```
#!/usr/bin/env python3
# -*- coding: utf-8 -*-
"""
These tools are the main functions for building the ANUGA models
Author: Alexandra Christensen
Created: Thu Jul 23 07:55:18 2020
Updated: 12/01/2022

"""

################################################################################

import subprocess
import rasterio
from osgeo import gdal, ogr
from rasterio._fill import _fillnodata
import numpy as np
import os
from shapely.geometry import Polygon, LineString
#from shapely.ops import cascaded_union
#from shapely.ops import polygonize, unary_union
#from shapely.validation import make_valid

import geopandas as gpd
import warnings; warnings.filterwarnings('ignore', 'GeoSeries.notna',
UserWarning)
import math
from string import Template
import pandas as pd
pd.options.mode.chained_assignment = None
from rasterstats import zonal_stats

import fnmatch
import matplotlib.pyplot as plt
import matplotlib as mpl
from mpl_toolkits.axes_grid1.anchored_artists import (AnchoredSizeBar)
from scipy import ndimage
import scipy
import warnings

#from terracatalogueclient import Catalogue
from skimage.segmentation import felzenszwalb, slic
from skimage.color import label2rgb
import requests
from tqdm.auto import tqdm  # provides a progressbar

import sys
if os.getcwd().split('/')[1] == 'Users':
    code_path = '/users/alchrist/documents/github/ANUGA/processing/code/'
else:
    code_path = '/projects/loac_hydro/alchrist/processing/code'
sys.path.insert(1, code_path)
```

```
from polygon_tools import findconnectedwater, get_nearest,
nearest_neighbor, getpolygonpoints, removenearbypoints, delete_holes
from download_tools import download_NASADEM2
from orinoco_tools import make_distance
from tide_tools import maketides

###############################################################################
###
###############################################################################
###
###############################################################################
###
###############################################################################
###
## PART I
## INPUT: Delta name
## OUTPUT: Working directory
###############################################################################
#

def build_directory(path, delta):
    '''
    Parameters
    ----------
    path : string
        root directory of files (inputs and GEE watermasks).
    delta : string
        AOI.
    '''

print('\n\n\n###############################################################
###################################')
    print('###################################[Step 1][Build
Directory]#########################################')

print('###############################################################################
############################\n')


    try:
        deltapath = [os.path.join(dirpath,f)
                for dirpath,dirnames, files in os.walk(path)
                for f in fnmatch.filter(dirnames,'%s' %(delta))][0] +'/'
    except:
        deltapath = input('The directory %s does not exist, what is the
working directory:') + '/'
    else:
        print('#################### The working directory set as:
\n\n%s\n ' %(deltapath))

    ## Build model directory
    tier1_folders = [deltapath + x for x in
['User_Defined_Files/','tmp/','Setup_Files/','Meshes/','DEMs/','Boundarie
s','Simulations/']]
```

```
    setup_path = tier1_folders[2]
    tier2_folders = [setup_path + x for x
in['Setup_SHP/','Setup_RST/','Setup_FIG/']]
    folders = tier1_folders + tier2_folders
    print('###################### Folders are:')
    print('###################### 0 User_Defined_Files --> User shapefile
of model domain and water mask')
    print('###################### 1 tmp --> For temporary files')
    print('###################### 3 Meshes --> Where we will build model
meshes')
    print('###################### 4 DEMs --> Where we will build digital
elevation models')
    print('###################### 5 Boundaries --> Where we will store
boundary files')
    print('###################### 6 Simulations --> Where we will run
simulations')
    print('###################### 7 Setup_Files/Setup_SHP --> Shapefiles
for setup ')
    print('###################### 8 Setup_Files/Setup_RST --> Rasters for
setup')
    print('###################### 9 Setup_Files/Setup_FIG --> Figures')

    for folder in folders:
        try: os.mkdir(folder)
        except: ''

    print('\n[Step 1][Build Directory] Finished .......\n')
    return deltapath,folders

#########################################################################
###
#########################################################################
###
#########################################################################
###
#########################################################################
###
## PART II
## INPUT: Delta name, working directory, folders
## OUTPUT: extent, bathymetry, topography, hydropolys, landcover, ndwi
watermask
#########################################################################
#
def get_extent_parameters(path,delta,folders,xres,parameters):
    '''
    Parameters
    ----------
    path : string
        root directory of files (inputs and GEE watermasks).
    delta : string
        AOI name, must match input shapefile and folder.
    folders : np.array of strings
        folders within deltapath for model files.
    parameters : pd.dataframe
```

```
        Configuration parameters for model setup.
    '''

print('\n\n\n#################################################################
##################################')
    print('###################################[Step
2][Setup_AOI_Files]#####################################')

print('#################################################################
############################\n')

#################################################################################
#####
    ############################## Get config file
###############################
    ######################### Set configuration parameters
########################

#################################################################################
#####
    # Set model resolution, set methods for building DEM of land and
ocean, and set method for land cover classification
    #yres, xres = res,res
    ref_res = 10
    landmethod      = parameters['LandElevMethod'][0]                #
File or method used for land topography
    oceanmethod     = parameters['OceanElevMethod'][0]               #
File or method used for ocean bathymetry
    landcovermethod = parameters['LandcoverMethod'][0]               #
File or method used for landcover classification

    input_path = path

#################################################################################
#####
    ###################### Get Extent and Coordinate System
######################

#################################################################################
#####
    # Model extent is defined with shapefile with naming format
DELTA_input.shp
    # Must be stored in User_Defined_Files folder within working
directory
    #print('\n[Step 2][Setup_AOI_Files][Get parameters from the
Configuration file] .......\n')
    print('###################### AOI extent is set by: %s_input.shp'
%(delta))

    if os.path.isfile('%s%s_input.shp' %(folders[0],delta)):
        AOI = gpd.read_file('%s%s_input.shp' %(folders[0],delta))
    else:
        inputfile = input('Model extent file %s%s_input.shp does not
exist. What is the full path of model extent file?' %(folders[0], delta))
```

```
        AOI = gpd.read_file(inputfile)

    if AOI.crs is None:
        AOI.crs = 'EPSG:4326'
    if AOI.crs != 'EPSG:4326':
        print("The input shapefile is not in the correction projection
(EPSG 4326), reprojecting to EPSG 4326")
        AOI = AOI.to_crs(4326)


    ## Total bounding coordinates in WGS84 coordinates
    ulx_wgs,lry_wgs,lrx_wgs,uly_wgs = AOI.total_bounds

    ## Determine the UTM coordinate system
    ## ANUGA models are assumed in UTM

    # identify north/south and east/west coordiantes
    x1,y1,x2,y2 =
math.floor(ulx_wgs),math.floor(uly_wgs),math.floor(lrx_wgs),math.floor(lr
y_wgs)

    print('\n[Step 2][Setup_AOI_Files][Determine EPSG code and UTM zone]
.......\n')
    print('###################### ANUGA Models must be in UTM')

    zone = int(np.ceil((ulx_wgs + 180)/6))

    if y1>=0 and y2>=0:
        NS = 'n'
        EPSG = 32200+zone
    elif y1>=0 and y2<0:
        NS = 'n'
        NS2 = 's'
        EPSG = 32200+zone
    else:
        NS = 's'
        y = abs(y1)
        EPSG = 32700+zone
    if x1>=0:
        EW = 'e'
    elif x1<0 and x2>=0:
        EW = 'e'
        EW2 = 'w'
    else:
        EW = 'w'
        x = abs(x1)
    print('###################### UTM Zone: %s%s' %(zone,NS))

    print('###################### EPSG: %s ' %(EPSG))
    AOI_warped = AOI.to_crs('EPSG:%s' %(EPSG))
    AOI_warped.to_file('%s%s_modeldomain.shp' %(folders[7],delta))

    ## Get bounding coordinates in UTM
    xs1,ys1,xs2,ys2 = AOI_warped.total_bounds
```

```
    ## Determine north/south and east/west
    print('\n[Step 2][Setup_AOI_Files][Extending AOI by 1000m]
.......\n')
    ulx = int(min(xs1,xs2) - 1000)
    uly = int(max(ys1,ys2) + 1000)
    lrx = int(max(xs1,xs2) + 1000)
    lry = int(min(ys1,ys2) - 1000)
    if (lrx-ulx)%xres !=0:
        lrx = int(lrx - (lrx-ulx)%xres)
    if (uly-lry)%xres !=0:
        uly = int(uly - (uly-lry)%xres)

    # Model domain extent
    print('\n[Step 2][Setup_AOI_Files][Setting up AOI extent] .......\n')
    extent = [(ulx),(uly)],[(lrx),(uly)],[(lrx),(lry)],[(ulx),(lry)]
    poly = Polygon([[p[0], p[1]] for p in extent])
    boundingbox =
gpd.GeoDataFrame(index=[0],crs='EPSG:%s'%(EPSG),geometry=[poly])

    np.savetxt('%s%s_extent.csv' %(folders[0],delta),
extent,delimiter=',', fmt= '%1.3f') ## USED BY ANUGA MODEL
    boundingbox.to_file("%s%s_extent_%s.shp" %(folders[7],delta,EPSG))
    extentarea = np.round(boundingbox.area[0],-6)
    print('###################### AOI bounds are : %s, %s, %s, %s'
%(round(ulx,-1),round(lry,-1),round(lrx,-1),round(uly,-1)))
    print('###################### Approximate area of AOI extent is %s
km^2' %(extentarea/1000000))


#########################################################################
#####
    ############################## Baseline Datasets
############################

#########################################################################
#####


#########################################################################
#####
    ############# GEBCO (Bathymetry, relative to mean sea level)
################

#########################################################################
#####
    # GEBCO file will be reference for projection, origin, resolution
    # Get GEBCO data within the model domain
    print('\n[Step 2][Setup_AOI_Files][Downloading GEBCO Dataset as
reference for projection, resolution, etc] .......\n')
    try:
        ref_10m = rasterio.open('%s%s_GEBCO_%s.tif'
%(folders[8],delta,ref_res))
    except:
```

```
        os.system('gdalwarp -overwrite -tr %s %s
%sgebco_2020_geotiff/gebco_all.vrt %s%s_GEBCO_%s.tif '\
                  '-t_srs EPSG:%s -te %s %s %s %s -srcnodata -9999 -
dstnodata -9999 -co COMPRESS=DEFLATE -q'

%(ref_res,ref_res,input_path,folders[8],delta,ref_res,EPSG,ulx,lry,lrx,ul
y))
        ref_10m = rasterio.open('%s%s_GEBCO_%s.tif'
%(folders[8],delta,ref_res))
    save_profile_10m = ref_10m.profile
    parameters['ulx'] = ulx
    parameters['lry'] = lry
    parameters['lrx'] = lrx
    parameters['uly'] = uly
    parameters['EPSG'] = EPSG
    parameters.to_csv('%s/%s_Configuration.csv' %(folders[2],delta))

    print('\n[Step 2][Setup_AOI_Files][Saving configuration file]
.......\n')
    print('###################### Saved as %s%s_Configuration.csv'
%(folders[2].split(delta)[-1],delta))
    return ref_10m,parameters

def setup_AOI_files(path,delta,folders,xres,parameters):
    '''
    Parameters
    ----------
    path : string
        root directory of files (inputs and GEE watermasks).
    delta : string
        AOI name, must match input shapefile and folder.
    folders : np.array of strings
        folders within deltapath for model files.
    parameters : pd.dataframe
        Configuration parameters for model setup.
    '''

print('\n\n\n##################################################################
###################################')
    print('###################################[Step
2][Setup_AOI_Files]#####################################')

print('##################################################################
############################\n')

    EPSG = parameters['EPSG'][0]
# Coordinate System must be UTM
    ulx  = parameters['ulx'][0]
# ULX coordinate
    lry  = parameters['lry'][0]
# LRY coordinate
    lrx  = parameters['lrx'][0]
# LRX coordinate
    uly  = parameters['uly'][0]
```

```
    #extentpoly = gpd.read_file("%s%s_extent_%s.shp"
%(folders[7],delta,EPSG))
    ref_res = 10
    landmethod      = parameters['LandElevMethod'][0]                #
File or method used for land topography
    oceanmethod     = parameters['OceanElevMethod'][0]               #
File or method used for ocean bathymetry
    landcovermethod = parameters['LandcoverMethod'][0]               #
File or method used for landcover classification
    if os.path.isfile('%s%s_input.shp' %(folders[0],delta)):
        AOI = gpd.read_file('%s%s_input.shp' %(folders[0],delta))
    else:
        inputfile = input('Model extent file %s%s_input.shp does not
exist. What is the full path of model extent file?' %(folders[0], delta))
        AOI = gpd.read_file(inputfile)

    if AOI.crs is None:
        AOI.crs = 'EPSG:4326'
    if AOI.crs != 'EPSG:4326':
        print("The input shapefile is not in the correction projection
(EPSG 4326), reprojecting to EPSG 4326")
        AOI = AOI.to_crs(4326)


    ## Total bounding coordinates in WGS84 coordinates
    ulx_wgs,lry_wgs,lrx_wgs,uly_wgs = AOI.total_bounds
    x1,y1,x2,y2 =
math.floor(ulx_wgs),math.floor(uly_wgs),math.floor(lrx_wgs),math.floor(lr
y_wgs)
    zone = int(np.ceil((ulx_wgs + 180)/6))

    ref_10m = rasterio.open('%s%s_GEBCO_%s.tif'
%(folders[8],delta,ref_res))
    save_profile_10m = ref_10m.profile

    ## Set profile settings for saving all rasters
    if os.path.isfile('%s%s_GEBCO_%s.tif' %(folders[8],delta,xres))
==False:
        os.system('gdalwarp -overwrite -tr %s %s %s%s_GEBCO_%s.tif
%s%s_GEBCO_%s.tif '\
                  ' -te %s %s %s %s -srcnodata -9999 -dstnodata -9999 -co
COMPRESS=DEFLATE -q'

%(xres,xres,folders[8],delta,ref_res,folders[8],delta,xres,ulx,lry,lrx,ul
y))

    print('###################### Reference raster is %s_GEBCO_%s.tif'
%(delta,xres))
    print('###################### This profile will be used for all future
warping')
    print('###################### Resolution is %sm' %(xres))

    ref = rasterio.open('%s%s_GEBCO_%s.tif' %(folders[8],delta,xres))
    save_profile_xres = ref.profile
```

```
    save_profile_xres['dtype'] = 'float64'
    save_profile_xres['compress'] = 'deflate'
    affine = save_profile_xres['transform']
    ref_array = ref.read(1)


#########################################################################
#####
    ############################## Ocean Bathymetry
###############################

#########################################################################
#####
    print('\n[Step 2][Setup_AOI_Files][Compiling OCEAN elevation dataset]
.......\n')
    print('###################### Ocean elevation set with:  %s'
%(oceanmethod))
    if os.path.isfile("%s%s_bathy_%s.tif"
%(folders[8],delta,xres))==False:
        if oceanmethod == 'default' or oceanmethod == 'GEBCO':
            bathy_1 = np.where(ref_array<=-9999,0,np.where(ref_array<-
0.9,ref_array,0))
            bathy = ndimage.gaussian_filter(bathy_1, sigma=(15, 15),
order=0)

        elif oceanmethod[:8] == 'constant':
            constantOceanElev = float(oceanmethod[9:])
            bathy = np.where(ref_array.isnan(),-9999,constantOceanElev)

        else:
            if os.path.isfile(folders[0] + oceanmethod + '.tif') ==
False:
                print('File not found \n')
                oceanmethod = input('What file would you like to use?
