```
#!/usr/bin/env python
# coding: utf-8

# # Particle Tracking Script using OceanParcels to model transport and
connectivity of T. gondii
#
# This is the version of the script includes:
# - base version of the script with particles bouncing perpendicular
to the coast line
# - nesting of the Kaneohe and Oahu South Shore ROMS
#
# by Jennifer Wong-Ala
# Original script put together by Gabi Mukai with aditional code from
Johanna Wren
#
# Date last updated: 12/01/2023
#

# # Load in libraries
# Includes some extra stuff from the tutorial that I didn't use

# In[1]:


import numpy as np
import numpy.ma as ma
from netCDF4 import Dataset
import xarray as xr
import pandas as pd
# import feather
from scipy import interpolate


# In[2]:


from parcels import FieldSet, ParticleSet, JITParticle, ScipyParticle,
AdvectionRK4, DiffusionUniformKh, Variable, Field, GeographicPolar,
Geographic, ErrorCode, plotTrajectoriesFile, NestedField
from datetime import timedelta as delta


# get_ipython().run_line_magic('matplotlib', 'inline')
# import matplotlib.pyplot as plt
# import matplotlib.gridspec as gridspec
# from matplotlib.colors import ListedColormap
# from matplotlib.lines import Line2D
# from copy import copy
# import cmocean
```

```
# # Initial conditions (start & end dates, simulation depth, etc...)

# In[3]:


file_name="test_toots_KANEOHE_NESTED_2019.zarr"

# path2 = '/home/pi/wongalaj/Ciannelli_Lab/wongalaj/TOOTS/
toots_OSS_oocysts_model_2019_03.nc' # give path to where .nc file is
path2 = '/Users/wongalaj/TOOTS/TOOTS_parcels/
test_toots_KANEOHE_NESTED_2019.zarr' # give path to where .nc file is

feather_path = 'test_toots_KANEOHE_NESTED_2019.feather' # assign name
of feather file


# In[4]:


startDate = '2019-01-01' # YYYY-MM-DD
endDate = '2019-01-03' # YYYY-MM-DD


# In[5]:


run_days = 4  # day 2018: 240, 2019: 365, 2020: 366, 2021: 365


# In[6]:


# 2018 runs=  2018-05-05 - 2018-12-31
# 2019 - 2021 runs = 20xx-01-01 - 20xx-12-31


# In[7]:


simDepthMHI = 1


# In[8]:


simDepthKANE = 1


# In[9]:
```

```
# simDepthOSS = 1


# In[10]:


kh = 10   # This is the eddy diffusivity in m2/s


# In[11]:


pld = 3   # in days


# # KANEOHE ROMS

# In[12]:


## U directory

file1_u = '/Users/wongalaj/TOOTS/TOOTS_parcels/ROMS_KANEOHE_Regrid/
2019/u_vel/*.nc'

# file1_u = '/Users/wongalaj/TOOTS/TOOTS_parcels/ROMS_KANEOHE_Regrid/
2018/u_vel/*.nc'
# file1_u = '/Users/wongalaj/TOOTS/TOOTS_parcels/ROMS_KANEOHE_Regrid/
2019/u_vel/*.nc'
# file1_u = '/Users/wongalaj/TOOTS/TOOTS_parcels/ROMS_KANEOHE_Regrid/
2020/u_vel/*.nc'
# file1_u = '/Users/wongalaj/TOOTS/TOOTS_parcels/ROMS_KANEOHE_Regrid/
2021/u_vel/*.nc'

## V directory

file1_v = '/Users/wongalaj/TOOTS/TOOTS_parcels/ROMS_KANEOHE_Regrid/
2019/v_vel/*.nc'

# file1_v = '/Users/wongalaj/TOOTS/TOOTS_parcels/ROMS_KANEOHE_Regrid/
2018/v_vel/*.nc'
# file1_v = '/Users/wongalaj/TOOTS/TOOTS_parcels/ROMS_KANEOHE_Regrid/
2019/v_vel/*.nc'
# file1_v = '/Users/wongalaj/TOOTS/TOOTS_parcels/ROMS_KANEOHE_Regrid/
2020/v_vel/*.nc'
# file1_v = '/Users/wongalaj/TOOTS/TOOTS_parcels/ROMS_KANEOHE_Regrid/
2021/v_vel/*.nc'
```

```
u1 = xr.open_mfdataset(file1_u)
u2 = u1.rename({'OUT_U_TIME': 'u','XAX': 'longitude','YAX':
'latitude', 'DEPTH':'depth', 'TIME':'time'})
# u3 = u2.isel(**{'depth': simDepthKANE}).sel(**{'time':
slice(startDate,endDate)}) # what does this mean? surface depth only
(5m)?


