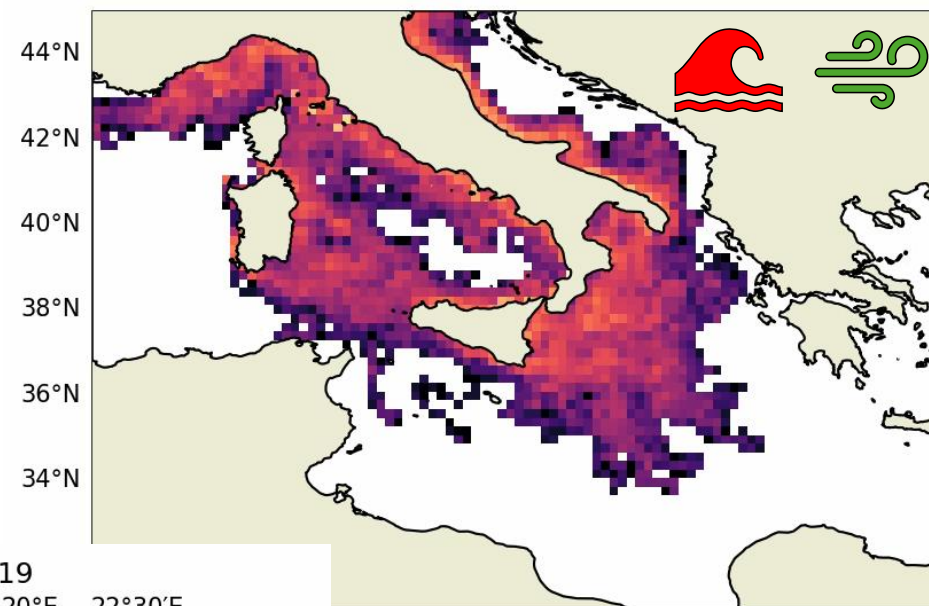
Wind+Waves - Output timestep: 119

7°30'E 10°E 12°30'E 15°E 17°30'E 20°E 22°30'E



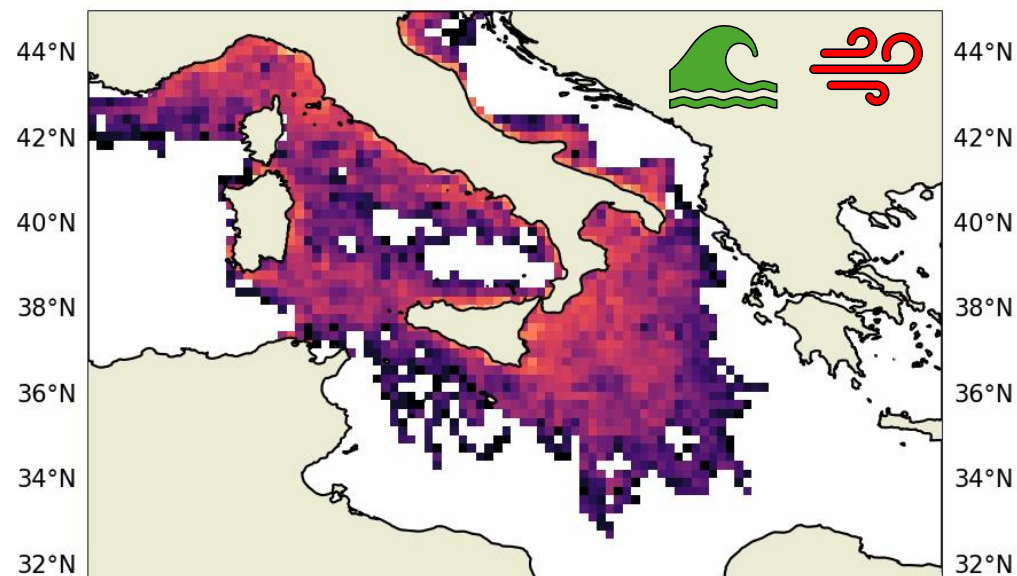
Wind - Output timestep: 119

7°30'E 10°E 12°30'E 15°E 17°30'E 20°E 22°30'E

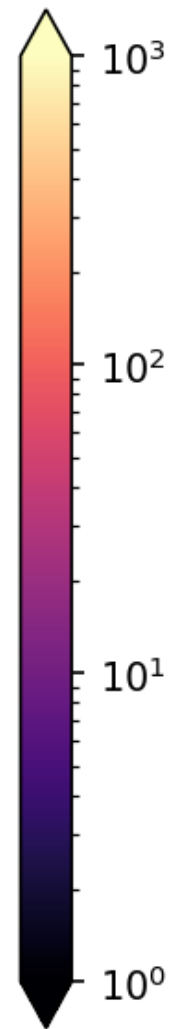


Waves - Output timestep: 119

7°30'E 10°E 12°30'E 15°E 17°30'E 20°E 22°30'E



7°30'E 10°E 12°30'E 15°E 17°30'E 20°E 22°30'E



Parcels v4 – currently in development

An alpha mode is available for testing! Currently running a hackathon in Kiel, Germany!

The key goals of this update are

1. to support `Fields` on unstructured grids;
2. to allow for user-defined interpolation methods (somewhat similar to user-defined kernels);
3. to make the codebase more modular, easier to extend, and more maintainable;
4. to align Parcels more with other tools in the [Pangeo ecosystem](#), particularly by leveraging `xarray` more; and
5. to improve the performance of Parcels.

Parcels v3 tips (and quirks)

In parcels v3, there are two “modes” of simulations we can; using **JIT** (Just-in-time, code is compiled and run in C and using **Scipy** (code is run natively in python).

When using JIT mode:

- Convert any integer into a float (e.g. 2/3 write as 2./3.)
- Can't use numpy, write everything using the math library
 - math.radians doesn't work as you will see in the later notebook... make the conversion yourself!
 - math.abs turns floats into integers, use math.fabs!
- No function calls, or complex numbers, or

Pro tip: Develop your kernel in Scipy mode, using a single timestep to check the results. When using JIT mode, your JIT simulation should match your Scipy simulation. If they don't likely one of the issues above has popped up!

v4 solves all these problems!!!

Notebook 1 - Developing advection and wind-induced drift kernels

advection_and_windage.ipynb

Part 1

- Create a second-order Runge-Kutta advection scheme
- Compare trajectories vs. built-in parcels EE and RK4 schemes
- Compare how changing timestep size affects your trajectories

Part 2

- Create different wind-induced drift kernels
- Compare trajectories!

Notebook 2 – Using parcels as an ODE solver

lorenz_and_lotka_volterra.ipynb

Part 1

- Create different kernels to solve the Lorenz system
- Compare trajectories and “Energy” to determine suitable solutions

Part 2

- Create different kernels to solve the Lotka-Volterra system
- Compare trajectories to determine suitable solutions