Should be saved in /User_Defined_Files subfolder ')
            os.system('gdalwarp -overwrite -tr %s %s %s%s.tif
%s%s_ocean.tif '\
                      '-t_srs EPSG:%s -srcnodata -9999 -dstnodata -9999
-co COMPRESS=DEFLATE -q'
                      %(xres,
xres,folders[0],oceanmethod,folders[8],oceanmethod,EPSG,ulx,lry,lrx,uly))
            bathy = gdal.Open("%s%s_ocean.tif"
%(folders[8],oceanmethod)).ReadAsArray()

        with rasterio.open("%s%s_bathy_%s.tif"
%(folders[8],delta,xres),'w', **save_profile_xres) as dst:
            dst.write_band(1,bathy.astype('float64'))
        print('###################### Saving Ocean Bathymetry file as
%s%s_bathy_%s.tif' %(folders[8].split(delta)[-1],delta,xres))


#########################################################################
#####
```

```
    ############################## Land Topography
################################

#########################################################################
#####
    print('\n[Step 2][Setup_AOI_Files][Compiling LAND elevation dataset]
.......\n')
    print('#################### Land elevation set with: %s'
%(landmethod))
    if os.path.isfile("%s%s_topo_%s.tif" %(folders[8],delta + '_'
+landmethod,xres))==False:
        if landmethod == 'default' or landmethod == 'NASADEM':
            print('#################### Downloading appropriate NASADEM
tiles and merging into VRT')
            if not os.path.isfile('%s%s_NASADEM_topo_%s.tif'
%(folders[8],delta,xres)):
                if not os.path.isfile('%s%s_NASADEM_egm08.tif'
%(folders[8],delta)):

download_NASADEM2('NASADEM',delta,'',folders[1:],x1,x2,y1,y2,zone,ulx,lry
,lrx,uly,ref_res,ref_res)
                    print('#################### Convert from native
EGM96 to EGM08 (EPSG 3855)')
                    os.system('gdalwarp -overwrite -srcnodata -9999 -
dstnodata -9999 -wt Float64 -ot Float64 '\
                              '-t_srs "+proj=longlat +datum=WGS84
+no_defs +geoidgrids=%segm08_25.gtx" '\
                              '-s_srs "+proj=longlat +datum=WGS84
+no_defs +geoidgrids=%segm96_15.gtx" '\
                              '-co COMPRESS=DEFLATE '\
                              '%s%s_NASADEM.vrt %s%s_NASADEM_egm08.tif -
co COMPRESS=DEFLATE -q'

%(input_path,input_path,folders[1],delta,folders[8],delta))
                print('#################### Convert from native WGS84
(EPSG 4326) to UTM (EPSG %s)' %(EPSG))
                os.system('gdalwarp -overwrite -tr %s %s -tap -te %s %s
%s %s -srcnodata -9999 -dstnodata -9999 '\
                          '-s_srs EPSG:4326 -t_srs EPSG:%s %s%s_egm08.tif
%s%s_topo_%s.tif -co COMPRESS=DEFLATE -q'

%(xres,xres,ulx,lry,lrx,uly,EPSG,folders[8],delta+'_'+landmethod,folders[
8],delta+'_'+landmethod,xres))
        elif landmethod == 'GLO30':
            if not os.path.isfile('%s%s_topo_%s.tif'
%(folders[8],delta+'_'+landmethod,xres)):
                if not os.path.isfile('%s%s_egm08.tif'
%(folders[8],delta+'_'+landmethod)):
                    print('#################### Downloading Tandem-X
GLO30')
                    ## Download appropriate TanDEM X Glo 30 files and
merge into VRT
```

```
download_NASADEM2('GLO30',delta,'',folders[1:],x1,x2,y1,y2,zone,ulx,lry,l
rx,uly,ref_res,ref_res)
                    print('##################### Already in EGM08 (EPSG
3855)')
                    os.system('gdalwarp -overwrite -srcnodata -9999 -
dstnodata -9999 -wt Float64 -ot Float64 '\
                              '%s%s_GLO30.vrt %s%s_egm08.tif -co
COMPRESS=DEFLATE -q'

%(folders[1],delta,folders[8],delta+'_'+landmethod))
                print('##################### Convert from native WGS84
(EPSG 4326) to UTM (EPSG %s)' %(EPSG))
                os.system('gdalwarp -overwrite -tr %s %s -te %s %s %s %s
-srcnodata -9999 -dstnodata -9999 '\
                          '-s_srs EPSG:4326 -t_srs EPSG:%s %s%s_egm08.tif
%s%s_topo_%s.tif -co COMPRESS=DEFLATE -q'

%(xres,xres,ulx,lry,lrx,uly,EPSG,folders[8],delta+'_'+landmethod,folders[
8],delta+'_'+landmethod,xres))

        elif landmethod[:8] == 'constant':
            constantLandElev = float(landmethod[9:])
            print('##################### Constate land elevation is set
to %s' %(constantLandElev))
            topo = np.where(ref_array.isnan(),0,constantLandElev)
            with rasterio.open("%s%s_topo_%s.tif" %(folders[8],delta +
'_' + constantLandElev,xres),'w', **save_profile_xres) as dst:
                dst.write_band(1,topo.astype('float64'))
        else:
            if os.path.isfile(folders[0] + landmethod) == False:
                print('File not found \n')
                landmethod = input('What file would you like to use?
Should be saved in /Users_Defined_Files subfolder ')
            os.system('gdalwarp -overwrite -tr %s %s -tap %s%s%s.tif
%s%s_%s_topo_%s.tif -t_srs EPSG:%s -te %s %s %s %s '\
                      '-srcnodata -9999 -dstnodata -9999 -co
COMPRESS=DEFLATE -q'
                      %(xres,
xres,folders[0],delta,landmethod,folders[8],delta,landmethod,xres,EPSG,ul
x,lry,lrx,uly))
        print('##################### Saving Land Topography file as
%s%s_topo_%s.tif' %(folders[8].split(delta)[-1],delta,xres))


#########################################################################
#####
    ###################### Landcover Classification Map
#########################

#########################################################################
#####
    print('\n[Step 2][Setup_AOI_Files][Compiling Landcover Classification
Maps] .......\n')
```

```
    print('##################### Landcover types based on: %s'
%(landcovermethod))
    if os.path.isfile('%s%s_%s_%s.tif'
%(folders[8],delta,landcovermethod,xres))==False:
        if landcovermethod == 'default' or landcovermethod ==
'WorldCover':
            print('##################### Landcover types based on
Sentinel 1 and 2 World Cover Maps')
            if not os.path.isfile('%s%s_%s_%s.tif'
%(folders[8],delta,landcovermethod,xres)):
                print('##################### Downloading new sentinel 1
and 2 landcover map')
                source = 'WorldCover'
                bounds = (ulx_wgs-1, lry_wgs-1, lrx_wgs+1, uly_wgs+1)
                geometry = Polygon.from_bounds(*bounds)

                ##Use AWS cloud data
                s3_url_prefix = "https://esa-worldcover.s3.eu-central-
1.amazonaws.com"
                # load worldcover grid
                url =
f'{s3_url_prefix}/v100/2020/esa_worldcover_2020_grid.geojson'
                grid = gpd.read_file(url)
                # get grid tiles intersecting AOI
                tiles = grid[grid.intersects(geometry)]
                for tile in tqdm(tiles.ll_tile):
                    out_fn = folders[1] +
f"ESA_WorldCover_10m_2020_v100_{tile}_Map.tif"
                    if os.path.isfile(out_fn)==False:
                        url =
f"{s3_url_prefix}/v100/2020/map/ESA_WorldCover_10m_2020_v100_{tile}_Map.t
if"
                        r = requests.get(url, allow_redirects=True)
                        with open(out_fn, 'wb') as f:
                            f.write(r.content)
                ### Authenticate to the Terrascope platform (registration
required)
                ### create catalogue object and authenticate
interactively with a browser
                # catalogue = Catalogue().authenticate()
                # products =
catalogue.get_products("urn:eop:VITO:ESA_WorldCover_10m_2020_V1",
geometry=geometry)
                # catalogue.download_products(products, folders[1],force
= True)
                landcovers = [os.path.join(dirpath,f)
                    for dirpath,dirnames, files in os.walk(folders[1])
                    for f in fnmatch.filter(files,'*_Map.tif')]
                with open("%s%s_%s.txt" %(folders[1],delta,source), 'w')
as f:
                    for item in landcovers:
                        f.write("%s\n" % item)
                # Merge NASADEM tiles to make topography file
```

```
                print('#################### Merging landcover tiles')
#EPSG:4326++5733
                os.system("gdalbuildvrt %s%s_%s.vrt -input_file_list
%s%s_%s.txt -a_srs EPSG:4326 "

%(folders[1],delta,landcovermethod,folders[1],delta,landcovermethod))
                os.system('gdalwarp -overwrite %s%s_%s.vrt %s%s_%s_%s.tif
'\
                          '-t_srs EPSG:%s -te %s %s %s %s -ts %s %s -co
COMPRESS=DEFLATE -q'

%(folders[1],delta,landcovermethod,folders[8],delta,landcovermethod,ref_r
es,EPSG,ulx,lry,lrx,uly,save_profile_10m['width'],save_profile_10m['heigh
t']))
                os.system('gdalwarp -overwrite %s%s_%s.vrt %s%s_%s_%s.tif
'\
                          '-t_srs EPSG:%s -te %s %s %s %s -ts %s %s -co
COMPRESS=DEFLATE -q'

%(folders[1],delta,landcovermethod,folders[8],delta,landcovermethod,xres,
EPSG,ulx,lry,lrx,uly,save_profile_xres['width'],save_profile_xres['height
']))
        elif landcovermethod == 'Copernicus':
            print('#################### Landcover types based on Landsat
Copernicus Landcover Maps')
            if not os.path.isfile('%s%s_%s_10.tif'
%(folders[8],delta,landcovermethod)):
                print('#################### using landcover
classification from Daniel, made using Copernicus Land Cover maps based
on Landsat')
                landcovers = [os.path.join(dirpath,f)
                    for dirpath,dirnames, files in
os.walk(os.path.dirname(os.path.dirname(os.path.dirname(os.path.dirname(f
olders[0])))))+'/Landcover')
                    for f in fnmatch.filter(files,'*%s*.tif'
%(delta.capitalize()))]
                landcover = landcovers[0]
                prj =
ogr.SpatialReference(wkt=gdal.Open(landcover).GetProjection()).GetAttrVal
ue('AUTHORITY',1)
                if prj != '4326':
                    os.system('gdalwarp -overwrite %s %s_4326.tif -t_srs
EPSG:4326 -te %s %s %s %s -srcnodata -9999 -dstnodata 0'
                                %(landcovers[0][:-
4]+'.tif',landcovers[0][:-4],ulx_wgs,lry_wgs,lrx_wgs,uly_wgs))
                    #gdal.Warp(landcovers[0][:-
4]+'_4326.tif',gdal.Open(landcover),dstSRS='EPSG:4326')
                    landcover = landcovers[0][:-4]+'_4326.tif'
                os.system('gdalwarp -overwrite -tr %s %s %s
%s%s_%s_%s.tif -t_srs EPSG:%s -te %s %s %s %s '\
                          '-srcnodata -9999 -dstnodata 0 -co
COMPRESS=DEFLATE'
```

```
%(ref_res,ref_res,landcover,folders[8],delta,landcovermethod,ref_res,EPSG
,ulx,lry,lrx,uly))
                os.system('gdalwarp -overwrite -tr %s %s %s
%s%s_%s_%s.tif -t_srs EPSG:%s -te %s %s %s %s '\
                          '-srcnodata -9999 -dstnodata 0 -co
COMPRESS=DEFLATE'

%(xres,xres,landcover,folders[8],delta,landcovermethod,xres,EPSG,ulx,lry,
lrx,uly))

    print('\n[Step 2][Setup_AOI_Files][Loading Global Mangrove Watch ]
.......\n')
    if os.path.isfile('%s%s_GMW_%s.tif' %(folders[8],delta,xres))==False:
        try:
            mangroves = gpd.read_file(input_path +
"GMW_2016_v2_fixed.shp")
            mangrove =
gpd.overlay(mangroves,model_domain.to_crs('EPSG:4326'),how='intersection'
)
            mangrove = mangrove.to_crs('EPSG:%s' %(EPSG))
            mangrove.to_file("%s%s_GMW.shp" %(folders[1],delta))     #
Save
        except:
            print('There are no mangroves in the %s domain' %(delta))
            mangrove = np.where(ref_10m.read(1),0.,0.)
            with rasterio.open('%s%s_GMW_10.tif' %(folders[8],delta),'w',
**save_profile_10m) as dst:
                dst.write_band(1,mangrove)
            mangrove = np.where(ref_array,0.,0.)
            with rasterio.open('%s%s_GMW_%s.tif'
%(folders[8],delta,xres),'w', **save_profile_xres) as dst:
                dst.write_band(1,mangrove)
        else:
            print('Saving mangrove file as %s%s_GMW_10.tif'
%(folders[8],delta))
            os.system('gdal_rasterize -burn 1 -tr %s %s -tap -te %s %s %s
%s %s%s_GMW.shp %s%s_GMW_%s.tif '\
                      '-a_nodata -9999 -co COMPRESS=DEFLATE'

%(ref_res,ref_res,ulx,lry,lrx,uly,folders[1],delta,folders[8],delta,ref_r
es))
            os.system('gdal_rasterize -burn 1 -tr %s %s -tap -te %s %s %s
%s %s%s_GMW.shp %s%s_GMW_%s.tif '\
                      '-a_nodata -9999 -co COMPRESS=DEFLATE'

%(xres,xres,ulx,lry,lrx,uly,folders[1],delta,folders[8],delta,xres))
    if landcovermethod == 'Copernicus':
        mangrove_class = 9
        wetland_class = 10
        crop_class = 6
    elif landcovermethod == 'WorldCover':
        mangrove_class = 95
        wetland_class = 90
```

```
        crop_class = 40
        water_class = 80

    landcover = rasterio.open('%s%s_%s_10.tif'
%(folders[8],delta,landcovermethod)).read(1).astype('int')
    mangrove = gdal.Open('%s%s_GMW_10.tif'
%(folders[8],delta)).ReadAsArray()

    landcover[landcover==-9999] = 0
    landcover = np.where(mangrove==1,mangrove_class,landcover)

    print('\n[Step 3A][Make_Watermask][Delineating wetland areas]
......\n')
    if os.path.isfile("%s%s_wetlands_%s.tif"
%(folders[8],delta,xres))==False:
        if os.path.isfile("%s%s_wetlands_10.tif"
%(folders[8],delta))==False:
            iswetland =
np.where(mangrove==1,1.,np.where((landcover==mangrove_class) |
(landcover==wetland_class),1.,-9999))
            iswetland = np.where(iswetland==1,1.,0.)
            with rasterio.open("%s%s_wetlands_10.tif"
%(folders[8],delta),'w', **save_profile_10m) as dst:
                dst.write_band(1,iswetland)
        os.system('gdalwarp -overwrite -tr %s %s %s%s_wetlands_10.tif
%s%s_wetlands_%s.tif -te %s %s %s %s '\
                  '-srcnodata -9999 -dstnodata -9999 -co
COMPRESS=DEFLATE'

%(xres,xres,folders[8],delta,folders[8],delta,xres,ulx,lry,lrx,uly))
    iswetland = rasterio.open("%s%s_wetlands_10.tif"
%(folders[8],delta)).read(1)

    print('\n[Step 3A][Make_Watermask][Delineating agriculture areas]
......\n')
    if os.path.isfile("%s%s_crops_%s.tif"
%(folders[8],delta,xres))==False:
        if os.path.isfile("%s%s_crops_10.tif"
%(folders[8],delta))==False:
            iscrop = np.where(landcover==crop_class,1.,-9999)
            iscrop = np.where(iscrop==1,1.,0.)
            with rasterio.open("%s%s_crops_10.tif"
%(folders[8],delta),'w', **save_profile_10m) as dst:
                dst.write_band(1,iswetland)
        os.system('gdalwarp -overwrite -tr %s %s %s%s_crops_10.tif
%s%s_crops_%s.tif -te %s %s %s %s -'\
                  'srcnodata -9999 -dstnodata -9999 -co COMPRESS=DEFLATE'

%(xres,xres,folders[8],delta,folders[8],delta,xres,ulx,lry,lrx,uly))
    iscrop = rasterio.open("%s%s_crops_10.tif"
%(folders[8],delta)).read(1)
```

```
    print('\n[Step 2][Setup_AOI_Files] Finished .......\n')

        # ref = rasterio.open('%s%s_GEBCO_%s.tif'
%(folders[8],delta,xres))
        # gee_ndwi_path = path + 'GEE_NDWI_watermask'
#'/Volumes/GoogleDrive/My Drive/GEE_drive/'
        # gee = [os.path.join(dirpath,f)
        #     for dirpath,dirnames, files in os.walk(gee_ndwi_path)
        #     for f in fnmatch.filter(files,'%s*' %(delta))]
        # watermaskname = gee[0].split('/')[-1].split('.')[0].split('-
')[0]
        # print('\n[Step 2][Setup_AOI_Files] SKIP.......\n')

    return ref

##########################################################################
###
##########################################################################
###
##########################################################################
###
## PART IIIa
## OUTPUT: landcover, water types, DEM
##########################################################################
#
def
make_watermask(path_ancillary,delta,folders,parameters,ref,clean_with_lan
dcover,skip=False):
    '''
    Parameters
    ----------
    delta : string
        AOI name, must match input shapefile and folder.
    folders : np.array of strings
        folders within deltapath for model files.
    parameters : pd.dataframe
        Configuration parameters for model setup.
    ref : open DatasetReader (rasterio)
        The reference file used to detemine resolution, width, height,
extent of all rasters
    watermaskname : string
        Name of the best water mask from GEE
    skip : boolean
        If True, do not build water polygons.