v1 = xr.open_mfdataset(file1_v)
v2 = v1.rename({'OUT_V_TIME': 'v','XAX': 'longitude','YAX':
'latitude', 'DEPTH':'depth', 'TIME':'time'})
# v3 = v2.isel(**{'depth': simDepthKANE}).sel(**{'time':
slice(startDate,endDate)}) # what does this mean? surface depth only
(5m)?

# u3 = u2.sel(**{'time': slice(startDate,endDate)})
# v3 = v2.sel(**{'time': slice(startDate,endDate)})

dimensions = {'lon': 'longitude', 'lat': 'latitude', 'depth': 'depth',
'time': 'time'}


U1=Field.from_xarray(u2.u, 'U', dimensions,
allow_time_extrapolation=True) # , interp_method = {'u': 'freeslip'}
V1=Field.from_xarray(v2.v, 'V', dimensions,
allow_time_extrapolation=True) # , interp_method = {'v': 'freeslip'}

# define u and v for the kaneohe roms
Ufineres = U1
Vfineres = V1

# U1=Field.from_xarray(u2.u, 'u', dimensions,
allow_time_extrapolation=True) # , interp_method = {'u': 'freeslip'}
# V1=Field.from_xarray(v2.v, 'v', dimensions,
allow_time_extrapolation=True) # , interp_method = {'v': 'freeslip'}

# fieldset = FieldSet(U1, V1)



# In[13]:


Ufineres.show() # show the u vel for kaneohe ROMS


# # MHI BASE ROMS

# In[14]:
```

```
file2 = 'https://pae-paha.pacioos.hawaii.edu/erddap/griddap/roms_hiig'
# MHI ROMS (BASE)

ds2 = xr.open_dataset(file2)  # this puts the opendap data into a
xarray dataset

# ds2.load() # https://github.com/pydata/xarray/issues/593 # work
around to prevent IndexError

myDat2 = ds2.isel(**{'depth': simDepthMHI}).sel(**{'time':
slice(startDate,endDate)}) # subset based on time and depth layer

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

U=Field.from_xarray(myDat2.u, 'U', dimensions,
allow_time_extrapolation=True)
V=Field.from_xarray(myDat2.v, 'V', dimensions,
allow_time_extrapolation=True)


# Define U and V for the MHI ROMS
Ucoarse = U
Vcoarse = V


# In[15]:


Ucoarse.show()


# # Nest ROMS together

# In[16]:


UNested = NestedField('U', [Ufineres, Ucoarse])
VNested = NestedField('V', [Vfineres, Vcoarse])

U = UNested
V = VNested

fieldset = FieldSet(U, V)


# # Displacement

# ## Code to make KANEOHE landmask
```

```
# In[17]:


def make_landmask1(fielddata):
    """Returns landmask where land = 1 and ocean = 0
    fielddata is a netcdf file.
    """
    datafile = Dataset(fielddata)

    landmask = datafile.variables['OUT_U_TIME'][0, 0]
    landmask = np.ma.masked_invalid(landmask)
    landmask = landmask.mask.astype('int')

    return landmask


# # Code to make MHI landmask

# In[18]:


def make_landmask2(fielddata):
    """Returns landmask where land = 1 and ocean = 0
    fielddata is a netcdf file.
