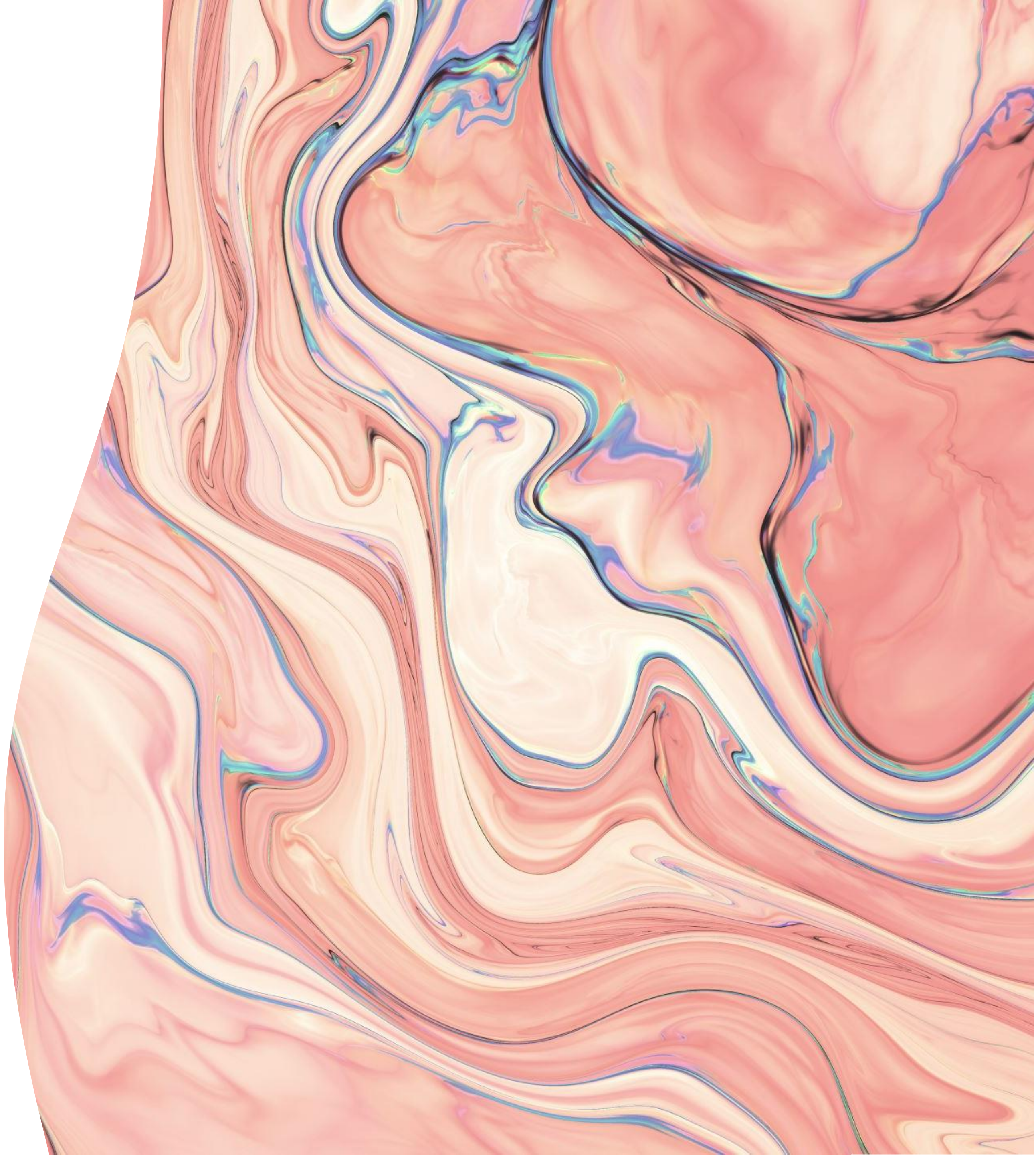
Workshop session 2:

Make your own Kernel

Michael Denes – postdoc in parcels group



Agenda

- A brief discussion on numerical modelling
- Parcels kernels – what are they, how do they work, and particle variables

Notebook 1:

  - Creating a simple advection kernel
  - Creating a wind-induced drift kernel

Notebook 2:

  - Using parcels as an ODE Solver – The Lorenz attractor and the Lotka-Volterra predator-prey model

- Kernel sharing session!