    '''


print('\n\n\n##################################################################
#################################')
    print('#################################[Step
3A][Make_Watermask]#################################')
```

```
print('#########################################################################
############################\n')
    save_profile_xres = ref.profile
    save_profile_xres['compress'] = 'deflate'
    xres,yres = int(save_profile_xres['transform'][0]),-
int(save_profile_xres['transform'][4])
    EPSG = parameters['EPSG'][0]
# Coordinate System must be UTM
    ulx  = parameters['ulx'][0]
# ULX coordinate
    lry  = parameters['lry'][0]
# LRY coordinate
    lrx  = parameters['lrx'][0]
# LRX coordinate
    uly  = parameters['uly'][0]
    extentpoly = gpd.read_file("%s%s_extent_%s.shp"
%(folders[7],delta,EPSG))
    landcovermethod = parameters['LandcoverMethod'][0]               #
File or method used for landcover classification
    wetland_class = int(parameters['WetlandClass'][0])      # Class #
representing wetlands in landcover file


#########################################################################
#####
    ###################### Get best watermask from GEE
############################

#########################################################################
#####
    print('\n[Step 2][Setup_AOI_Files][Compiling Water Masks] .......\n')
    # gee_ndwi_path = path + 'GEE_NDWI_watermask'
#'/Volumes/GoogleDrive/My Drive/GEE_drive/'
    # gee = [os.path.join(dirpath,f)
    #     for dirpath,dirnames, files in os.walk(gee_ndwi_path)
    #     for f in fnmatch.filter(files,'%s*' %(delta))]
    # if len(gee)<=0:
    #     print('There are no GEE watermasks')
    gee = [os.path.join(dirpath,f)
        for dirpath,dirnames, files in os.walk(folders[0])
        for f in fnmatch.filter(files,'%s*_finalwatermask.tif' %(delta))]

        #user_input = input('Did you put it there? If yes, we can proceed
if you enter "y". If not, the remaining code will fail ')
    watermaskname = gee[0].split('/')[-1].split('.')[0].split('-')[0]

    if skip==False:

        print('###################### Google Earth Engine water mask file
: %s.tif' %(watermaskname))
        if os.path.isfile('%s%s_10b.tif'
%(folders[8],watermaskname))==False:
            with open("%sgee_ndwi_files.txt" %(folders[1]), 'w') as f:
```

```
                for item in gee:
                    f.write("%s\n" % item)
            os.system('gdalbuildvrt -input_file_list
%s/gee_ndwi_files.txt -vrtnodata -9999 -srcnodata -9999 '\
                      '-a_srs EPSG:4326 %s%s.vrt '
                      %(folders[1],folders[1],watermaskname))
            os.system('gdalwarp -overwrite -tr %s %s %s%s.vrt %s%s_10.tif
-s_srs EPSG:4326 -t_srs EPSG:%s -te %s %s %s %s '\
                      '-srcnodata -9999 -dstnodata -9999 -co
COMPRESS=DEFLATE'
                      %(10,10,
folders[1],watermaskname,folders[8],watermaskname,EPSG,ulx,lry,lrx,uly))



        def fix_connectivity(structure,input_mask,origin):
            full_watermask2 =
ndimage.morphology.binary_hit_or_miss(input_mask, structure1 =
structure.astype(input_mask.dtype),origin1 =origin)
            full_watermask3 = np.where(full_watermask2==True,1,0)
            full_watermask = input_mask + full_watermask3
            # structure_flip = np.flip(structure,flip_dir)
            full_watermask = np.where(full_watermask>0,1.,0.)
            return full_watermask


##########################################################################
##
        ########################## Import Hydropolys
#############################

##########################################################################
##
        ## Import hydropolys (UCLA) shapefile to define rivers and oceans
        ## Hydrolakes do not match perfectly with hydropolys - need to
remove small artifacts
        print('\n[Step 3A][Make_Watermask][Load Hydropolys] .......\n')
        try: hydropoly = gpd.read_file("%s%s_hydropolys.shp"
%(folders[0],delta))
        except:
            print('#################### Extracting UCLA Hydropolys for
the AOI')
            hydropolys = gpd.read_file("%sucla/hydropolys_fix.shp"
%(path_ancillary))
            hydropolys_crs = hydropolys.crs
            extentpoly2 = extentpoly.buffer(xres*10)
# Create slightly larged extent for clipping in 4326 crs
            extentpoly2 =  extentpoly2.to_crs(hydropolys_crs)
# Reproject to 4326 CRS
            hydropoly = gpd.clip(hydropolys,extentpoly2)
# Clip hydropolys to model domain extent
            hydropoly = hydropoly.to_crs("EPSG:%s" %(EPSG))
# Reproject to UTM
```

```
            hydropoly =
gpd.overlay(hydropoly,extentpoly,how='intersection')        # Clip to
extent in correct EPSG, ensure clipped area is correct
            hydropoly.to_file("%s%s_hydropolys.shp" %(folders[0],delta))
# Save

        print('\n[Step 3A][Make_Watermask][Load SWOT PLD lakes]
.......\n')
        try: thelake = gpd.read_file("%s%s_SWOTPLD.shp"
%(folders[0],delta))
        except:
            print('#################### Extracting SWOT PLD Lakes for
the AOI')
            try:
                plds = gpd.read_file('%sSWOT_PLD.gdb/SWOT_PLD.shp'
%(path_ancillary))
            except:''
            else:
                hydrolakes_crs = plds.crs
                extentpoly2 = extentpoly.buffer(xres)
# Create slightly larged extent for clipping in 4326 crs
                extentpoly2 =  extentpoly2.to_crs(hydrolakes_crs)
# Reproject to 4326 CRS
                thelake = gpd.clip(plds,extentpoly2)
# Clip HydroLakes to model domain extent
                thelake = thelake.to_crs("EPSG:%s" %(EPSG))
# Save
                if len(thelake) == 0:
                    print('There are no lakes in the model domain')
                    d = {'geometry': [Polygon([(0, 0), (0,0),(0,0)])]}
                    thelake = gpd.GeoDataFrame(d,crs="EPSG:%s" %(EPSG))
                else:
                    thelake['TYPE_2'] = 'Lake'
# Add label to identify lakes
                    thelake =
gpd.overlay(thelake,extentpoly,how='intersection')        # Clip to
extent in correct EPSG, ensure clipped area is correct
                thelake.to_file("%s%s_SWOTPLD.shp" %(folders[0],delta))

        # print('\n[Step 3A][Make_Watermask][Load HydroLAKES] .......\n')
        # try: thelake = gpd.read_file("%s%s_HydroLAKES.shp"
%(folders[0],delta))
        # except:
        #     hydrolakes = gpd.read_file(path_ancillary +
'/HydroSheds/HydroLAKES_polys_v10_shp/HydroLAKES_polys_v10_fixed.shp')
        #     hydrolakes_crs = hydrolakes.crs
        #     extentpoly2 = extentpoly.buffer(xres)
# Create slightly larged extent for clipping in 4326 crs
        #     extentpoly2 =  extentpoly2.to_crs(hydrolakes_crs)
# Reproject to 4326 CRS
        #     thelake = gpd.clip(hydrolakes,extentpoly2)
# Clip HydroLakes to model domain extent
        #     thelake = thelake.to_crs("EPSG:%s" %(EPSG))
# Save
```

```
        #     if len(thelake) == 0:
        #         print('There are no lakes in the model domain')
        #         d = {'geometry': [Polygon([(0, 0), (0,0),(0,0)])]}
        #         thelake = gpd.GeoDataFrame(d,crs="EPSG:%s" %(EPSG))
        #     else:
        #         thelake['TYPE_2'] = 'Lake'
# Add label to identify lakes
        #         thelake.geometry = thelake.buffer(50)
# Extra buffer to account low precision of hydrolakes
        #         thelake =
gpd.overlay(thelake,extentpoly,how='intersection')        # Clip to
extent in correct EPSG, ensure clipped area is correct
        #     thelake.to_file("%s%s_HydroLAKES.shp" %(folders[0],delta))

        print('\n[Step 3A][Make_Watermask][Smoothing water and land
masks] .......\n')

        #if os.path.isfile('%s%s_expanded_10.tif'
%(folders[8],watermaskname))==False:
        print('\n[Step 3A][Make_Watermask][Start with the GEE watermask
at 10m resolution] .......\n')
        full_watermask = rasterio.open('%s%s_10.tif'
%(folders[8],watermaskname))

        save_profile_10m = full_watermask.profile
        save_profile_10m['Compress'] = 'deflate'
        save_profile_10m['nodata'] = 0
        save_profile_10m['dtype'] = 'float64'
        if clean_with_landcover:
            if landcovermethod == 'Copernicus':
                mangrove_class = 9
                wetland_class = 10
                crop_class = 6
            elif landcovermethod == 'WorldCover':
                mangrove_class = 95
                wetland_class = 90
                crop_class = 40
                water_class = 80

            landcover = rasterio.open('%s%s_%s_10.tif'
%(folders[8],delta,landcovermethod)).read(1).astype('int')
            mangrove = gdal.Open('%s%s_GMW_10.tif'
%(folders[8],delta)).ReadAsArray()

            landcover[landcover==-9999] = 0
            landcover = np.where(mangrove==1,mangrove_class,landcover)

            iswetland = rasterio.open("%s%s_wetlands_10.tif"
%(folders[8],delta)).read(1)

            iscrop = rasterio.open("%s%s_crops_10.tif"
%(folders[8],delta)).read(1)
```

```
        print('\n[Step 3A][Make_Watermask][Removing wetland and
agriculture areas from water mask] ......\n')
        print('\n[Step 3A][Make_Watermask][Determine whether 0 or 1 is
water in GEE watermask] ......\n')

        test_ocean = hydropoly[hydropoly['TYPE']=='Ocean or Sea']
        test_ocean =
test_ocean.loc[test_ocean.area==max(test_ocean.area)]
        watermaskvalue = int(zonal_stats(vectors =
test_ocean['geometry'],raster = '%s%s_10.tif'
%(folders[8],watermaskname),stats='majority',nodata = '-
9999')[0]['majority'])

        full_watermask = full_watermask.read(1)
        print('\n[Step 3A][Make_Watermask][Update water mask to remove
mangroves, crops] ......\n')
        if clean_with_landcover:
            full_watermask =
np.where(np.isnan(full_watermask),0,np.where((mangrove==1) |
(iscrop==1),0,np.where((landcover==water_class) |
(full_watermask==watermaskvalue),1,0)))
        else:
            full_watermask =
np.where(np.isnan(full_watermask),0,np.where(full_watermask==watermaskval
ue,1,0))
        # full_watermask2 =
ndimage.morphology.binary_hit_or_miss(full_watermask, structure1 =
structure_flip.astype(full_watermask.dtype))


        print('\n[Step 3A][Make_Watermask][Clean up areas of bad
connectivity] ......\n')
        print('\n[Step 3A][Make_Watermask][Clean up areas of bad
connectivity][Single Pixel channels --> two pixels channels] ......\n')

        structure = np.array([[0., 1., 0.], [0., 1., 0.]])
        full_watermask = fix_connectivity(structure,full_watermask,(-1,-
1))

        structure = np.array([[1., 1., 1.], [0., 1., 0.]])
        full_watermask = fix_connectivity(structure,full_watermask,(0,-
1))
        structure = np.array([[0., 1., 0.], [1., 1., 1.]])
        full_watermask = fix_connectivity(structure,full_watermask,(-
1,1))

        structure = np.array([[0., 1., 1.], [0., 1., 0.]])
        full_watermask = fix_connectivity(structure,full_watermask,(0,1))
        structure = np.array([[0., 1., 0.], [0., 1., 1.]])
        full_watermask = fix_connectivity(structure,full_watermask,(-
1,1))

        structure = np.array([[1., 1., 0.], [0., 1., 0.]])