    """
    datafile = Dataset(fielddata)

    landmask = datafile.variables['u'][0, 0]
    landmask = np.ma.masked_invalid(landmask)
    landmask = landmask.mask.astype('int')

    return landmask


# ## Import velocity field for KANEOHE and MHI ROMS

# In[19]:


# Kaneohe ROMS
file_path1 = '/Users/wongalaj/TOOTS/TOOTS_parcels/ROMS_KANEOHE_Regrid/
2018/u_vel/out_u_timerect_2018_01.nc'

# MHI ROMS
file_path2 = '/Users/wongalaj/TOOTS/TOOTS_parcels/ROMS_MHI/
roms_hiig_2018_05_05_685a_c38d_b3fe.nc'
# file_path2 = '/nfs7/CEOAS/Ciannelli_Lab/wongalaj/TOOTS/
roms_hiig_2018_05_05_685a_c38d_b3fe.nc'
```

```
# ## Make landmask for KANEOHE and MHI ROMS

# #### "land = 1" and "ocean = 0"

# In[20]:


landmask_fineres = make_landmask1(file_path1) # KANEOHE

landmask_coarse = make_landmask2(file_path2) # MHI


# ## Check to make sure landmask was created correctly

# ## Detect the coast
# We can detect the edges between land and ocean nodes by computing
the Laplacian with the 4 nearest neighbors [i+1,j], [i-1,j], [i,j+1]
and [i,j-1]:
#
#
# ∇2landmask=∂xxlandmask+∂yylandmask,
#
# and filtering the positive and negative values. This gives us the
location of coast nodes (ocean nodes next to land) and shore nodes
(land nodes next to the ocean).
#
# Additionally, we can find the nodes that border the coast/shore
diagonally by considering the 8 nearest neighbors, including
[i+1,j+1], [i-1,j+1], [i-1,j+1] and [i-1,j-1].

# In[21]:


def get_coastal_nodes(landmask):
    """Function that detects the coastal nodes, i.e. the ocean nodes
directly
    next to land. Computes the Laplacian of landmask.

    - landmask: the land mask built using `make_landmask`, where land
cell = 1
                and ocean cell = 0.

    Output: 2D array array containing the coastal nodes, the coastal
nodes are
            equal to one, and the rest is zero.
    """
    mask_lap = np.roll(landmask, -1, axis=0) + np.roll(landmask, 1,
axis=0)
    mask_lap += np.roll(landmask, -1, axis=1) + np.roll(landmask, 1,
```

```
axis=1)
    mask_lap -= 4*landmask
    coastal = np.ma.masked_array(landmask, mask_lap > 0)
    coastal = coastal.mask.astype('int')

    return coastal


# In[22]:


def get_shore_nodes(landmask):
    """Function that detects the shore nodes, i.e. the land nodes
directly
    next to the ocean. Computes the Laplacian of landmask.

    - landmask: the land mask built using `make_landmask`, where land
cell = 1
                and ocean cell = 0.

    Output: 2D array array containing the shore nodes, the shore nodes
are
            equal to one, and the rest is zero.
    """
    mask_lap = np.roll(landmask, -1, axis=0) + np.roll(landmask, 1,
axis=0)
    mask_lap += np.roll(landmask, -1, axis=1) + np.roll(landmask, 1,
axis=1)
    mask_lap -= 4*landmask
    shore = np.ma.masked_array(landmask, mask_lap < 0)
    shore = shore.mask.astype('int')

    return shore


# In[23]:


def get_coastal_nodes_diagonal(landmask):
    """Function that detects the coastal nodes, i.e. the ocean nodes
where
    one of the 8 nearest nodes is land. Computes the Laplacian of
landmask
    and the Laplacian of the 45 degree rotated landmask.

    - landmask: the land mask built using `make_landmask`, where land
cell = 1
                and ocean cell = 0.

    Output: 2D array array containing the coastal nodes, the coastal
```

```
nodes are
            equal to one, and the rest is zero.
    """
    mask_lap = np.roll(landmask, -1, axis=0) + np.roll(landmask, 1,
axis=0)
    mask_lap += np.roll(landmask, -1, axis=1) + np.roll(landmask, 1,
axis=1)
    mask_lap += np.roll(landmask, (-1,1), axis=(0,1)) +
np.roll(landmask, (1, 1), axis=(0,1))
    mask_lap += np.roll(landmask, (-1,-1), axis=(0,1)) +
np.roll(landmask, (1, -1), axis=(0,1))
    mask_lap -= 8*landmask
    coastal = np.ma.masked_array(landmask, mask_lap > 0)
    coastal = coastal.mask.astype('int')

    return coastal


# In[24]:


def get_shore_nodes_diagonal(landmask):
    """Function that detects the shore nodes, i.e. the land nodes
where
    one of the 8 nearest nodes is ocean. Computes the Laplacian of
landmask
    and the Laplacian of the 45 degree rotated landmask.

    - landmask: the land mask built using `make_landmask`, where land
cell = 1
                and ocean cell = 0.

    Output: 2D array array containing the shore nodes, the shore nodes
are
            equal to one, and the rest is zero.
