ERCore User Manual

MetOcean Solutions Ltd.

April 18, 2017
Draft Version

Abstract

This is a user manual. For techincal description see the other document

Contents

1 Introduction

ERCore is a lagrangian model that for every model time step computes positions of a number of particles. These particles can be of different *materials* (passive tracers, oil, plankton, etc..), can be released from different locations and durations, can be moved by different *fields* (currents, tide, wind, etc..) and can be intersected by boundaries like bottom, shoreline, surface elevation. When intersected, particles can be "sticked" to the boundary as for the case of oil in shorelines, or sediments sedimented in the bottom. That is why we call these interesectors *stickers* even though we can choose their degree of stickyness.

1.1 Quickstart

To run the model, you need to define essentially four main blocks:

- `movers`: a list of fields that will advect particles (e.g. currents)
- `diffusers`: a list of fields that will diffuse particles (e.g. diffusion coefficients)
- `stickers`: a list of fields that intercept particle (e.g. shoreline, depth)
- `materials`: a list of releases, where each release must have a particle type, a release origin and a release duration.

The example below (Listing 1) provides an example of a Yaml config file for a buoyant tracer (`class`) release at coordinates `P0`, moving with a rightward current of 1 m/s and downwards (negative) settling velocity of $0.1\ m/s$.

Listing 1: config.yml

```
movers:
    class:  ConstantMover
    id:     cur
    vars:   [uo, vo]
    uo:     1.0
    vo:     0.0
    topo:   bathy

stickers:
    class:  ConstantTopo
    id:     bathy
    vars:   dep
    dep:    -999

diffusers:
    class:  ConstantDiffuser
    id:     diff
    diffx:  0.0
    diffy:  0.0
    diffz:  0.001
    vars:   [diffx, diffy, diffz]

materials:
    class:      BuoyantTracer
    id:         particles
    nbuff:      10000
    movers:     [cur]
    diffusers:  [diff]
    tstart:     '2009-01-01 00:00:00'
    tend:       '2009-01-02 00:00:00'
    P0:         [0,0,0]
    reln:       24
    w0:         -0.1
```

Finally, the model can be run by doing:

```
erall config.yml 20090101_00z 20090102_00z  --disable-geod --dt 3600
```

Note the `-disable-geod` is only here because our test case is not using lat/lon coordinates (section 1.2).

1.2 Coordinate systems

ERCore can use any system of coordinates, provided they are consistent in all input and configuraton data.

If not using lat/lon coordinates, disable geod by instanciating the model with `geod=False` or using `-disable-geod`. It's True by default.

zinvert = True

1.3 Date and time format

Internally, the model uses time in NCEP/CF convention decimal time (matlab time?) which is the "number of days since 1-1-1" and can be computed with:

```
netCDF4.date2num(t0, units='days since 0001-01-01 00:00:00', calendar='standard')
```

or

```
_DT0_=datetime.datetime(2000,1,1)
_NCEPT0_=730120.99999
ncep2dt=lambda t:_DT0_+datetime.timedelta(t-_NCEPT0_)
dt2ncep=lambda t: (1.+t.toordinal()+t.hour/24.+t.minute/1440.+t.second/86400.)
```

Input dates can be either:

- CF decimal time
- datetime python objects, or
- strings like "%Y%m%d_%Hz" or "%Y-%m-%d %H:%M:%S".

1.4 Running backwards in time

ERCore can also run backwards in time, by inverting the release `tstart` and `tend` and using a negative time step:

```
erall config_backwards.yml 20090102_00z 20090101_00z  --disable-geod --dt -3600
```

Note that diffusion is not inverted, it still diffuses particles unless switched off in the release configuration (`diffusers: []`).

2 Materials

2.1 Base configuration

ERCore allows for releases of different particle types, called *materials*. The options for the base class, from which all materials inherit, are listed in table 1.

Table 1: Common options for all materials. *Particle vertical level Z is positive upwards with sea surface = 0, i.e. -10 is 10 m below sea surface. **http://toblerity.org/shapely/manual.html#polygons.