```

```
        full_watermask = fix_connectivity(structure,full_watermask,(0,-
1))
        structure = np.array([[0., 1., 0.], [1., 1., 0.]])
        full_watermask = fix_connectivity(structure,full_watermask,(-
1,1))

        print('\n[Step 3A][Make_Watermask][Clean up areas of bad
connectivity][Corner connectivity --> edge connectivity] ......\n')
        structure = np.array([[1., 0.], [0., 1.]])
        full_watermask = fix_connectivity(structure,full_watermask,(-
1,0))
        structure = np.array([[0., 1.], [1., 0.]])
        full_watermask = fix_connectivity(structure,full_watermask,(0,0))
        #full_watermask = ndimage.binary_opening(full_watermask, np.ones
([2,2]),iterations=1).astype(full_watermask.dtype)
        #full_watermask2 = np.where(full_watermask==1,0,1)
        #full_watermask2 = ndimage.binary_opening(full_watermask2,
np.ones([3,3]),iterations=1).astype(full_watermask2.dtype)
        #print('binary_opening 3x3')
        #full_watermask = np.where(full_watermask2==0,1,0)
        print('\n[Step 3A][Make_Watermask][Save as expanded water mask]
......\n')
        with rasterio.open('%s%s_expanded_10.tif'
%(folders[8],watermaskname),'w', **save_profile_10m) as dst:
            dst.write_band(1,full_watermask)

        if xres != 10:
            print('\n[Step 3A][Make_Watermask][Resample to desired
resolution %sm] ......\n' %(xres))
            os.system('gdalwarp -overwrite -tr %s %s %s%s_expanded_10.tif
%s%s_expanded_%s.tif -te %s %s %s %s -srcnodata -9999 '\
                      '-dstnodata -9999 -co COMPRESS=DEFLATE'

%(xres,yres,folders[8],watermaskname,folders[8],watermaskname,xres,ulx,lr
y,lrx,uly))


        print('\n[Step 3A][Make_Watermask] Finished .........\n')
        return watermaskname
    else:
        print('\n[Step 3A][Make_Watermask] Skip .........\n')
        return watermaskname

def more_opening(delta,folders,watermaskname,structure,ref,parameters):
    if structure == None:
        print('\n[Step 3B][Open_Watermask] Skip .........\n')
    else:
        print('\n[Step 3B][Open_Watermask] \n')
        print('If an area has many bridges or narrow channel sections,
more opening might be needed')
        save_profile_xres = ref.profile
        save_profile_xres['compress'] = 'deflate'
        xres,yres = int(save_profile_xres['transform'][0]),-
int(save_profile_xres['transform'][4])
```

```
        EPSG = parameters['EPSG'][0]
# Coordinate System must be UTM
        ulx  = parameters['ulx'][0]
# ULX coordinate
        lry  = parameters['lry'][0]
# LRY coordinate
        lrx  = parameters['lrx'][0]
# LRX coordinate
        uly  = parameters['uly'][0]

        full_watermask_file = rasterio.open('%s%s_expanded_10.tif'
%(folders[8],watermaskname))
        save_profile_10m = full_watermask_file.profile
        full_watermask = full_watermask_file.read(1)
        full_watermask2 = np.where(full_watermask==1,0,1)

        print('binary_opening %sx%s' %(structure,structure))
        full_watermask2 = ndimage.binary_opening(full_watermask2,
np.ones([structure,structure]),iterations=1).astype(full_watermask2.dtype
)
        full_watermask = np.where(full_watermask2==0,1,0)

        print('\n[Step 3A][Make_Watermask][Save as expanded water mask]
......\n')
        with rasterio.open('%s%s_expanded_10_%sx%s.tif'
%(folders[8],watermaskname,structure,structure),'w', **save_profile_10m)
as dst:
            dst.write_band(1,full_watermask)

        if xres != 10:
            print('\n[Step 3A][Make_Watermask][Resample to desired
resolution %sm] ......\n' %(xres))
            os.system('gdalwarp -overwrite -tr %s %s
%s%s_expanded_10_%sx%s.tif %s%s_expanded_%s_%sx%s.tif -te %s %s %s %s -
srcnodata -9999 '\
                      '-dstnodata -9999 -co COMPRESS=DEFLATE'

%(xres,yres,folders[8],watermaskname,structure,structure,folders[8],water
maskname,xres,structure,structure,ulx,lry,lrx,uly))

        full_watermask = rasterio.open('%s%s_expanded_%s_%sx%s.tif'
%(folders[8],watermaskname,xres,structure,structure))
        save_profile_xres = full_watermask.profile
        save_profile_xres['nodata']  = -9999
        full_watermask = full_watermask.read(1)
        full_watermask = np.where(full_watermask==-
9999,np.nan,full_watermask)

        print('\n[Step 3A][Make_Watermask][Fix water mask at domain edges
caused by resampling] ......\n')
        temp_watermask = np.where(full_watermask==0,2,full_watermask)
        fillers = np.where(temp_watermask==-9999,0,temp_watermask)## 0
will be filled, nans will be excluded
```

```
        full_watermask =
np.where(_fillnodata(temp_watermask,fillers,500)>1,0,1)

        # structure = np.array([[1., 0.], [0., 1.]])
        # full_watermask = fix_connectivity(structure,full_watermask,(-
1,0))
        # structure = np.array([[0., 1.], [1., 0.]])
        # full_watermask =
fix_connectivity(structure,full_watermask,(0,0))
        # full_watermask = ndimage.binary_opening(full_watermask, np.ones
([2,2]),iterations=1).astype(full_watermask.dtype)
        full_watermask = np.where(full_watermask==1,1,-9999)

        print('\n[Step 3A][Make_Watermask][Save as final water mask at
%sm resolution] ......\n' %(xres))
        with rasterio.open('%s%s_watermask_%s.tif'
%(folders[8],delta,xres),'w', **save_profile_xres) as dst:
            dst.write_band(1,full_watermask)

        print('\n[Step 3A][Make_Watermask][Make land mask as inverse of
water mask] ......\n')
        full_landmask =
np.where(np.isnan(full_watermask),np.nan,np.where(full_watermask==1,-
9999,1))

        print('\n[Step 3A][Make_Watermask][Save as final land mask at %sm
resolution] ......\n' %(xres))
        with rasterio.open("%s%s_landmask_%s.tif"
%(folders[8],delta,xres),'w', **save_profile_xres) as dst:
            dst.write_band(1,full_landmask.astype('float64'))

    print('\n[Step 3A][Make_Watermask] Finished .........\n')


##########################################################################
###
##########################################################################
###
##########################################################################
###
## PART IV
## OUTPUT: landcover, water types, DEM
##########################################################################
#

def
make_polygons(delta,folders,parameters,ref,watermaskname,templates_path,s
kip=False):
    '''
    Parameters
    ----------
    delta : string
        AOI name, must match input shapefile and folder.
    folders : np.array of strings
```

```
        folders within deltapath for model files.
    parameters : pd.dataframe
        Configuration parameters for model setup.
    ref : open DatasetReader (rasterio)
        The reference file used to detemine resolution, width, height,
extent of all rasters
    watermaskname : string
        Name of the best water mask from GEE
    templates_path : string
        directory where templates are stored
    skip : boolean
        If True, do not build water polygons.
    '''

    step = 4

print('\n\n\n##################################################################
###################################')
    print('#################################[Step
%s][Make_Polygons]#######################################' %(step))

print('##################################################################
############################\n')

    save_profile = ref.profile
    save_profile['compress'] = 'deflate'
    xres,yres = int(save_profile['transform'][0]),-
int(save_profile['transform'][4])

    EPSG = parameters['EPSG'][0]
# Coordinate System must be UTM
    ulx  = parameters['ulx'][0]
# ULX coordinate
    lry  = parameters['lry'][0]
# LRY coordinate
    lrx  = parameters['lrx'][0]
# LRX coordinate
    uly  = parameters['uly'][0]

    extentpoly = gpd.read_file("%s%s_extent_%s.shp"
%(folders[7],delta,EPSG))
    landcovermethod = parameters['LandcoverMethod'][0]               #
File or method used for landcover classification
    wetland_class = int(parameters['WetlandClass'][0])      # Class #
representing wetlands in landcover file

    try: os.mkdir(folders[1])
    except:''


##################################################################################
#####
    ###################### Defined land, water polygons
############################
```

```
#########################################################################
#####
    print('\n[Step %s][Make_Polygons][Open land/water mask] .......\n'
%(step))
    print('#################### Water delineated with %s_%s'
%(watermaskname,xres))
    if skip == False:
        if os.path.isfile('%s%s_landmask_%s.shp'
%(folders[7],delta,xres))==False:
            #os.remove('%s%s_watermask.shp' %(folders[7],delta))
            # if os.path.isfile('%s%s_watermask_%s.shp'
%(folders[7],delta,xres)):
            #     os.remove('%s%s_watermask_%s.shp'
%(folders[7],delta,xres))
            # if os.path.isfile('%s%s_landmask_%s.shp'
%(folders[7],delta,xres)):
            #     os.remove('%s%s_landmask_%s.shp'
%(folders[7],delta,xres))
            filein = open(templates_path + 'run_gdal24.sh')
            template = Template(filein.read())
            # For some reason, the gdal goddess is picky and an older
version of gdal is needed for polgonizing successfully.
            # This will leave our conda environment and run
gdal_polygonize in a env with older gdal ;)
            replacements = {'start':  "$(conda shell.bash hook)",
                            'environment':  'testgdal',
                            'command' : 'gdal_polygonize.py
"%s%s_watermask_%s.tif" "%s%s_watermask_%s.shp" -8 -b 1 -f "ESRI
Shapefile" %s DN' %(folders[8],delta,xres,folders[7],delta,xres,delta),
                            'command2' : 'gdal_polygonize.py
"%s%s_landmask_%s.tif" "%s%s_landmask_%s.shp" -8 -b 1 -f "ESRI Shapefile"
%s DN' %(folders[8],delta,xres,folders[7],delta,xres,delta)
                            }
            makeoutput = template.substitute(replacements)
            file = open(folders[1] + 'run_gdal24_test.sh', 'w')
            file.write(makeoutput)
            file.close()
            subprocess.call(['sh',folders[1] + 'run_gdal24_test.sh'])

        hydropoly = gpd.read_file("%s%s_hydropolys.shp"
%(folders[0],delta))
        # hydropoly =
gpd.overlay(hydropoly,extentpoly,how='intersection')
        try:
            hydrolake = gpd.read_file("%s%s_SWOTPLD.shp"
%(folders[0],delta))
        except:
            hydrolake = gpd.read_file("%s%s_HydroLAKES.shp"
%(folders[0],delta))
        # hydrolake =
gpd.overlay(hydrolake,extentpoly,how='intersection')
```

```
        ndwi_watermask = gpd.read_file("%s%s_watermask_%s.shp"
%(folders[7],delta,xres))
        # ndwi_watermask =
gpd.overlay(ndwi_watermask,extentpoly,how='intersection')
        ndwi_watermask['dissolve'] = 1

        ndwi_landmask = gpd.read_file("%s%s_landmask_%s.shp"
%(folders[7],delta,xres))
        # ndwi_landmask =
gpd.overlay(ndwi_landmask,extentpoly,how='intersection')
        ndwi_landmask['dissolve'] = 1


########################################################################
        ## WATER BODIES FOR DELINEATION OF BATHYMETRY ##

########################################################################
        ## All water bodies from hydrolakes database (UCLA)
        print('\n[Step %s][Make_Polygons][All initial water from
Hydropolys and HydroLAKES] .......\n'%(step))
        try:
            hydro_all = gpd.overlay(hydropoly,hydrolake,how='union')
        except: hydro_all = hydropoly.copy(deep=True)
        hydro_all =
hydro_all.explode(index_parts=True).reset_index(drop=True)
        hydro_all.geometry = hydro_all.buffer(0)
        hydro_all['dissolve'] = 1

        ## LAKES
        print('\n[Step %s][Make_Polygons][Lakes from Hydropolys Type
"Lake" and HydroLAKES] .......\n'%(step))
        try:lakes = gpd.read_file("%s%s_lakes_%s.shp"
%(folders[7],delta,xres))
        except:
            try:
                lakes = hydro_all.loc[(hydro_all['TYPE'] == 'Lake') |
(hydro_all['TYPE_2'] == 'Lake')].reset_index(drop=True)
            except:
                lakes = hydro_all.loc[(hydro_all['TYPE'] ==
'Lake')].reset_index(drop=True)
            lakes.drop(lakes.columns[:-2],inplace=True,axis=1)
            if len(lakes) <1:
                print('There are no lakes')
            else:
                lakes.to_file("%s%s_lakes_%s.shp"
%(folders[7],delta,xres)) ## SAVE
                print('###################### Lake polygons saved to
%s%s_lakes_%s.shp' %(folders[7].split(delta)[-1],delta,xres))
        # else:
        #     lakes = gpd.overlay(lakes,extentpoly,how='intersection')
        #     lakes.to_file("%s%s_lakes_%s.shp" %(folders[7],delta,xres))
## SAVE

        ## OCEANS
```

```
        print('\n[Step %s][Make_Polygons][Oceans from Hydropolys Type
Ocean or Sea] .......\n'%(step))
        try:oceans = gpd.read_file("%s%s_fulloceans_%s.shp"
%(folders[7],delta,xres))
        except:
            oceans = hydro_all.loc[hydro_all['TYPE'] == 'Ocean or
Sea'].reset_index(drop=True)
            oceans.drop(oceans.columns[:-2],inplace=True,axis=1)
            oceans.geometry = oceans.buffer(200)
            #oceans = oceans.dissolve(by='dissolve')

            oceans = gpd.overlay(oceans,ndwi_landmask,how='difference')
            oceans = gpd.overlay(oceans,extentpoly,how='intersection')
            oceans =
oceans.dissolve(by='dissolve').reset_index(drop=True)
            oceans.to_file("%s%s_fulloceans_%s.shp"
%(folders[7],delta,xres))
        # else:
        #     oceans = gpd.overlay(oceans,extentpoly,how='intersection')
        #     oceans.to_file("%s%s_fulloceans_%s.shp"
%(folders[7],delta,xres))
        print('#################### Ocean polygons saved to
%s%s_fulloceans_%s.shp' %(folders[7].split(delta)[-1],delta,xres))
        os.system('gdal_rasterize -burn 1 -tr %s %s -te %s %s %s %s
%s%s_fulloceans_%s.shp %s%s_fulloceans_%s.tif -co COMPRESS=DEFLATE'
%(xres,yres,ulx,lry,lrx,uly,folders[7],delta,xres,folders[8],delta,xres))