    """
    mask_lap = np.roll(landmask, -1, axis=0) + np.roll(landmask, 1,
axis=0)
    mask_lap += np.roll(landmask, -1, axis=1) + np.roll(landmask, 1,
axis=1)
    mask_lap += np.roll(landmask, (-1,1), axis=(0,1)) +
np.roll(landmask, (1, 1), axis=(0,1))
    mask_lap += np.roll(landmask, (-1,-1), axis=(0,1)) +
np.roll(landmask, (1, -1), axis=(0,1))
    mask_lap -= 8*landmask
    shore = np.ma.masked_array(landmask, mask_lap < 0)
    shore = shore.mask.astype('int')

    return shore
```

```
# In[25]:


coastal_fineres = get_coastal_nodes_diagonal(landmask_fineres)
shore_fineres = get_shore_nodes_diagonal(landmask_fineres)

coastal_coarse = get_coastal_nodes_diagonal(landmask_coarse)
shore_coarse = get_shore_nodes_diagonal(landmask_coarse)


# # Assigning coastal velocities
# For the displacement kernel we define a velocity field that pushes
the particles back to the ocean. This velocity is a vector normal to
the shore.
#
# For the shore nodes directly next to the ocean, we can take the
simple derivative of landmask and project the result to the shore
array, this will capture the orientation of the velocity vectors.
#
# For the shore nodes that only have a diagonal component, we need to
take into account the diagonal nodes also and project the vectors only
onto the inside corners that border the ocean diagonally.
#
# Then to make the vectors unitary, we normalize them by their
magnitude.

# In[26]:


def create_displacement_field(landmask, double_cell=False):
    """Function that creates a displacement field 1 m/s away from the
shore.

    - landmask: the land mask dUilt using `make_landmask`.
    - double_cell: Boolean for determining if you want a double cell.
      Default set to False.

    Output: two 2D arrays, one for each camponent of the velocity.
    """
    shore = get_shore_nodes(landmask)
    shore_d = get_shore_nodes_diagonal(landmask) # bordering ocean
directly and diagonally
    shore_c = shore_d - shore                    # corner nodes that
only border ocean diagonally

    Ly = np.roll(landmask, -1, axis=0) - np.roll(landmask, 1, axis=0)
# Simple derivative
    Lx = np.roll(landmask, -1, axis=1) - np.roll(landmask, 1, axis=1)
```

```
    Ly_c = np.roll(landmask, -1, axis=0) - np.roll(landmask, 1,
axis=0)
    Ly_c += np.roll(landmask, (-1,-1), axis=(0,1)) + np.roll(landmask,
(-1,1), axis=(0,1)) # Include y-component of diagonal neighbours
    Ly_c += - np.roll(landmask, (1,-1), axis=(0,1)) -
np.roll(landmask, (1,1), axis=(0,1))

    Lx_c = np.roll(landmask, -1, axis=1) - np.roll(landmask, 1,
axis=1)
    Lx_c += np.roll(landmask, (-1,-1), axis=(1,0)) + np.roll(landmask,
(-1,1), axis=(1,0)) # Include x-component of diagonal neighbours
    Lx_c += - np.roll(landmask, (1,-1), axis=(1,0)) -
np.roll(landmask, (1,1), axis=(1,0))

    v_x = -Lx*(shore)
    v_y = -Ly*(shore)

    v_x_c = -Lx_c*(shore_c)
    v_y_c = -Ly_c*(shore_c)

    v_x = v_x + v_x_c
    v_y = v_y + v_y_c

    magnitude = np.sqrt(v_y**2 + v_x**2)
    # the coastal nodes between land create a problem. Magnitude there
is zero
    # I force it to be 1 to avoid problems when normalizing.
    ny, nx = np.where(magnitude == 0)
    magnitude[ny, nx] = 1

    v_x = v_x/magnitude
    v_y = v_y/magnitude

    return v_x, v_y


# In[27]:


#again make for KANEOHE

v_x_f, v_y_f = create_displacement_field(landmask_fineres)
v_x_c, v_y_c = create_displacement_field(landmask_coarse)


# # Calculate the distance to the shore
# In this tutorial, we will only displace particles that are within
some distance (smaller than the grid size) to the shore.
#
# For this we map the distance of the coastal nodes to the shore:
```

Coastal nodes directly neighboring the shore are 1dx away. Diagonal neighbors are $\sqrt{2}$dx away. The particles can then sample this field and will only be displaced when closer than a threshold value. This gives a crude estimate of the distance.

In[28]:

```
def distance_to_shore(landmask, dx=1):
    """Function that computes the distance to the shore. It is based
in the
    the `get_coastal_nodes` algorithm.