Numerical modelling of trajectories

The equation we are trying to integrate:

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

Water velocity

Particle-dependent physics

- Buoyancy forces
- Wind-drag
- “missing”/unresolved physics from ocean models

Particle-dependent behaviour

- Biofouling
- Swimming
- Dial vertical migration

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

Numerical modelling of trajectories

The equation we are trying to integrate:

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

Water velocity

Particle-dependent physics

- Buoyancy forces
- Wind-drag
- “missing”/unresolved physics from ocean models

Particle-dependent behaviour

- Biofouling
- Swimming
- Dial vertical migration

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

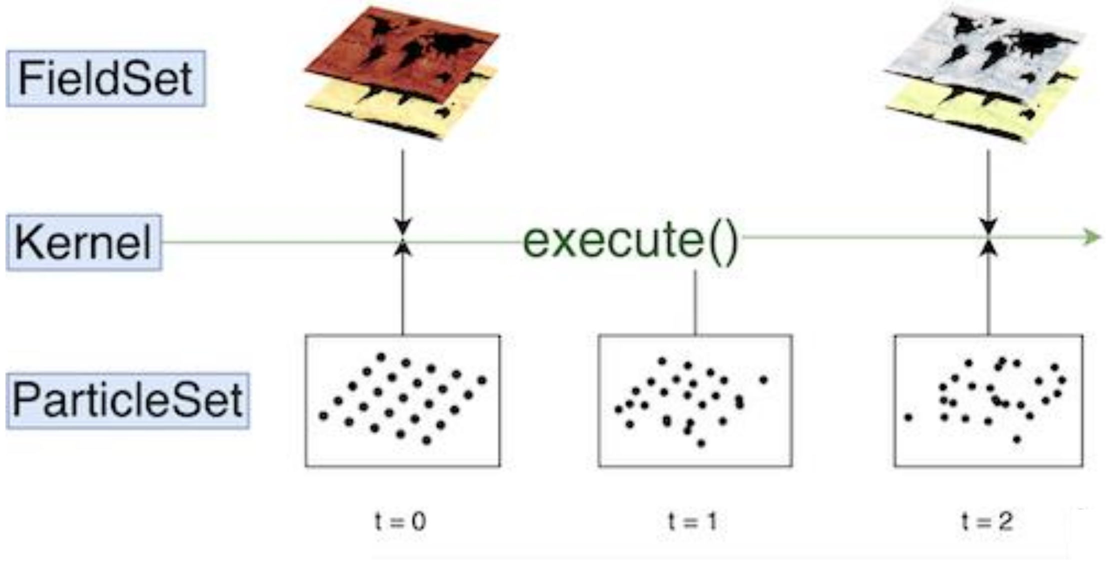
Explicit-Euler formation

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

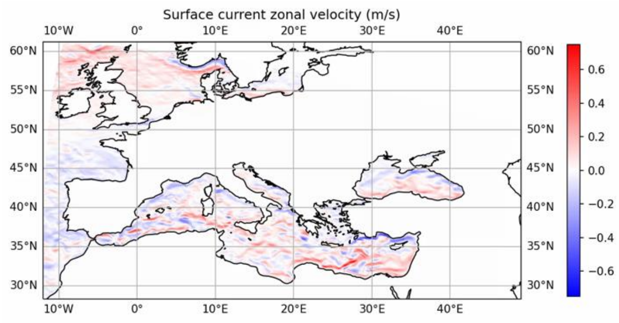
$$\boxed{\Delta\mathbf{x} = \mathbf{v}(\mathbf{x}(t), t)\Delta t}$$

Displacement (change in position) = velocity × timestep

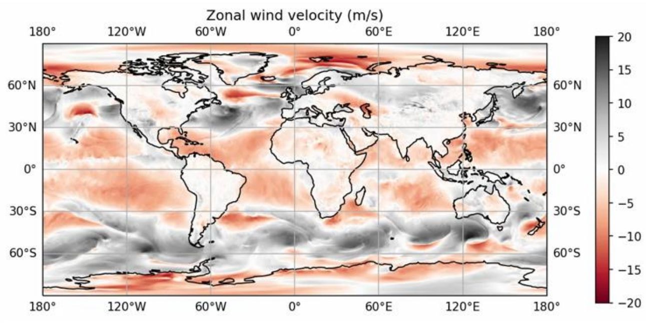
Parcels - structure



Ocean currents



Surface winds



Your kernels may require outputs on different grids at different spatiotemporal resolutions

Custom kernels to simulate bio/chem/physical behaviour

- Stokes Drift
- Tuna Swimming
- Windage
- ARGO Floats
- Biofouling
- Icebergs
- Sea-ice Capture
- Turtle Drift

How do kernels work?