        ## OCEANS AND LAKES together!
        print('\n[Step %s][Make_Polygons][Ocean and lakes]
.......\n'%(step))
        try: ocean_lake = gpd.read_file("%s%s_ocean_and_lake_%s.shp"
%(folders[7],delta,xres))
        except:
            ocean_lake =
gpd.overlay(oceans,lakes,how='union',keep_geom_type=True)
            #ocean_lake = oceans + lakes
            ocean_lake['dissolve'] = 1
            ocean_lake =
ocean_lake.dissolve(by='dissolve').reset_index(drop=True)
            ocean_lake.to_file('%s%s_ocean_and_lake_%s.shp'
%(folders[7],delta,xres))
            print('#################### Ocean and lake polygons saved to
%s%s_ocean_lakes_%s.shp' %(folders[7],delta,xres))
        # else:
        #     ocean_lake =
gpd.overlay(ocean_lake,extentpoly,how='intersection')
        #     ocean_lake.to_file('%s%s_ocean_and_lake_%s.shp'
%(folders[7],delta,xres))


        ## CONNECTED WATER, EXCLUDING LAKES
        print('\n[Step %s][Make_Polygons][Find connected water, starting
with largest ocean polygon] .......\n'%(step))
        try:
```

```
            water_connected_not_lakes =
gpd.read_file("%s%s_water_connected_%s.shp" %(folders[7],delta,xres))
        except:
            water_connected_not_lakes =
ndwi_watermask.dissolve(by='dissolve')
            water_connected_not_lakes =
water_connected_not_lakes.explode(index_parts=True)
            water_connected_not_lakes =
water_connected_not_lakes.loc[water_connected_not_lakes.area==max(water_c
onnected_not_lakes.area)].copy(deep=True)
            water_connected_not_lakes =
water_connected_not_lakes.dissolve(by='dissolve').reset_index(drop=True)
            #water_connected_not_lakes =
findconnectedwater(watermask_not_lakes)

water_connected_not_lakes.to_file("%s%s_water_connected_%s.shp"
%(folders[7],delta,xres))
            print('###################### All water connected to ocean
polygons saved to %s%s_water_connected_%s.shp'
%(folders[7].split(delta)[-1],delta,xres))
        # else:
        #     water_connected_not_lakes =
gpd.overlay(water_connected_not_lakes,extentpoly,how='intersection')
        #
water_connected_not_lakes.to_file("%s%s_water_connected_%s.shp"
%(folders[7],delta,xres))

        print('\n[Step %s][Make_Polygons][River from water that is not
lake or ocean] .......\n'%(step))
        try:rivers = gpd.read_file("%s%s_rivers_%s.shp"
%(folders[7],delta,xres))
        except:
            rivers =
gpd.overlay(water_connected_not_lakes,ocean_lake,how='difference')
            rivers =
rivers.explode(index_parts=True).reset_index(drop=True)
            rivers = rivers[rivers.geometry.type=='Polygon']
            rivers= gpd.overlay(rivers,extentpoly,how='intersection')
            rivers.to_file("%s%s_rivers_%s.shp" %(folders[7],delta,xres))
            print('###################### River polygons saved to
%s%s_rivers_%s.shp' %(folders[7].split(delta)[-1],delta,xres))
        # else:
        #     rivers = gpd.overlay(rivers,extentpoly,how='intersection')
        #     rivers.to_file("%s%s_rivers_%s.shp"
%(folders[7],delta,xres))
        #print('###################### River polygons saved to
%s%s_rivers.shp' %(folders[7].split(delta)[-1],delta))
        os.system('gdal_rasterize -burn 1 -tr %s %s -te %s %s %s %s
%s%s_rivers_%s.shp %s%s_rivers_%s.tif -co COMPRESS=DEFLATE'
%(xres,yres,ulx,lry,lrx,uly,folders[7],delta,xres,folders[8],delta,xres))

        ## CONNECTED WATER, ALL
        print('\n[Step %s][Make_Polygons][All water from connected water
and lakes] .......\n'%(step))
```

```
        try: allwater =
gpd.read_file("%s%s_water_connected_and_lakes_%s.shp"
%(folders[7],delta,xres))
        except:
            allwater =
gpd.overlay(ocean_lake,water_connected_not_lakes,how='union',keep_geom_ty
pe=True)
            allwater =
allwater.explode(index_parts=True).reset_index(drop=True)
            allwater.drop(allwater.columns[:-1],inplace=True,axis=1)
            #allwater.geometry = allwater.buffer(40)
            #allwater.geometry = allwater.buffer(-40)
            allwater =
gpd.overlay(allwater,extentpoly,how='intersection')
        # else:
        #     allwater =
gpd.overlay(allwater,extentpoly,how='intersection')
        allwater['dissolve'] = 1
        allwater =
allwater.dissolve(by='dissolve').reset_index(drop=True)
        allwater.to_file('%s%s_water_connected_and_lakes_%s.shp'
%(folders[7],delta,xres))
        print('###################### All water polygons saved to
%s%s_water_connected_and_lakes_%s.shp' %(folders[7].split(delta)[-
1],delta,xres))

        print('\n[Step %s][Make_Polygons][Land from area that is not
water] .......\n'%(step))
        try:lands = gpd.read_file("%s%s_lands_%s.shp"
%(folders[7],delta,xres))
        except:
            lands = gpd.overlay(extentpoly,allwater,how='difference')
            lands =
lands.explode(index_parts=True).reset_index(drop=True)
            lands = lands[lands.geometry.type=='Polygon']
        # else:
        #     lands = gpd.overlay(lands,extentpoly,how='intersection')
        lands.geometery = lands.buffer(0)
        lands['dissolve'] = 1
        lands = lands.dissolve(by='dissolve').reset_index(drop=True)
        lands.to_file("%s%s_lands_%s.shp" %(folders[7],delta,xres))
        print('###################### Land polygons saved to
%s%s_lands_%s.shp' %(folders[7].split(delta)[-1],delta,xres))

        # print('[Step 3][Extend river boundary offshore by 1000m]
.......')
        # extendedriver = rivers.copy(deep=True)
        # extendedriver.geometry = extendedriver.buffer(1000)
        # extendedriver =
gpd.overlay(extendedriver,lands,how='difference')
        # #extendedriver = extendedriver.difference(lands)
        # extendedriver.to_file("%s%s_riverextended.shp"
%(folders[7],delta))
```

```
        # print('[Step 3][Move ocean boundary offshore by 1000m]
.......')
        # extendedocean =
gpd.overlay(water_connected_not_lakes,extendedriver,how='difference')
        # #extendedocean =
water_connected_not_lakes.difference(extendedriver)
        # extendedocean.to_file("%s%s_oceanextended.shp"
%(folders[7],delta))

        print('#################### Rasterizing land, ocean, lake, and
river polygons')
        os.system('gdal_rasterize -burn 1 -tr %s %s -te %s %s %s %s
%s%s_water_connected_and_lakes_%s.shp
%s%s_water_connected_and_lakes_%s.tif -co COMPRESS=DEFLATE'
%(xres,yres,ulx,lry,lrx,uly,folders[7],delta,xres,folders[8],delta,xres))
        os.system('gdal_rasterize -burn 1 -tr %s %s -te %s %s %s %s
%s%s_water_connected_and_lakes_%s.shp
%s%s_water_connected_and_lakes_%s.tif -co COMPRESS=DEFLATE'
%(xres,yres,ulx,lry,lrx,uly,folders[7],delta,xres,folders[8],delta,xres))
        os.system('gdal_rasterize -burn 1 -tr %s %s -te %s %s %s %s
%s%s_lands_%s.shp %s%s_lands_%s.tif -co COMPRESS=DEFLATE'
%(xres,yres,ulx,lry,lrx,uly,folders[7],delta,xres,folders[8],delta,xres))
        os.system('gdal_rasterize -burn 1 -tr %s %s -te %s %s %s %s
%s%s_lakes_%s.shp %s%s_lakes_%s.tif -co COMPRESS=DEFLATE'
%(xres,yres,ulx,lry,lrx,uly,folders[7],delta,xres,folders[8],delta,xres))
        print('\n[Step %s][Make_Polygons] Finished .......\n'%(step))

    else:
        print('\n[Step %s][Make_Polygons] SKIP.......\n'%(step))

#########################################################################
###
#########################################################################
###
#########################################################################
###
## PART V
## OUTPUT: channel network, distances, widths
#########################################################################
#
def
make_channel_networks(folders,delta,ref,parameters,pixel_step,skip=False)
:

    '''
    Parameters
    ----------
    parameters : pd.dataframe
        Configuration parameters for model setup.
    skip : boolean
        If True, do not build water polygons.

    Returns
    -------
```

```
    distance : np.array
        Distances from ocean along river channels.
    widths : np.array
        Widths of river cross sections, approximately every 150m
    riverscale : float
        Maximum triangle area for generating mesh in rivers.
    med_width : float
        median width of river channels.
    cellsperwidth : float
        minimum number of cells per channel width.
    watermask : np.array
        Mask of water/land.

    '''
    step = 5

print('\n\n\n##################################################################
###################################')
    print('################################[Step
%s][Make_Channel_Networks]###################################'%(step))

print('##################################################################
############################\n')
    save_profile = ref.profile
    save_profile['compress'] = 'deflate'

    xres,yres = int(save_profile['transform'][0]),-
int(save_profile['transform'][4])


#########################################################################
##
    ###################### Import Config Parameters
#########################

#########################################################################
##

    parameters.iloc[0]
    EPSG = parameters['EPSG'][0]
# Coordinate System must be UTM
    ulx  = parameters['ulx'][0]
# ULX coordinate
    lry  = parameters['lry'][0]
# LRY coordinate
    lrx  = parameters['lrx'][0]
# LRX coordinate
    uly  = parameters['uly'][0]
# ULY coordinate
    #os.system('gdal_rasterize -burn 1 -tr %s %s -te %s %s %s %s -tap
%s%s_oceanextended.shp %s%s_oceanextended.tif'
%(xres,yres,ulx,lry,lrx,uly,folders[7],delta,folders[1],delta))
```

```
    #os.system('gdal_rasterize -burn 1 -tr %s %s -te %s %s %s %s -tap
%s%s_riverextended.shp %s%s_riverextended.tif'
%(xres,yres,ulx,lry,lrx,uly,folders[7],delta,folders[1],delta))

    try: os.mkdir(folders[1])
    except:''


#########################################################################
#
    ## RASTERIZE WATER/LAND DELIENATI SHAPEFILES

#########################################################################
#
    # print('\n[Step %s][Make_Channel_Networks][Import water mask (%sm)]
.......\n'%(step,xres))
    # watermask = rasterio.open('%s%s_watermask_%s.tif'
%(folders[8],delta,int(xres)))
    print('\n[Step %s][Make_Channel_Networks][Import river mask (%sm)]
.......\n'%(step,xres))
    #river = rasterio.open('%s%s_rivers_%s.tif' %(folders[8],delta,xres))
    river = rasterio.open('%s%s_watermask_%s.tif'
%(folders[8],delta,xres))
    rivermask = np.where(river.read(1) ==1,1.,0.)
    print('\n[Step %s][Make_Channel_Networks][Import ocean mask (%sm)]
.......\n'%(step,xres))
    ocean = rasterio.open('%s%s_fulloceans_%s.tif'
%(folders[8],delta,xres))


#########################################################################
#####
    ######################## Get River Widths and Depths
#############################

#########################################################################
#####
    print('\n[Step %s][Make_Channel_Networks][Use Orinoco code (Charlie)
to get distance and widths of rivers] .......\n'%(step))
    print('###################### Pixel step will be %s' %(pixel_step))
    print('###################### Therefore, segments will be %s * %s =
%sm wide' %(xres, pixel_step, xres*pixel_step))

    if skip ==False:
        print('\n[Step %s][Make_Channel_Networks][Orinoco]
.......\n'%(step))
        distance = make_distance(river,ocean,folders,delta,pixel_step)
    else:
        print('\n[Step %s][Make_Channel_Networks] SKIP .......\n'%(step))

    distance = gdal.Open('%s%s_distance_%s.tif'
%(folders[8],delta,xres)).ReadAsArray()
    widths = np.where((rivermask==1),gdal.Open('%s%s_widths_%sx%s.tif'
%(folders[8],delta,xres,pixel_step)).ReadAsArray(),np.nan)
```

```
    os.system('gdal_rasterize -burn 1 -ts %s %s -te %s %s %s %s '\
            '%s%s_river_centerline_%sx%s.shp
%s%s_river_centerline_%sx%s.tif '\
            '-co COMPRESS=DEFLATE'

%(distance.shape[1],distance.shape[0],ulx,lry,lrx,uly,folders[7],delta,xr
es,pixel_step,folders[8],delta,xres,pixel_step))


    print('\n[Step %s][Make_Channel_Networks][Finished]
.......\n'%(step))


    return distance, widths

##############################################################################
###
##############################################################################
###
##############################################################################
###
## PART VI
## OUTPUT: Final elevation file
##############################################################################
#
def
make_model_foundation(path,parameters,delta,folders,ref,distance,widths,w
atermask,pixel_step,build_path):
    '''

    Parameters
    ----------
    parameters : pd.dataframe
        Configuration parameters for model setup.
    delta : string
        AOI.
    deltapath : string
        directory for AOI.
    folders : np.array of strings
        folders within deltapath for model files.
    save_profile : TYPE
        DESCRIPTION.
    skip : boolean
        If True, do not build water polygons.
    distance : np.array
        Distances from ocean along river channels.

    Returns
    -------
    '''
    step = 6
```

```
print('\n\n\n##################################################################
###################################')
    print('################################[Step
%s][Make_Model_Foundation]################################'%(step))

print('##################################################################
############################\n')

#########################################################################
##
    ###################### Import Config Parameters
#########################

#########################################################################
##
    save_profile = ref.profile
    save_profile['compress'] = 'deflate'

    xres,yres = int(save_profile['transform'][0]),-
int(save_profile['transform'][4])
    parameters.iloc[0]
    EPSG = parameters['EPSG'][0]
# Coordinate System must be UTM
    ulx  = parameters['ulx'][0]
# ULX coordinate
    lry  = parameters['lry'][0]
# LRY coordinate
    lrx  = parameters['lrx'][0]
# LRX coordinate
    uly  = parameters['uly'][0]
# ULY coordinate
    #manningfile = parameters['ManningLUT'][0]               # File for
manning lookup table
    landcovermethod = parameters['LandcoverMethod'][0]              #
File or method used for landcover classification
    max_nearshore_depth = int(parameters['MaxNearshoreDepth'][0])
# Bathymetry threshold (below this depth, NAN)
    max_ocean_depth = int(parameters['MaxOceanDepth'][0])         #
Bathymetry threshold (below this depth, NAN)
    max_river_depth = int(parameters['MaxRiverDepth'][0])   # Bathymetry
threshold (below this depth, NAN)
    riverBoundary = int(parameters['RiverOceanBoundary'][0])# Distance
upstream that represents change from lower to upper river geometry
    landmethod = parameters['LandElevMethod'][0]                # File or
method used for land topography
    oceanmethod = parameters['OceanElevMethod'][0]              # File or
method used for ocean bathymetry
    lower_rivermethod = parameters['LowerRiverElevMethod'][0]   # File or
method used for lower river bathymetry
    upper_rivermethod = parameters['UpperRiverElevMethod'][0]   # File or
method used for upper river bathymetry
    lakemethod = parameters['LakeElevMethod'][0]                # File or
method used for lake bathymetry
```

```
    if lakemethod[:8] == 'constant':
        constantLakeDepth = float(lakemethod[9:])
        lakefile_name = 'uni%sm' %(str(int(constantLakeDepth)))
    else: lakefile_name = lakemethod
    if os.path.isfile("%s%s_wetlands_%s.tif" %(folders[8],delta,xres)) ==
False:
        os.system('gdalwarp -overwrite -tr %s %s %s%s_wetlands_10.tif
%s%s_wetlands_%s.tif -te %s %s %s %s '\
                  '-srcnodata -9999 -dstnodata -9999 -co
COMPRESS=DEFLATE'

%(xres,yres,folders[8],delta,folders[8],delta,xres,ulx,lry,lrx,uly))
    iswetland = rasterio.open("%s%s_wetlands_%s.tif"
%(folders[8],delta,xres)).read(1)

    wetlandmethod  = parameters['WetlandElevMethod'][0]         # File or
method used for wetland bathymetry
    if wetlandmethod[:8] == 'constant':
        constantWetlandDepth = float(wetlandmethod[9:])
        if constantWetlandDepth<1:
            wetlandfile_name = 'uni0%sm'
%(str(int(constantWetlandDepth*10)))
        else:
            wetlandfile_name = 'uni%sm' %(str(int(constantWetlandDepth)))
    else: wetlandfile_name = wetlandmethod
    try: os.mkdir(folders[1])
    except:''
    # if (os.path.isfile('%s%s_water_connected_and_lakes_%s.tif'
%(folders[8],delta,xres))==False) or (os.path.isfile('%s%s_oceans_%s.tif'
%(folders[1],delta,xres))==False):
    #     print('###################### Rasterizing land, ocean, lake, and
river polygons')
    #     os.system('gdal_rasterize -burn 1 -tr %s %s -te %s %s %s %s -
tap %s%s_water_connected_and_lakes_%s.shp
%s%s_water_connected_and_lakes_%s.tif -co COMPRESS=DEFLATE'
%(xres,yres,ulx,lry,lrx,uly,folders[7],delta,xres,folders[8],delta,xres))
    #     os.system('gdal_rasterize -burn 1 -tr %s %s -te %s %s %s %s -
tap %s%s_oceans_%s.shp %s%s_oceans_%s.tif'
%(xres,yres,ulx,lry,lrx,uly,folders[7],delta,xres,folders[1],delta,xres))
    #     os.system('gdal_rasterize -burn 1 -tr %s %s -te %s %s %s %s -
tap %s%s_rivers_%s.shp %s%s_rivers_%s.tif'
%(xres,yres,ulx,lry,lrx,uly,folders[7],delta,xres,folders[1],delta,xres))
    #     os.system('gdal_rasterize -burn 1 -tr %s %s -te %s %s %s %s -
tap %s%s_lakes_%s.shp %s%s_lakes_%s.tif'
%(xres,yres,ulx,lry,lrx,uly,folders[7],delta,xres,folders[1],delta,xres))
    print('\n[Step %s][Make_Model_Foundation][Determine all methods for
calculating elevation for each land/water type] .......\n'%(step))
    print('###################### Method for land = %s' %(landmethod))
    print('###################### Method for ocean = %s' %(oceanmethod))
    print('###################### Method for lower river reaches = %s'
%(lower_rivermethod))
    print('###################### Method for upper river reaches = %s'
%(upper_rivermethod))
    print('###################### Method for lakes = %s' %(lakemethod))
```

```
    print('###################### Method for wetlands = %s'
%(wetlandmethod))
    print('###################### Max Depth for Rivers = %s'
%(max_river_depth))
    print('###################### Max Depth for Oceans = %s'
%(max_ocean_depth))
    print('###################### Max Depth for Nearshore = %s'
%(max_nearshore_depth))
    print('###################### Boundary between Upper and Lower river
reaches =  %sm' %(riverBoundary))

    #discharge  = parameters['Discharge'][0]         # File or method
used for wetland bathymetry
    #centerline = gpd.read_file('%s%s_river_centerline_%s.shp'
%(folders[7],delta,round(pixel_step*xres)))
    # os.system('gdal_rasterize -burn 1 -tr %s %s -te %s %s %s %s
%s%s_river_centerline_%s.shp '\
    #           '%s%s_river_centerline_%s.tif -co COMPRESS=DEFLATE'
    #
%(xres,yres,ulx,lry,lrx,uly,folders[7],delta,round(pixel_step*xres),folde
rs[8],delta,round(pixel_step*xres)))
    #extentpoly = gpd.read_file("%s%s_extent_%s.shp"
%(folders[7],delta,EPSG))
    max_river_width = parameters['MaxRiverWidth'][0]
    ref_array = ref.read(1)
    if riverBoundary>0:
        elev_name = 'Elevation_ocean-%s_land-%s_above%sm-%s_below%sm-
%s_wetland-%s_lakes-%s'
%(oceanmethod,landmethod,riverBoundary,upper_rivermethod,riverBoundary,lo
wer_rivermethod,wetlandfile_name,lakefile_name)
    else:
        elev_name = 'Elevation_ocean-%s_land-%s_rivers-%s_wetland-
%s_lakes-%s'
%(oceanmethod,landmethod,upper_rivermethod,wetlandfile_name,lakefile_name
)
    elevationpath  = folders[4] + elev_name
    print(elevationpath,elev_name)
    # try: os.remove("%s/%s_%s.tif" %(elevationpath,delta,elev_name))
    # except:''
    if os.path.isfile("%s/%s_%s_%s.tif"
%(elevationpath,delta,elev_name,xres))==False:

#########################################################################
##
        ###################### Landcover Classification
#########################

#########################################################################
##
        ##  Import Global Mangrove Watch and Daniel's Landcover
Classifications
```

```
#########################################################################
##
        ############################ Final Masks
################################

#########################################################################
##
        #print('#################### Rasterizing land, ocean, lake, and
river polygons')
        # if os.path.isfile('%s%s_lands.tif' %(folders[1],delta))==False:
        #     os.system('gdal_rasterize -burn 1 -tr %s %s -te %s %s %s %s
-tap %s%s_water_connected_and_lakes.shp %s%s_watermask.tif -co
COMPRESS=DEFLATE'
%(xres,yres,ulx,lry,lrx,uly,folders[7],delta,folders[8],delta))
        #     os.system('gdal_rasterize -burn 1 -tr %s %s -te %s %s %s %s
-tap %s%s_lands.shp %s%s_lands.tif'
%(xres,yres,ulx,lry,lrx,uly,folders[7],delta,folders[1],delta))
        #     os.system('gdal_rasterize -burn 1 -tr %s %s -te %s %s %s %s
-tap %s%s_oceans.shp %s%s_oceans.tif'
%(xres,yres,ulx,lry,lrx,uly,folders[7],delta,folders[1],delta))
        #     os.system('gdal_rasterize -burn 1 -tr %s %s -te %s %s %s %s
-tap %s%s_rivers.shp %s%s_rivers.tif'
%(xres,yres,ulx,lry,lrx,uly,folders[7],delta,folders[1],delta))
        print('#################### Loading Ocean, River, Land, and Land
Masks ')
        isocean = rasterio.open('%s%s_fulloceans_%s.tif'
%(folders[8],delta,xres)).read(1)
        isocean = np.where(isocean ==1,1.,0.)