    - landmask: the land mask dUilt using `make_landmask` function.
    - dx: the grid cell dimension. This is a crude approxsimation of
the real
    distance (be careful).

    Output: 2D array containing the distances from shore.
    """
    ci = get_coastal_nodes(landmask) # direct neighbours
    dist = ci*dx                     # 1 dx away

    ci_d = get_coastal_nodes_diagonal(landmask) # diagonal neighbours
    dist_d = (ci_d - ci)*np.sqrt(2*dx**2)       # sqrt(2) dx away

    return dist+dist_d
```

In[29]:

```
d_2_s_f = distance_to_shore(landmask_fineres)
d_2_s_c = distance_to_shore(landmask_coarse)
```

Add displacement

create u and v displacement for coarse and fine

In[30]:

```
u_displacement_f = v_x_f
v_displacement_f = v_y_f

u_displacement_c = v_x_c
v_displacement_c = v_y_c
```

```
# In[31]:


# fieldset for KANEOHE (u and v)

# have to index to choose which field we want to base it off of; U[1]
is choosing coarse ROMS, U[0] = fine ROMS

fieldset.add_field(Field('dispUF', data=u_displacement_f,
                         lon=fieldset.U[0].grid.lon,
lat=fieldset.U[0].grid.lat,
                         mesh='spherical'))

fieldset.add_field(Field('dispVF', data=v_displacement_f,
                         lon=fieldset.V[0].grid.lon,
lat=fieldset.V[0].grid.lat,
                         mesh='spherical'))


# In[32]:


# fieldset for MHI (u and v)

# have to index to choose which field we want to base it off of; U[1]
is choosing coarse ROMS, U[0] = fine ROMS

fieldset.add_field(Field('dispUC', data=u_displacement_c,
                         lon=fieldset.U[1].grid.lon,
lat=fieldset.U[1].grid.lat,
                         mesh='spherical'))

fieldset.add_field(Field('dispVC', data=v_displacement_c,
                          lon=fieldset.V[1].grid.lon,
lat=fieldset.V[1].grid.lat,
                          mesh='spherical'))


# In[33]:


fieldset.dispUF.units = GeographicPolar()
fieldset.dispUC.units = GeographicPolar()
fieldset.dispVF.units = Geographic()
fieldset.dispVC.units = Geographic()


# # Add landmask for KANEOHE and MHI

# In[34]:
```

```
# have to index to choose which field we want to base it off of; U[1]
is choosing MHI (coarse) ROMS, U[0] = Kaneohe (fine) ROMS

fieldset.add_field(Field('landmask_fine', landmask_fineres,
                         lon=fieldset.U[0].grid.lon,
lat=fieldset.U[0].grid.lat,
                         mesh='spherical'))

fieldset.add_field(Field('landmask_coarse', landmask_coarse,
                         lon=fieldset.U[1].grid.lon,
lat=fieldset.U[1].grid.lat,
                         mesh='spherical'))


# # Add distance_to_shore for KANEOHE and MHI

# In[35]:


fieldset.add_field(Field('distance2shore_fine', d_2_s_f,
                         lon=fieldset.U[0].grid.lon,
lat=fieldset.U[0].grid.lat,
                         mesh='spherical'))

fieldset.add_field(Field('distance2shore_coarse', d_2_s_c,
                         lon=fieldset.U[1].grid.lon,
lat=fieldset.U[1].grid.lat,
                         mesh='spherical'))


# # Add eddy diffusivity

# In[36]:


# Add even diffusivity to the fieldset

fieldset.add_constant_field('Kh_zonal', kh, mesh='spherical')
fieldset.add_constant_field('Kh_meridional', kh, mesh='spherical')


# # Add PLD

# In[37]:


# Add PLD to fieldset
```

```
fieldset.add_constant('pld', (pld*86400))


# ## Particle and Kernels
# The distance to shore, used to flag whether a particle must be
displaced, is stored in a particle Variable d2s. To visualize the
displacement, the zonal and meridional displacements are stored in the
variables dU and dV.

# To write the displacement vector to the output before displacing the
particle, the set_displacement kernel is invoked after the advection
kernel. Then only in the next timestep are particles displaced by
displace, before resuming the advection.