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, *soon to be changed to z*)

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

particle_dlon = Δx = 0, particle_dlat = Δy = 0, particle_ddepth = Δz = 0

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**).

How do kernels work?

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, *soon to be changed to z*)

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

particle_dlon = Δx = 0, particle_dlat = Δy = 0, particle_ddepth = Δz = 0

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**).

PlasticParcels in-built 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)

```python
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
```

```python
def Biofouling(particle, fieldset, time):

# seawater_density = particle.seawater_density  # [kg m-3
temperature = fieldset.conservative_temperature[time, par
seawater_salinity = fieldset.absolute_salinity[time, part
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 vi
water_dynamic_viscosity = 4.2844E-5 + (1. / ((0.156 * (te
A = 1.541 + 1.998E-2 * temperature - 9.52E-5 * temperatur
B = 7.974 - 7.561E-2 * temperature + 4.724E-4 * temperatu
seawater_dynamic_viscosity = water_dynamic_viscosity * (1
seawater_kinematic_viscosity = seawater_dynamic_viscosity

# Compute the algal growth component
# Sample fields
mol_concentration_diatoms = fieldset.bio_diatom[time, par
mol_concentration_nanophytoplankton = fieldset.bio_nanoph
total_primary_production_of_phyto = fieldset.pp_phyto[tim
median_mg_carbon_per_cell = 2726e-9  # Median mg of Carbo
# carbon_molecular_weight = fieldset.carbon_molecular_wei

# Compute concentration numbers
number_concentration_diatoms = mol_concentration_diatoms 
number_concentration_diatoms = max(number_concentration_d
number_concentration_nanophytoplankton = mol_concentratio
number_concentration_nanophytoplankton = max(number_conce
number_concentration_total = number_concentration_diatoms

# Compute primary production
primary_production_per_cell = total_primary_production_of
primary_production_numcell_per_cell = primary_production_
primary_production_numcell_per_cell = max(primary_product

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

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

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

```python
particle_surface_a
algal_cell_radius 
biofilm_volume = f

total_volume = bio
total_radius = ((t

# Compute diffusiv
total_density = (p
plastic_diffusivit
algae_diffusivity 

# Compute the enco
beta_abrown = 4. *
beta_ashear = 1.3 
beta_aset = (1. / 
beta_a = beta_abro

# Compute the alga
a_collision = fiel

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

# 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/

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

# 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.) / (
```

```python
# Compute the dimensionless settling velocity w_*
if dimensionless_diameter > 5E9:  # "The boundary layer around the sphere becomes fully turbulent, causing a reduct
    dimensionless_velocity = 265000.  # Set a maximum dimensionless settling velocity
elif dimensionless_diameter < 0.05:  # "At values of D_* less than 0.05, (9) deviates signficantly ... from Stokes'
    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.lo
                                    0.00575 * math.log10(dimensionless_diameter) ** 3.) + (0.00056 * math.log10(di