        #isnearshore = rasterio.open('%s%s_nearshore_%s.tif'
%(folders[8],delta,xres)).read(1)
        #isnearshore = np.where(isnearshore ==1,1.,0.)
        isriver = rasterio.open('%s%s_rivers_%s.tif'
%(folders[8],delta,xres)).read(1)
        isriver = np.where(isriver ==1,1.,0.)
        try:
            islake = rasterio.open('%s%s_lakes_%s.tif'
%(folders[8],delta,xres)).read(1)
            islake = np.where(islake==1,1.,0.)
        except: islake = np.where(ref_array,0,0)
        island = rasterio.open('%s%s_lands_%s.tif'
%(folders[8],delta,xres)).read(1)
        island = np.where(island ==1,1.,0.)


#########################################################################
##
        ######################## Assign Manning Roughness
#########################

#########################################################################
##
        ## Manning roughness based off of lookup table
```

```
        mangrove = gdal.Open('%s%s_GMW_%s.tif'
%(folders[8],delta,xres)).ReadAsArray()
        if landcovermethod == 'Copernicus':
            mangrove_class = 9
        elif landcovermethod == 'WorldCover':
            mangrove_class = 95

        landcover = rasterio.open('%s%s_%s_%s.tif'
%(folders[8],delta,landcovermethod,xres)).read(1).astype('int')
        landcover[landcover==-9999] = 0
        landcover = np.where((isriver==1) | (isocean==1) | (islake==1)
,1,landcover)#|(isnearshore==1),1,landcover)
        landcover = np.where(mangrove==1,mangrove_class,landcover)


        island = np.where((island==0) | (isriver==1) | (islake==1) |
(iswetland==1) | (isocean==-1) ,0.,1)#|(isnearshore==1),0.,1.)
        upper = np.where((distance>riverBoundary),1,0)*isriver# &
(widths<max_river_width),1,0)*isriver
        #upper_wide = np.where((distance>riverBoundary) &
(widths>=max_river_width),1,0)*isriver
        lower = np.where((distance<=riverBoundary),1,0)*isriver

        print('\n[Step %s][Make_Model_Foundation][Output all land/water
types] ......\n'%(step))

        water_type =
np.where(isocean==1,1,np.where(upper==1,2,np.where(lower==1,3,np.where(is
wetland==1,4,np.where(islake==1,5,np.where(island==1,6,6))))))
        with rasterio.open("%s%s_water_type.tif" %(folders[8],delta),'w',
**save_profile) as dst:
            dst.write_band(1,water_type.astype('float64'))

        # with rasterio.open("%s%s_lands.tif" %(folders[8],delta),'w',
**save_profile) as dst:
        #     dst.write_band(1,island)"


#########################################################################
#####
        ######################### Make Manning Roughness File
########################

#########################################################################
#####
        print('\n[Step %s][Make_Model_Foundation][Fetching Manning
coefficients LUT] .....\n'%(step))
        if landcovermethod == 'Copernicus':
            lut = np.genfromtxt(build_path +
'/configs/Copernicus_manningLUT.csv',delimiter = ',')

np.savetxt('%s%s_%s_manningLUT.csv'%(folders[8],delta,landcovermethod),lu
t,delimiter=',')
        elif landcovermethod == 'WorldCover':
```

```
            lut = np.genfromtxt(build_path +
'/processing/configs/WorldCover_manningLUT.csv',delimiter = ',')

np.savetxt('%s%s_%s_manningLUT.csv'%(folders[8],delta,landcovermethod),lu
t,delimiter=',')
        else:
            landcovermethod = 'User'
            lut = np.genfromtxt("%s%s_manningLUT.csv"
%(folders[0],delta),delimiter=',')

np.savetxt('%s%s_%s_manningLUT.csv'%(folders[8],delta,landcovermethod),lu
t,delimiter=',')

        print('###################### Manning roughness coefficients set
according to ManningLUT.csv')
        manning = lut[landcover]
        frict_name = 'manning'
        save_profile['dtype'] = 'float64'
        with rasterio.open("%s%s_%s_%s.tif"
%(folders[8],delta,frict_name,landcovermethod),'w', **save_profile) as
dst:
            dst.write_band(1,manning.astype('float64'))


##########################################################################
#
        ########################## River Centerline
##############################

##########################################################################
#
        centerline = gdal.Open('%s%s_river_centerline_%sx%s.tif'
%(folders[8],delta,xres,pixel_step))
        drv = gdal.GetDriverByName('GTiff')
        dst_ds = drv.Create('%s%s_river_proximity_%sx%s.tif'
%(folders[8],delta,xres,pixel_step),
                            centerline.RasterXSize,
centerline.RasterYSize, 1,
                            gdal.GetDataTypeByName('Float32'))
        dst_ds.GetRasterBand(1).SetNoDataValue( -9999 )
        dst_ds.SetGeoTransform( centerline.GetGeoTransform() )
        dst_ds.SetProjection( centerline.GetProjectionRef() )
        dstband = dst_ds.GetRasterBand(1)
        srcband = centerline.GetRasterBand(1)
        test =
gdal.ComputeProximity(srcband,dstband,["VALUES=1","DISTUNITS=GEO"])
        srcband = None
        dstband = None
        src_ds = None
        dst_ds = None
        proximity = gdal.Open('%s%s_river_proximity_%sx%s.tif'
%(folders[8],delta,xres,pixel_step)).ReadAsArray()# * (widths>0)
        centerline = centerline.ReadAsArray()
        centerline = np.where(centerline==0,np.nan,centerline)
```

```
        centerline = np.ma.masked_invalid(centerline)
        x = np.arange(0,centerline.shape[1])
        y = np.arange(0,centerline.shape[0])
        xx,yy = np.meshgrid(x,y)
        x1 = xx[~centerline.mask]
        y1 = yy[~centerline.mask]

        segments = gpd.read_file('%s%s_segments_%sx%s.shp'
%(folders[7],delta,xres,pixel_step))
        prox_widths = segments.join(pd.DataFrame(zonal_stats(segments,
'%s%s_river_proximity_%sx%s.tif'
%(folders[8],delta,xres,pixel_step),stats="max")))
        prox_widths['newwidth'] = prox_widths['max']*2
        prox_widths = prox_widths[prox_widths['DN']!=0]
        try:
            os.remove('%s%s_widthsfromproximity_%sx%s.shp'
%(folders[7],delta,xres,pixel_step))
        except: ''
        prox_widths.to_file('%s%s_widthsfromproximity_%sx%s.shp'
%(folders[7],delta,xres,pixel_step), driver='ESRI Shapefile')
        os.system('gdal_rasterize -a newwidth -ts %s %s -te %s %s %s %s
'\
                  '%s%s_widthsfromproximity_%sx%s.shp
%s%s_widthsfromproximity_%sx%s.tif -co COMPRESS=DEFLATE'

%(distance.shape[1],distance.shape[0],ulx,lry,lrx,uly,folders[7],delta,xr
es,pixel_step,folders[8],delta,xres,pixel_step))
        widths = rasterio.open('%s%s_widthsfromproximity_%sx%s.tif'
%(folders[8],delta,xres,pixel_step)).read(1)

        ## Make parabolic cross section based on max depth from width-
depth power law
        def
make_parabolic_depths(maxdepths,widths,proximity,area_to,eqn):
            parabola_depths =
((maxdepths/((widths/2)**2))*(proximity**2))-maxdepths
            plt.imshow(parabola_depths,vmin=-10,vmax=10)
            parabola_depths[parabola_depths<=0] = 0
            fillers =
np.where(area_to==0,np.nan,np.where(parabola_depths==0,0,np.where(abs(par
abola_depths)>abs(max_river_depth),0,1)))
            fillers2 = _fillnodata(parabola_depths,fillers,500)
            river_depth = np.where(area_to==1, fillers2,0)
            return river_depth

#########################################################################
#
        ######################### Final Elevation File
############################

#########################################################################
#
        print('\n[Step %s][Make_Model_Foundation][Building DEM]
.....\n'%(step))
```

```
        print('\n[Step %s][Make_Model_Foundation][Building DEM][Building
ocean bathymetry] .....\n'%(step))

        ## Ocean
        print('#################### Ocean from %s_bathy.tif' %(delta))
        bathy = rasterio.open("%s%s_bathy_%s.tif"
%(folders[8],delta,xres)).read(1)
        bathy = np.where(bathy==-9999,np.nan,bathy)
        temp_ocean =
np.where((isocean==1),scipy.ndimage.gaussian_filter(bathy,1),0)
        #temp_nearshore =
np.where((isnearshore==1),scipy.ndimage.gaussian_filter(bathy,1),0)
        temp_river =
np.where((isriver==1),scipy.ndimage.gaussian_filter(bathy,1),0)

        fillers =
np.where(temp_ocean<max_ocean_depth,0,np.where(temp_ocean>=-
0.9,0,temp_ocean)) ## 0 will be filled, nans will be excluded
        ocean_elev =
np.where(isocean==1,_fillnodata(temp_ocean,fillers,500),0)

        #fillers =
np.where(temp_nearshore<max_nearshore_depth,0,np.where(temp_nearshore>=-
0.9,0,temp_nearshore)) ## 0 will be filled, nans will be excluded
        #nearshore_elev =
np.where(isnearshore==1,_fillnodata(temp_nearshore+temp_ocean,fillers,500
),0)


        print('\n[Step %s][Make_Model_Foundation][Building DEM][Building
river bathymetry] .....\n'%(step))
        ## River
        # fillers =
np.where(temp_river<max_river_depth,0,np.where(temp_river>=-
0.9,0,temp_river)) ## 0 will be filled, nans will be excluded
        # temp_river = _fillnodata(temp_river,fillers,500)
        temp_river =
np.where(isriver==0,np.nan,np.where((temp_river<max_river_depth) |
(temp_river>=-0.9),np.nan,temp_river))


        print('#################### Upper/Lower river boundary == %s'
%(riverBoundary))

        if upper_rivermethod == 'default' or upper_rivermethod ==
'GEBCO':
            maxdepths = np.where(upper==1,scipy.ndimage.median_filter(-
temp_river,3),0)
            maxdepths = np.where(upper==1,maxdepths,np.nan)
            upper_river_elev
=np.where(upper==1,np.where(abs(maxdepths)<abs(max_river_depth),0-
abs(maxdepths),np.nan),0)
            print('\n#################### Upper river elevation set
with:  GEBCO')
```

```
        elif (upper_rivermethod in
['wdlinear','wdpower','distanceslope']) | (upper_rivermethod[:8] ==
'constant'):
            if upper_rivermethod[:8] == 'constant':
                maxdepths = float(upper_rivermethod[9:])*upper
                print('#################### An average depth of %sm will
be carved below the NASADEM water level for the upper river'
%(upper_rivermethod[5:]))
            elif upper_rivermethod == 'wdlinear':
                slope =
parameters['WD_LINEARSLOPE_upper'][0].astype('float64')
                intercept =
parameters['WD_LINEARINTERCEPT_upper'][0].astype('float64')
                print('#################### Upper river depth is
determined with a linear relationship Depth = %s*Width + %s based on
Andreadis et al. 2013' %(slope,intercept))
                maxdepths =
np.where(widths<10000,(widths*slope+intercept),np.where(temp_river>=-
0.9,-temp_river,np.nan))*upper
            elif upper_rivermethod == 'wdpower':
                coeff_a =
parameters['WD_POWERA_upper'][0].astype('float64')
                coeff_b =
parameters['WD_POWERB_upper'][0].astype('float64')
                print('#################### Upper river depth is
determined with a power law relationship: depth = %s*width^%s'
%(coeff_a,coeff_b))#'Depth = %s*Width + %s based on Andreadis et al.