# ## Nested model: DisplacementParticle and set_displacement

# In[38]:


class DisplacementParticle(JITParticle):
    dU = Variable('dU')
    dV = Variable('dV')
    d2s = Variable('d2s', initial=1e3)
    age = Variable('age', dtype=np.float32, initial=0.)
    releaseSite = Variable('releaseSite', dtype=np.int32)
    IslandReleaseSite = Variable('IslandReleaseSite', dtype=np.int32)


# In[39]:


def set_displacement(particle, fieldset, time):
    # try saying in between max and min

    if  particle.lon <= fieldset.max_lon and particle.lon >=
fieldset.min_lon and particle.lat <= fieldset.max_lat and particle.lat
>= fieldset.min_lat:
        particle.d2s = fieldset.distance2shore_fine[time,
particle.depth,
                                particle.lat, particle.lon]
        if  particle.d2s < 0.5:
            dispUab = fieldset.dispUF[time, particle.depth,
particle.lat,
                                particle.lon]
            dispVab = fieldset.dispVF[time, particle.depth,
particle.lat,
                                particle.lon]

            particle.dU = dispUab
            particle.dV = dispVab
```

```
        else:
            particle.dU = 0.
            particle.dV = 0.
    else:
        particle.d2s = fieldset.distance2shore_coarse[time,
particle.depth,
                                particle.lat, particle.lon]

        if  particle.d2s < 0.5:
            dispUab = fieldset.dispUC[time, particle.depth,
particle.lat,
                                particle.lon]
            dispVab = fieldset.dispVC[time, particle.depth,
particle.lat,
                                particle.lon]


            particle.dU = dispUab
            particle.dV = dispVab
        else:
            particle.dU = 0.
            particle.dV = 0.


# In[40]:


def displace(particle, fieldset, time):
    if  particle.d2s < 0.5:
        particle.lon += particle.dU*particle.dt
        particle.lat += particle.dV*particle.dt


# ## Delete Particle
# For when it goes past the boundary

# In[41]:


def DeleteParticle(particle, fieldset, time):
    print('deleted particle')
    particle.delete()


# Age Particle
# I want the model to record age at each timestep so it can be used
for analyses over time

# In[42]:
```

```
def Ageing(particle, fieldset, time):
    particle.age += particle.dt
    if particle.age >= fieldset.pld:
        particle.delete()


# # KANEOHE ROMS Boundaries

# In[43]:


# add min lon of base
fieldset.add_constant('min_lon', -158.02)

# add max lon of base
fieldset.add_constant('max_lon', -157.6207)

# add min lat of base
fieldset.add_constant('min_lat', 21.34312)

# add max lat of base
fieldset.add_constant('max_lat', 21.71657)


# # MHI ROMS Boundaries

# In[44]:


# add min lon of base
fieldset.add_constant('min_lon', -163.83070)

# add max lon of base
fieldset.add_constant('max_lon', -152.51930)

# add min lat of base
fieldset.add_constant('min_lat', 17.01843)

# add max lat of base
fieldset.add_constant('max_lat', 23.98238)


# # Simulation
# Now let us test the displacement of the particles as they approach
the shore.

# # Number of particle released per location

# In[45]:
```

```
# constant release number
nrepeat = 1 # how many times do you want locations to repeat (original
= 500)


# In[46]:


# Read in the seeding location file
source_loc = pd.read_csv("all_release_locs_oocysts_final.csv") # read
in file I make


# In[47]:


# source_loc.head


# In[48]:


# subset for just release locations inside of kaneohe ROMS

# kaneohe = source_loc[source_loc["ID"].isin([41,42,43,44,45,29])]
# kane_locs = [28,40,41,42,43,44] # replace range(0,117) with
kane_locs in for loop to release particles only from kaneohe roms
region

# kaneohe <-subset(source_loc, ID %in% c(41,42,43,44,45,29))
# print(kaneohe)


# In[49]:


# kaneohe # what does data look like


# ## Oocysts Model: Create for loop to release number of particles
based on oocysts hydrological model output

# In[50]:


# toots_norm = kaneohe.oocysts_normalized #assign column of
oocysts_normalized to toots-norm
```

```
toots_norm = source_loc.oocysts_normalized #assign column of
oocysts_normalized to toots-norm
# print(toots_norm)


# In[51]:


par_release_prop = nrepeat * toots_norm # multiple nrepeat to the