Keyword	Type	Default	Description
id	str		Unique id for release
outfile	str	ercore.$< id >$.out	Filename of output file
P0	[float,float[,float]]	[0,0,0]	Initial position of release $[x, y, z]$*
circular_radius	float		Release particles in a circle shape centered at P0 with radius (in meters)
polygon	[(float,float[,float]), ...]		Release particles in a polygon shape**
movers	list	[]	List of mover id strings
reactors	list	[]	List of reactor id strings
diffusers	list	[]	List of diffuser id strings
stickers	list	[]	List of sticker id strings
unstick	boolean	0	
tstart	datetime/int	0	Starting time for release
tend	datetime/int	1.e10	Ending time for release
nbuff	int		Maximum number of particles (buffer)
reln	int	0	Total number of particles to be released
tstep_release	float		Periodic release of particles (in hours)
R0	float	1.	Total release of material
Q0	float	1.	Flux of material (per day)
spawn	int	1	Number of spawned particles (per day)
maxage	float	1.e20	Particles die when reach this age
is3d	boolean	True	
geod	boolean	False	

nrel is the total number of particles to be released over the current material time interval (e.g. if tstart and tend cover 1 day, and nrel = 24, the model will release 1 particle per hour). For staged or periodic releases, tstep_release can be used (e.g. if 3 hours, the same amount of particles will be released as before, but accumulated every 3 hours).

Each particle can have the following status:

- 0: Not released
- 1: Released and active
- -1: Stuck to shoreline or bottom
- -2: Dead

whereas status 0 and -2 will never appear in the output files.

For each model time step, new particle positions are computed from the *active* pool (status 1) and stored in an array with size (nbuff $\times 3$) where colums are x, y and z coordinates. This buffer array should be big enough to accomodate all particles in the computational pool, but small enough to maintain memory and performance. With that in mind, the model reuses array position of *dead* particles (status -2).

The choice of nbuff is also defined for each material, and should be consistent with the nrel and the simulation characteristics (currents magnitude, stickers, simulation length, etc.) so that it can provide enough buffer size for all the computational pool. If there is no more buffer, a warning will be printed (*Warning: particles exhausted for* $< id >$)

TO DO particle mass here

2.2 Types of origin

Each release can originate from a:

- **Point**: defined by coordinates $[x, y, z]$ in P0,
- **Circle**: shape centered in P0 with radius circular_radius in meters, or
- **Polygon**: defined by polygon keyword as an ordered sequence of $(x, y[, z])$ point tuples, e.g. [(x0,y0), (x1,y1), (x2,y2), ...].

For circular and polygon options, nbuff random points are initialized within the shape.

If necessary, particles release depths are updated according to bathymetry at new locations within the shape. This means that if a particle is located below bathymetry ($z < zbottom$), it's new vertical position will be $z = zbottom + 0.1$

Note that in the Yaml file, the polygon must be specified as in the example below:

```
polygon: [  !!python/tuple [x0,y0],
            !!python/tuple [x1,y1],
            ...
         ]
```

2.3 Types of materials

The following sections describe each material type (or class) presented in Table 2.

Table 2: Material types hierarchy

Class	Parent	Module	Description
PassiveTracer	_Material	.	Basic class passive tracer
BuoyantTracer	PassiveTracer	.	
Drifter	PassiveTracer	.	
BDTracer	BuoyantTracer	.	
Plankton	BuoyantTracer	biota	
Sediment	BuoyantTracer	sediment	
Plume	_Material	plume	Core class for plumes
BuoyantPlume	Plume	plume	Generic buoyant plume class
HydroCarbon	-	hydrocarbons	Base class to hold all chemical processes
HCPlume	BuoyantPlume,HydroCarbon	hydrocarbons	
HCGas	BDTracer	hydrocarbons	
HCDroplets	BDTracer	hydrocarbons	
HCGas	BDTracer	hydrocarbons	
HCSlick	Drifter	hydrocarbons	

2.3.1 PassiveTracer

The most simple material is the inert passive tracer (`class PassiveTracer`), which can only be advected and diffused, without other sinks and sources. `PassiveTracer` particles enter the computational pool by being released, and can leave by either being transported out of the spatial domain or by interception with shoreline or depth ("stickers", see section **??**).

2.3.2 Drifter

A `Drifter` is a passive tracer with

2.3.3 BuoyantTracer

A `BuoyantTracer` is a passive tracer with settling velocity $w0$. Note that positive direction is upwards, so a downward settling velocity will be negative.

2.3.4 BDTracer

A `Drifter` derived class from `BuoyantTracer`