2013' %(slope,intercept))
                maxdepths =
np.where(upper==1,np.where(widths<max_river_width,(coeff_a)*(widths**coef
f_b),(coeff_a)*(max_river_width**coeff_b)),np.nan)
            elif upper_rivermethod == 'distanceslope':
                bedslope =
parameters['BEDSLOPE_upper'].astype('float64')[0]
                bedintercept =
parameters['BEDINTERCEPT_upper'].astype('float64')[0]
                print('#################### Upper river dpeth is
determined by distance upstream: depth = %s + %s x distance'
%(bedintercept,bedslope))
                #print('#################### Bed slope is applied to the
lower river with \nBed Slope is %s from 0m (river mouth) to %sm \nand %s
from %sm to the upstream boundary' %(bed_slope1, slope_boundary,
bed_slope2, slope_boundary))
                distance2 = np.where(np.isnan(distance),0,distance)
                #rivermouth_elev =
np.nanmean(np.where(distance2<=(np.nanmin(distance)+10),np.where(distance
2>0,bathy,np.nan),np.nan))
                maxdepths = -
np.where(upper==1,distance2*bedslope+bedintercept,np.nan)
            fillers =
np.where(upper==1,np.where(maxdepths>max_river_depth,0,np.where(maxdepths
<=0.9,0,1)),0) ## 0 will be filled, nans will be excluded
            maxdepths2 = _fillnodata(maxdepths,fillers,500)
            maxdepths = np.where(upper==1,-maxdepths2,np.nan)
```

```
            # with warnings.catch_warnings():
            #     warnings.simplefilter("ignore",
category=RuntimeWarning)
            #     average_depth =
np.nanmean(np.array([temp_river,maxdepths]),axis=0)
            # fillers =
np.where(upper==1,np.where(average_depth<max_river_depth,0,np.where(avera
ge_depth>=-0.9,0,1)),0) ## 0 will be filled, nans will be excluded
            # average_depth = _fillnodata(average_depth,fillers,500)
            # average_depth =
np.where(upper==1,scipy.ndimage.gaussian_filter(average_depth,3),0)


            centerline_nan =
np.where(np.isnan(centerline),np.nan,maxdepths)
            #maxdepths_full =
np.where(isriver==1,griddata((x1,y1),centerline_nan[~centerline.mask].rav
el(),(xx,yy),method = 'nearest'),0)
            riverdepth =
make_parabolic_depths(maxdepths,widths,proximity,upper,upper_rivermethod)
            upper_river_elev =
np.where(upper==1,np.where(abs(riverdepth)<abs(max_river_depth),0-
abs(riverdepth),np.nan),0)
        else:
            while True:
                try:
                    print('###################### Upper river elevation
set with:  %s' %(upper_rivermethod))
                    os.system('gdalwarp -overwrite -tr %s %s -tap
%s%s.tif %s%s_upperriver.tif -t_srs EPSG:%s -te %s %s %s %s -srcnodata -
9999 -dstnodata -9999'  %(xres,
yres,folders[0],upper_rivermethod,folders[1],upper_rivermethod,EPSG,ulx,l
ry,lrx,uly))
                except AttributeError:
                    print('File not found \n')
                    upper_rivermethod = input('What file would you like
to use for the upper river elevation? Should be saved in
/User_Defined_Files subfolder ')
                riverbathy = gdal.Open("%s%s_upperriver.tif"
%(folders[2],upper_rivermethod)).ReadAsArray()
                #upper_river_elev
=np.where(upper==1,np.where((abs(riverbathy)<abs(max_river_depth)) &
(riverbathy<-0.9),riverbathy,np.nan),0)
                upper_river_elev =np.where(upper==1,riverbathy,0)
                break


        if lower_rivermethod == 'default' or lower_rivermethod ==
'GEBCO':
            maxdepths = np.where(lower==1,scipy.ndimage.median_filter(-
temp_river,3),0)
            maxdepths = np.where(lower==1,maxdepths,np.nan)
```

```
            lower_river_elev =
np.where(lower==1,np.where(abs(maxdepths)<abs(max_river_depth),0-
abs(maxdepths),np.nan),0)
            print('\n##################### Lower river elevation set
with:  GEBCO')
        elif (lower_rivermethod in
['wdlinear','wdpower','distanceslope','plane']) | (lower_rivermethod[:8]
== 'constant'):
            if lower_rivermethod[:8] == 'constant':
                maxdepths = float(lower_rivermethod[9:])*lower
                print('##################### An average depth of %sm will
be carved below the NASADEM water level for the lower river'
%(lower_rivermethod[5:]))
            elif lower_rivermethod == 'wdlinear':
                #slope =
parameters['WD_LINEARSLOPE_lower'][0].astype('float64')
                #intercept =
parameters['WD_LINEARINTERCEPT_lower'][0].astype('float64')
                riverBoundaryWidth =
np.nanmean(np.where((distance<=riverBoundary+100)&(distance>=riverBoundar
y-100),widths,np.nan))
                riverBoundaryDepth =
np.nanmax(np.where((distance<=riverBoundary+100)&(distance>=riverBoundary
-100),-upper_river_elev,np.nan))

#min(max_river_depth,coeff_a*(riverBoundaryWidth**coeff_b))#np.nanmean(np
.where((distance<=riverBoundary+10)&(distance>=riverBoundary-
10),riverdepth,np.nan))
                oceanBoundaryWidth =
np.nanmax(np.where((distance<=1200)&(distance>=1000),widths,np.nan))
                oceanBoundaryDepth =
np.nanmean(np.where((distance<=1200)&(distance>=1000)&(-
nearshore_elev!=0),-nearshore_elev,np.nan))
                slope = (oceanBoundaryDepth -
riverBoundaryDepth)/(oceanBoundaryWidth-riverBoundaryWidth)
                intercept = riverBoundaryDepth-riverBoundaryWidth*slope
                intercept = oceanBoundaryDepth-oceanBoundaryWidth*slope

#print(riverBoundaryWidth,riverBoundaryDepth,oceanBoundaryWidth,oceanBoun
daryDepth)
                print('##################### Lower river depth is
determined with a linear relationship: Depth = %s x Width + %s based on
linear relationship between width and depth using river mouth and upper
river data points' %(slope,intercept))
                maxdepths =
np.where(lower==1,np.where(widths<=oceanBoundaryWidth,widths*slope+interc
ept,oceanBoundaryWidth*slope+intercept),np.nan)
            elif lower_rivermethod == 'wdpower':
                coeff_a =
parameters['WD_POWERA_lower'][0].astype('float64')
                coeff_b =
parameters['WD_POWERB_lower'][0].astype('float64')
```

```
                print('##################### Lower river depth is
determined with a power law relationship: depth = %s x width^%s'
%(coeff_a,coeff_b))#'Depth = %s*Width + %s based on Andreadis et al.
2013' %(slope,intercept))
                maxdepths =
np.where(lower==1,np.where(widths<max_river_width,(coeff_a)*(widths**coef
f_b),np.where(temp_river<-
0.9,temp_river,(coeff_a)*(max_river_width**coeff_b))),np.nan)
            elif lower_rivermethod == 'distanceslope':
                bedslope =
parameters['BEDSLOPE_lower'].astype('float64')[0]
                bedintercept =
parameters['BEDINTERCEPT_lower'].astype('float64')[0]
                print('##################### Lower river dpeth is
determined by distance upstream: depth = %s + %s x distance'
%(bedintercept,bedslope))
                #print('##################### Bed slope is applied to the
lower river with \nBed Slope is %s from 0m (river mouth) to %sm \nand %s
from %sm to the upstream boundary' %(bed_slope1, slope_boundary,
bed_slope2, slope_boundary))
                distance2 = np.where(np.isnan(distance),0,distance)
                #rivermouth_elev =
np.nanmean(np.where(distance2<=(np.nanmin(distance)+10),np.where(distance
2>0,bathy,np.nan),np.nan))
                maxdepths = -
np.where(lower==1,distance2*bedslope+bedintercept,np.nan)
            elif lower_rivermethod == 'plane':
                riverBoundaryDistance = riverBoundary
                riverBoundaryDepth =
np.nanmax(np.where((distance<=riverBoundary+100)&(distance>=riverBoundary
-100),-upper_river_elev,np.nan))

#min(max_river_depth,coeff_a*(riverBoundaryWidth**coeff_b))#np.nanmean(np
.where((distance<=riverBoundary+10)&(distance>=riverBoundary-
10),riverdepth,np.nan))
                oceanBoundaryDistance =
1200#np.nanmax(np.where((distance<=1200)&(distance>=100),distance,np.nan)
)
                oceanBoundaryDepth =
np.nanmean(np.where((distance<=1200)&(distance>=100)&(bathy!=0)&(bathy>ma
x_nearshore_depth),-bathy,np.nan))
                slope = (oceanBoundaryDepth -
riverBoundaryDepth)/(oceanBoundaryDistance-riverBoundaryDistance)
                intercept = riverBoundaryDepth-
riverBoundaryDistance*slope
                intercept = oceanBoundaryDepth-
oceanBoundaryDistance*slope

#print(riverBoundaryWidth,riverBoundaryDepth,oceanBoundaryWidth,oceanBoun
daryDepth)
                print('##################### River boundary: Distance =
%sm Depth = %sm \n##################### Ocean boundary: Distance = %sm
Depth = %sm'
```

```
%(riverBoundaryDistance,riverBoundaryDepth,oceanBoundaryDistance,oceanBou
ndaryDepth))
                print('#################### Lower river depth is
determined with a linear relationship: Depth = %s x Distance + %s based
on linear relationship between width and depth using river mouth and
upper river data points' %(slope,intercept))
                maxdepths =
np.where(lower==1,np.where(distance>=oceanBoundaryDistance,distance*slope
+intercept,oceanBoundaryDistance*slope+intercept),np.nan)

            fillers =
np.where(lower==1,np.where(maxdepths>max_river_depth,0,np.where(maxdepths
<=0.9,0,1)),0) ## 0 will be filled, nans will be excluded
            maxdepths2 = _fillnodata(maxdepths,fillers,500)
            maxdepths = np.where(lower==1,-maxdepths2,np.nan)
            # with warnings.catch_warnings():
            #     warnings.simplefilter("ignore",
category=RuntimeWarning)
            #     average_depth =
np.nanmean(np.array([temp_river,maxdepths]),axis=0)
            # #fillers =
np.where(lower==1,np.where(average_depth<max_river_depth,0,np.where(avera
ge_depth>=-0.9,0,1)),0) ## 0 will be filled, nans will be excluded
            # fillers =
np.where(lower==1,np.where(average_depth<max_river_depth,0,np.where(avera
ge_depth>=-0.9,0,1)),0) ## 0 will be filled, nans will be excluded
            # average_depth = _fillnodata(average_depth,fillers,500)
            # average_depth =
np.where(lower==1,scipy.ndimage.gaussian_filter(average_depth,3),0)

            centerline_nan =
np.where(np.isnan(centerline),np.nan,maxdepths)
            centerline_nan2 =
scipy.ndimage.gaussian_filter(centerline_nan,10)
            centerline_nan =
np.where(np.isnan(centerline),np.nan,centerline_nan2)
            riverdepth =
make_parabolic_depths(maxdepths,widths,proximity,lower,lower_rivermethod)
            lower_river_elev =
np.where(lower==1,np.where(abs(riverdepth)<abs(max_river_depth),0-
abs(riverdepth),np.nan),0)
        else:
            while True:
                try:
                    print('#################### Lower river elevation
set with:  %s' %(lower_rivermethod))
                    os.system('gdalwarp -overwrite -tr %s %s -tap
%s%s.tif %s%s_lowerriver.tif -t_srs EPSG:%s -te %s %s %s %s -srcnodata -
9999 -dstnodata -9999 co COMPRESS=DEFLATE'  %(xres,
yres,folders[0],lower_rivermethod,folders[1],lower_rivermethod,EPSG,ulx,l
ry,lrx,uly))
                except AttributeError:
                    print('File not found \n')
```

```
                    lower_rivermethod = input('What file would you like
to use for the lower river elevation? Should be saved in
/User_Defined_Files subfolder ')
                riverbathy = gdal.Open("%s%s_lowerriver.tif"
%(folders[1],lower_rivermethod)).ReadAsArray()
                #lower_river_elev
=np.where(lower==1,np.where((abs(riverbathy)<abs(max_river_depth)) &
(riverbathy<=-0.9),riverbathy,np.nan),0)
                lower_river_elev =np.where(lower==1,riverbathy,0)
                break


        river_elev = upper_river_elev + lower_river_elev
        river_elev = np.where(isriver==1,river_elev,np.nan)
        #river_elev = scipy.ndimage.median_filter(river_elev,3)

        sigma=3.0                   # standard deviation for Gaussian
kernel
        truncate=4.0                # truncate filter at this many sigmas

        river_elev[river_elev>2]=np.nan              # ...with NaNs for
testing

        V=river_elev.copy()
        V[np.isnan(river_elev)]=0
        VV=scipy.ndimage.gaussian_filter(V,sigma=sigma,truncate=truncate)

        W=0*river_elev.copy()+1
        W[np.isnan(river_elev)]=0
        WW=scipy.ndimage.gaussian_filter(W,sigma=sigma,truncate=truncate)

        river_elev=np.where(isriver==1,VV/WW,0)

        with rasterio.open("%s/%s_temp.tif" %(folders[1],delta),'w',
**save_profile) as dst:
            dst.write_band(1,river_elev)


        ## Land
        print('\n[Step %s][Make_Model_Foundation][Building DEM][Building
land topography] .....\n'%(step))
        print('#################### Land from %s%s_topo.tif'
%(folders[8],delta))
        topo = rasterio.open("%s%s_topo_%s.tif"
%(folders[8],delta+'_'+landmethod,xres)).read(1)
        land_elev = np.where((topo==-9999) | (island!=1),0.,topo)

        ##  Lake
        print('\n[Step %s][Make_Model_Foundation][Building DEM][Building
lake bathymetry] .....\n'%(step))
        if lakemethod == 'default' or lakemethod == 'GEBCO':
            temp_lake = np.where(islake==1,bathy,0)
```

```
            fillers =
np.where(temp_lake<max_ocean_depth,0,np.where(temp_lake>=-0.9,0,1)) ## 0
will be filled, nans will be excluded
            temp_lake = _fillnodata(temp_lake,fillers,500)
            lake_elev =
np.where(islake==1,scipy.ndimage.gaussian_filter(temp_lake,1),0)
            lakefile_name = 'GEBCO'
            print('##################### Lake elevation set with:
GEBCO')
        elif lakemethod[:8] == 'constant':
            constantLakeDepth = float(lakemethod[9:])
            lake_elev = islake*(topo-constantLakeDepth)
            lakefile_name = 'uni%sm' %(str(int(constantLakeDepth)))
            print('##################### Lake Depth set with:  %sm
uniform' %(constantLakeDepth))
        else:
            while True:
                try:
                    print('##################### Lake depth set with:
\n##################### %s ' %(lakemethod))
                    os.system('gdalwarp -overwrite -tr %s %s -tap
%s%s%s.tif %s%s%s_lake.tif -t_srs EPSG:%s -te %s %s %s %s -srcnodata -
9999 -dstnodata -9999 co COMPRESS=DEFLATE'  %(xres,
yres,deltapath,folders[3],lakemethod,deltapath,folders[2],lakemethod,EPSG
,ulx,lry,lrx,uly))
                    lake_elev = gdal.Open("%s_land.tif"
%(lakemethod)).ReadAsArray()
                except AttributeError:
                    print('File not found \n')
                    lakefile = input('What file would you like to use?