toots_norm data to get the number of particles that need to be added
in addition to the 500 minimum


# In[52]:


print(toots_norm)


# In[53]:


release_par_final = par_release_prop + nrepeat # add the additional
particles to nrepeat (500) so all locations release a minimum of 500
particles (max of 1000)
# print(release_par_final[21:45])
# print (np.finfo(release_par_final).max) # look at min of toots_norm
# print (np.finfo(par_release_prop).max) # look at min of toots_norm
# print (np.finfo(toots_norm).min) # look at min of toots_norm0


# In[54]:


# create empty np.array to append data too
lon_fin = np.array([])
lat_fin = np.array([])
site_fin = np.array([])
islandsite_fin = np.array([])


# type(lon1)
# print(lon_fin)

# In[55]:


for i in range(0,117):
#     print (i) # what i-th is loop at
```

```
    tmp = release_par_final[i] # get out number of particles to be
released at that specific location using [i]

    lon1 = np.repeat(source_loc.lon[i], tmp) # subset out the
variables I need to be repeated in the pset and replicate it by number
of particles needed
    lat1 = np.repeat(source_loc.lat[i], tmp)
    site1 = np.repeat(source_loc.ID[i], tmp)
    islandsite1 = np.repeat(source_loc.island_release[i], tmp)

    lon_fin = np.append(lon_fin, lon1) # bind all of the initial
conditions data to one final vector for the model psete input
    lat_fin = np.append(lat_fin, lat1)
    site_fin = np.append(site_fin, site1)
    islandsite_fin = np.append(islandsite_fin, islandsite1)


# In[56]:


# Release location from the file read in above
# load in lon and lat from file
#add repeat if applicable

habilon = lon_fin
habilat = lat_fin
habisite = site_fin
islandsite = islandsite_fin


# ## Define the pset and associated variables

# Time interval between particle release (in seconds)

# In[57]:


release_days = 1 # how often to release particles (original == 15
days)


# In[58]:


release_int = release_days*86400 # 15 days converted to seconds


# Start date for release (if you want it different from the first day
of the currents in the fielset)
# start_date = datetime(2000, 1, 16)
```

```
# In[59]:


# Define the pset
pset = ParticleSet.from_list(fieldset=fieldset, #parameter found in
ParticleSet
                             pclass=DisplacementParticle, #parameter
found in ParticleSet
                             lon=habilon, #parameter found in
ParticleSet
                             lat=habilat, #parameter found in
ParticleSet
                             releaseSite=habisite,
                             IslandReleaseSite=islandsite,
                             repeatdt=release_int) #parameter found in
ParticleSet


# In[63]:


print(pset)



# ## Assign kernels, label output file, define run_days

# In[64]:


kernels = pset.Kernel(displace) + pset.Kernel(AdvectionRK4) +
pset.Kernel(DiffusionUniformKh) + pset.Kernel(set_displacement) +
pset.Kernel(Ageing)


# In[65]:


output_file = pset.ParticleFile(name=file_name,
outputdt=delta(hours=4))


# ## Execute model

# In[66]:


# don't print depth
# pset.set_variable_write_status('depth', False)
```

```
# In[67]:


pset.execute(kernels,
             runtime=delta(days=run_days),
             dt=delta(minutes=15),
             recovery={ErrorCode.ErrorOutOfBounds: DeleteParticle},
             output_file=output_file)


# In[68]:


# now stop the repeated release
pset.repeatdt = None


# In[69]:


# now continue running for the remaining length of the PLD
pset.execute(kernels,
             runtime=delta(days=pld+1),
             dt=delta(minutes=15),
             recovery={ErrorCode.ErrorOutOfBounds: DeleteParticle},
             output_file=output_file)


# In[70]:


# output_file.close()
pset


# In[71]:


plt = plotTrajectoriesFile('test_toots_KANEOHE_NESTED_2019.zarr')


# # Convert .nc to .feather file

# In[ ]:


# import pyarrow.feather as feather
```

```
# In[ ]:


# reading zarr file to ncdf


# In[ ]:


# dat_from_zarr = xr.open_zarr(store = file_name)
# dat_from_zarr.to_netcdf(path2)


# In[ ]:


# ds = xr.open_dataset(path2) # load in the .nc file


# In[ ]:


# df = ds.to_dataframe() # convert ds to a dataframe called df


# In[ ]:


# feather.write_feather(df,feather_path) # write df as a feather file
```