# 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_velocit

# Update the settling velocity
particle.settling_velocity = settling_velocity

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

```python
def Biofouling(particle, fieldset, time):

# seawater_density = particle.seawater_density  # [kg m-3
temperature = fieldset.conservative_temperature[time, par
seawater_salinity = fieldset.absolute_salinity[time, part
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 vi
water_dynamic_viscosity = 4.2844E-5 + (1. / ((0.156 * (te
A = 1.541 + 1.998E-2 * temperature - 9.52E-5 * temperatur
B = 7.974 - 7.561E-2 * temperature + 4.724E-4 * temperatu
seawater_dynamic_viscosity = water_dynamic_viscosity * (1
seawater_kinematic_viscosity = seawater_dynamic_viscosity

# Compute the algal growth component
# Sample fields
mol_concentration_diatoms = fieldset.bio_diatom[time, par
mol_concentration_nanophytoplankton = fieldset.bio_nanoph
total_primary_production_of_phyto = fieldset.pp_phyto[tim
median_mg_carbon_per_cell = 2726e-9  # Median mg of Carbo
# carbon_molecular_weight = fieldset.carbon_molecular_wei

# Compute concentration numbers
number_concentration_diatoms = mol_concentration_diatoms
number_concentration_diatoms = max(number_concentration_d
number_concentration_nanophytoplankton = mol_concentratio
number_concentration_nanophytoplankton = max(number_conce
number_concentration_total = number_concentration_diatoms

# Compute
primary_pr
primary_pr
primary_pr

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

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

# Compute the radius, surface area, volume
particle_volume = (4. / 3.) * math.pi * part
```

```python
particle_surface_a
algal_cell_radius
biofilm_volume = f

total_volume = bio
total_radius = ((t

# Compute diffusiv
total_density = (p
plastic_diffusivit
algae_diffusivity

# Compute the enco
beta_abrown = 4. *
beta_ashear = 1.3
beta_aset = (1. /
beta_a = beta_abro

# Compute the alga
a_collision = fiel

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

# 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_diameter = 2. * (total_radius)  # equivalent spherical diameter [m], calculated from Dietrich (1982) from A = pi/
```

```python
# Compute the dimensionless settling velocity w_*
if dimensionless_diameter > 5E9:  # "The boundary layer around the sphere becomes fully turbulent, causing a reduct
    dimensionless_velocity = 265000.  # Set a maximum dimensionless settling velocity
elif dimensionless_diameter < 0.05:  # "At values of D_* less than 0.05, (9) deviates signficantly ... from Stokes'
    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.lo
                                    0.00575 * math.log10(dimensionless_diameter) ** 3.) + (0.00056 * math.log10(di

# 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_velocit

# Update the settling velocity
particle.settling_velocity = settling_velocity

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

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

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

(parcels v4 allows for calling functions!)

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

Work in groups, and use: 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!

Go here: edu.nl/h4y8w

Notebook 2 – Using parcels as an ODE solver

Work in groups and use: 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

Go here: **edu.nl/h4y8w**

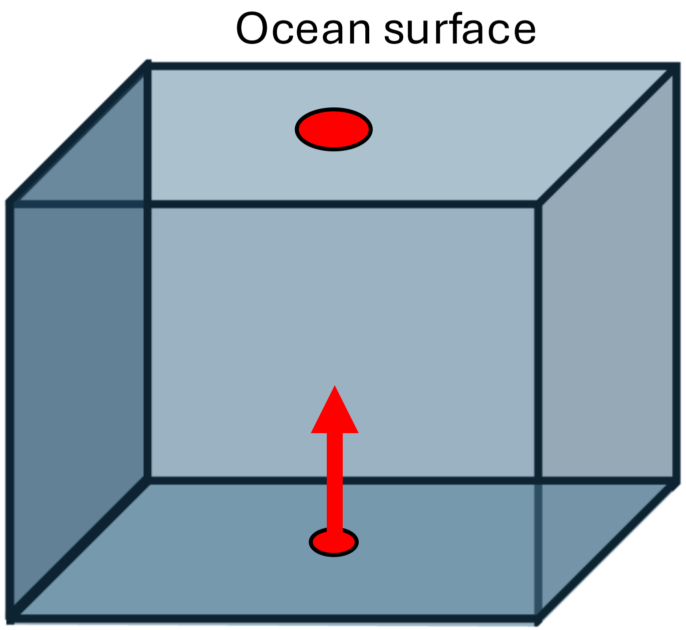
Kernel Sharing Session

Or kernel show-and-tell!

Let’s discuss/share some kernels that we have been working on!

Now is a good time to ask for help, guidance, or clarification.

Bonus – An error handling example




```python
def checkErrorThroughSurface_2DAdvection(particle, fieldset, time):
    # This is a kernel to handle 3D advection leading to ErrorThroughSurface
    if particle.state == StatusCode.ErrorThroughSurface:
        # Perform 2D horizontal advection only!
        """Advection of particles using fourth-order Runge-Kutta integration."""
        (u1, v1) = fieldset.UV[particle]
        lon1, lat1 = (particle.lon + u1 * 0.5 * particle.dt, particle.lat + v1 * 0.5 * particle.dt)
        (u2, v2) = fieldset.UV[time + 0.5 * particle.dt, particle.depth, lat1, lon1, particle]
        lon2, lat2 = (particle.lon + u2 * 0.5 * particle.dt, particle.lat + v2 * 0.5 * particle.dt)
        (u3, v3) = fieldset.UV[time + 0.5 * particle.dt, particle.depth, lat2, lon2, particle]
        lon3, lat3 = (particle.lon + u3 * particle.dt, particle.lat + v3 * particle.dt)
        (u4, v4) = fieldset.UV[time + particle.dt, particle.depth, lat3, lon3, particle]
        particle_dlon += (u1 + 2 * u2 + 2 * u3 + u4) / 6.0 * particle.dt  # noqa
        particle_dlat += (v1 + 2 * v2 + 2 * v3 + v4) / 6.0 * particle.dt  # noqa

        particle.state = StatusCode.Success
```