Should be saved in /User_Defined_Files subfolder ')
                else: break

        ## Wetlands
        print('\n[Step %s][Make_Model_Foundation][Building DEM][Building
wetland topography] .....\n'%(step))
        if wetlandmethod =='default' or wetlandmethod == 'NASADEM':
            wetland_elev = iswetland*topo
            print('##################### Wetland elevation set with:
NASADEM')
        elif wetlandmethod[:8] == 'constant':
            constantWetlandElev = float(wetlandmethod[9:])
            wetland_elev = np.where(iswetland==1,constantWetlandElev,0)
            wetlandfile_name = 'uni%sm' %(str(int(constantWetlandElev)))
            print('##################### Wetland elevation set with:  %sm
uniform' %(constantWetlandElev))
        else:
             while True:
                try:
                    print('##################### Wetland depth set with:
\n##################### %s ' %(wetlandmethod))
                    os.system('gdalwarp -overwrite -tr %s %s -tap
%s%s%s.tif %s%s%s_wetland.tif -t_srs EPSG:%s -te %s %s %s %s -srcnodata -
9999 -dstnodata -9999 co COMPRESS=DEFLATE'  %(xres,
```

```
yres,deltapath,folders[3],wetlandmethod,deltapath,folders[2],wetlandmetho
d,EPSG,ulx,lry,lrx,uly))
                    wetland_elev = gdal.Open("%s%s_wetland.tif"
%(folders[0],wetlandmethod)).ReadAsArray()
                except AttributeError:
                    print('File not found \n')
                    wetlandfile = input('What file would you like to use?
Should be saved in /User_Defined_Files subfolder ')
                else: break

        ####  Full elevation raster
        river_ocean_elev =np.where((isriver==1) | (isocean==1)
,scipy.ndimage.median_filter(river_elev+ocean_elev,3),0)#+nearshore_elev,
3),0)
        fillers = np.where((isriver==1) | (isocean==1)
,np.where(river_ocean_elev<max_ocean_depth,0,np.where(river_ocean_elev>=-
0.9,0,1)),np.nan) ## 0 will be filled, nans will be excluded
        river_ocean_elev2 = np.where((isriver==1) | (isocean==1)
,_fillnodata(river_ocean_elev,fillers,500),0)
        elevation = (river_ocean_elev2+land_elev+lake_elev+wetland_elev)
        fillers =
np.where(elevation<max_ocean_depth,0,np.where(np.isnan(elevation),0,1))
        elevation2 = _fillnodata(elevation,fillers,500)

        # import scipy
        # # avg_200_slope = np.nanmean((np.where((distance<=10000) &
(distance>=50),riverdepth/distance,np.nan)))
        # o_lr_boundary = np.where(isbuffer==0,0,np.nanmean(np.array([
np.where(lower_river_elev==0,np.nan,lower_river_elev), ocean_elev ]),
axis=0 ))
        # o_lr_smoothed = scipy.signal.medfilt(o_lr_boundary,9)
        # lr_ur_boundary = np.where((distance<=riverBoundary+2500) &
(distance>=riverBoundary-2500),np.nanmean(np.array([ upper_river_elev,
lower_river_elev ]), axis=0 ),0)
        # lr_ur_smoothed = scipy.signal.medfilt(lr_ur_boundary,9)
        # new_elevation =
np.where(isbuffer==1,o_lr_smoothed,np.where((distance<=riverBoundary+2500
) & (distance>=riverBoundary-2500),lr_ur_smoothed,elevation2))
        # o_lr_boundary_mean = np.nanmean(np.array([
np.where(lower_river_elev==0,np.nan,lower_river_elev), ocean_elev ]),
axis=0 )
        # lr_ur_boundary_mean = np.nanmean(np.array([ upper_river_elev,
lower_river_elev ]), axis=0)
        # new_elevation =
np.where(isbuffer==1,o_lr_boundary_mean,np.where((distance<=riverBoundary
+2500) & (distance>=riverBoundary-2500),lr_ur_boundary_mean,elevation2))
        # new_elevation =
np.where(isriver+isocean>0,scipy.signal.medfilt(new_elevation,31),new_ele
vation)
        print('\n[Step %s][Make_Model_Foundation][Building DEM][Put it
all together into final DEM] .....\n'%(step))
        print('#################### Elevation File: \n%s' %(elev_name))

        #elevationame = '%s_%s' %(delta,elev_name)
```

```
        try: os.mkdir(elevationpath)
        except: ''
        print('################################## Working directory will
be: \n%s' %('/'+elevationpath))
        with rasterio.open("%s/%s_%s.tif"
%(folders[4],elev_name,xres),'w', **save_profile) as dst:
            dst.write_band(1,elevation2.astype('float64'))
        with rasterio.open("%s/%s_landcover_final_%s.tif"
%(folders[8],delta,xres),'w', **save_profile) as dst:
            dst.write_band(1,landcover)



##########################################################################
######
        ################################## Save Plots
##################################
        print('\n[Step %s][Make_Model_Foundation][Make some figures]
.......\n'%(step))

        fig, ((ax1,ax2,ax3),(ax4,ax5,ax6)) =
plt.subplots(ncols=3,nrows=2,figsize=(15,10))
        im1 = ax1.imshow(ocean_elev,vmin=-40,vmax=0,cmap = 'viridis')
        ax1.set_title('Ocean Bathymetry')
        im2 = ax2.imshow(upper_river_elev,vmin=-20,vmax=0,cmap =
'viridis')
        ax2.set_title('Upper River Elevation')
        im3 = ax3.imshow(lower_river_elev,vmin=-20,vmax=0,cmap =
'viridis')
        ax3.set_title('Lower River Elevation')
        im4 = ax4.imshow(land_elev,vmin=0,vmax=5,cmap = 'viridis')
        ax4.set_title('Land Elevation')
        im5 = ax5.imshow(wetland_elev,vmin=-1,vmax=1,cmap = 'viridis')
        ax5.set_title('Wetland Elevation')
        im6 = ax6.imshow(lake_elev,vmin=-2,vmax=0,cmap = 'viridis')
        ax6.set_title('Lake Elevation')
        #cbar_ax = fig.add_axes([0.93, 0.1, 0.01, 0.8])
        #cbar = fig.colorbar(im3, cax=cbar_ax)
        fig.tight_layout()

        #
        # fig, axes = plt.subplots(figsize=[10,10],facecolor = 'w',
edgecolor = 'k')
        # plt.imshow(ocean_elev)
        # plt.title('Ocean Bathymetry')
        # plt.clim(-20, 20)
        # plt.colorbar(orientation='horizontal',shrink = 0.5,pad =
0.01,aspect = 40)
        # ax = plt.gca()
        # bar = AnchoredSizeBar(ax.transData, 100, '3000m', 'lower
center',pad=0.1,color='red',frameon=False)#, pad = 0.1, color = 'white',
frameon=True,size_vertical = 1)
        # ax.add_artist(bar)
        # ax.axes.xaxis.set_visible(False)
```

```
        # ax.axes.yaxis.set_visible(False)
        # plt.tight_layout()
        # plt.savefig("%s%s_%s.png" %(folders[9],delta,elev_name))
        # plt.savefig("%s/summary_figures/bathy/%s_bathy.png"
%(path,delta))


        fig = plt.figure(figsize = (10,10), facecolor = 'w', edgecolor =
'k')
        plt.imshow(watermask,cmap='Greys',interpolation='nearest')
        plt.title('Water Mask')
        ax = plt.gca()
        bar = AnchoredSizeBar(ax.transData,  100, '3000m', 'lower
center',pad=0.1,color='red',frameon=False)#, pad = 0.1, color = 'white',
frameon=True,size_vertical = 1)
        ax.add_artist(bar)
        ax.axes.xaxis.set_visible(False)
        ax.axes.yaxis.set_visible(False)
        plt.tight_layout()
        plt.savefig("%s%s_wetlands.png" %(folders[9],delta))

        fig = plt.figure(figsize = (10,10), facecolor = 'w', edgecolor =
'k')
        cmap1 = cmap=plt.cm.get_cmap('gist_ncar',
len(np.unique(landcover)))
        #cmap1 =
mpl.colors.ListedColormap(['white','blue','green','magenta','cyan','black
','yellow','red','turquoise','lime'])
        plt.imshow(landcover,cmap=cmap1)
        #plt.clim(0,9)
        plt.colorbar(orientation='horizontal',shrink = 0.5,pad =
0.01,aspect = 40)
        plt.title('Landcover Type')
        ax = plt.gca()
        bar = AnchoredSizeBar(ax.transData,  100, '3000m',  'lower
center',pad=0.1,color='red',frameon=False)#, pad = 0.1, color = 'white',
frameon=True,size_vertical = 1)
        ax.add_artist(bar)
        ax.axes.xaxis.set_visible(False)
        ax.axes.yaxis.set_visible(False)
        plt.tight_layout()
        plt.savefig("%s%s_landcover.png" %(folders[9],delta))

        # fig = plt.figure(figsize = (10,10), facecolor = 'w', edgecolor
= 'k')
        # cmap2 = mpl.cm.get_cmap('nipy_spectral')
        # cmap3 =
mpl.colors.ListedColormap([cmap2(0.2),cmap2(0.4),cmap2(0.6),cmap2(0.8),cm
ap2(1)])
        # colormap = plt.imshow(water_type,cmap=cmap3)
        # plt.clim(1,5)
        # cbar = plt.colorbar(colormap, orientation='horizontal',shrink =
0.5,pad = 0.01,aspect = 40)
        # cbar.set_ticks([1.4,2.2,3.0,3.8,4.6])
```

```
        # cbar.set_ticklabels(["Ocean", "River", "Wetland", "Lake",
"Land"])
        # plt.title('Water/Land Type')
        # ax = plt.gca()
        # bar = AnchoredSizeBar(ax.transData, 100, '3000m',  'lower
center',pad=0.1,color='red',frameon=False)#, pad = 0.1, color = 'white',
frameon=True,size_vertical = 1)
        # ax.add_artist(bar)
        # ax.axes.xaxis.set_visible(False)
        # ax.axes.yaxis.set_visible(False)
        # plt.tight_layout()
        # plt.savefig("%s%s_riverlandocean.png" %(folders[9],delta))
        # plt.title('%s'%(delta.capitalize()))
        #
plt.savefig("%s/summary_figures/watermasks/%s_riverlandocean.png"
%(path,delta))


        # fig, ax, = plt.subplots(ncols=1, figsize=[10,10],facecolor =
'w', edgecolor = 'k')
        # plt.imshow(elevation2)
        # plt.title('Bathymetry (m)')
        # plt.clim(-20, 0)
        # plt.colorbar(orientation='horizontal',shrink = 0.5,pad =
0.01,aspect = 40)
        # ax = plt.gca()
        # bar = AnchoredSizeBar(ax.transData, 100, '3000m', 'lower
center',pad=0.1,color='red',frameon=False)#, pad = 0.1, color = 'white',
frameon=True,size_vertical = 1)
        # ax.add_artist(bar)
        # ax.axes.xaxis.set_visible(False)
        # ax.axes.yaxis.set_visible(False)
        # plt.tight_layout()
        # plt.savefig("%s/summary_figures/bathy/%s_bathy.png"
%(path,delta))

        plt.close('all')
        print('\n[Step %s][Make_Model_Foundation] Finished
.......\n'%(step))

    else:
        print('\n[Step %s][Make_Model_Foundation] SKIP.......\n'%(step))
        #elevationpath  = folders[4] + elev_name
        elevation2 = rasterio.open('%s/%s_%s.tif' %(folders[3],
elev_name,xres)).read(1)

        # fig, ax, = plt.subplots(ncols=1, figsize=[10,10],facecolor =
'w', edgecolor = 'k')
        # plt.imshow(elevation2)
        # plt.title('Topo/Bathy Elevation (m)')
        # plt.clim(-20, 20)
        # plt.colorbar(orientation='horizontal',shrink = 0.5,pad =
0.01,aspect = 40)
        # ax = plt.gca()
```

```
        # bar = AnchoredSizeBar(ax.transData, 100, '3000m', 'lower
center',pad=0.1,color='red',frameon=False)#, pad = 0.1, color = 'white',
frameon=True,size_vertical = 1)
        # ax.add_artist(bar)
        # ax.axes.xaxis.set_visible(False)
        # ax.axes.yaxis.set_visible(False)
        # plt.tight_layout()
        # plt.savefig("%s%s_%s.png" %(folders[9],delta,elev_name))
        # plt.savefig("%s/summary_figures/DEM/%s_%s.png"
%(path,delta,elev_name))

    return elevation2, elev_name+'_'+str(xres)


##########################################################################
###
##########################################################################
###
##########################################################################
###
## PART VIII
## OUTPUT: Boundary condition information
##########################################################################
#

def set_boundary_conditions(delta,folders,res,parameters,elev_name):
    step = 8

print('\n\n\n##########################################################################
#################################')
    print('################################[Step
%s][Set_Boundary_Conditions]################################'%(step))

print('##########################################################################
############################\n')

##########################################################################
##
    ###################### Import Config Parameters
############################

##########################################################################
##
    xres,yres = res,res
    EPSG = parameters['EPSG'][0]
# Coordinate System must be UTM

    extentpoly = gpd.read_file("%s%s_modeldomain.shp"
%(folders[7],delta))
    ulx,lry,lrx,uly = extentpoly.total_bounds   # Coordinates converted
to UTM coordinate system
    extentpoly_line = extentpoly.geometry.boundary
```

```
##########################################################################
######
    ######################### Set final files for model
############################
    # Set Boundaries. Open, tidal boundary determined as boundary with
lowest elevation.
    # Other boundaries are set as transmissive Bi2
    # Set Upstream Boundary Condition at largest river

    ## Boundaries: North = 0, East = 1, South = 2, West = 3
    print('\n[Step %s][Set_Boundary_Conditions][Determine boundary type
of each model side] .......\n'%(step))
    # sides =
{'id':[0,1,2,3],'geometry':[LineString([(ulx,uly),(lrx,uly)]),LineString(
[(lrx,uly),(lrx,lry)]),LineString([(ulx,lry),(lrx,lry)]),LineString([(ulx
,uly),(ulx,lry)])]}
    # df_line = gpd.GeoDataFrame(sides,columns=['id','geometry'])
    # df_line['geometry'] = df_line.geometry.buffer(1000)
    # df_line.crs = extentpoly.crs
    # boundaries =
['BoundaryType1','BoundaryType1','BoundaryType1','BoundaryType1'] #
Boundary types: Bi tides, Bi2 transmissive, but no stage set

    x,y = extentpoly_line.geometry[0].coords.xy
    xy = pd.DataFrame(list(zip(x,y)), columns=['LON', 'LAT'])

    geoms = []
    ids = []
    for xy1 in range(len(xy)-1):
        x1 = xy.iloc[xy1]['LON']
        y1 = xy.iloc[xy1]['LAT']
        x2 = xy.iloc[xy1+1]['LON']
        y2 = xy.iloc[xy1+1]['LAT']
        geoms.append(LineString([(x1,y1),(x2,y2)]))
        ids.append(xy1)
    sides = {'id':ids,'geometry':geoms}
    df_line = gpd.GeoDataFrame(sides,columns=['id','geometry'])
    df_line['geometry'] = df_line.geometry.buffer(500)
    df_line.crs = extentpoly.crs
    df_line['index'] = df_line.index

    print('\n[Step %s][Set_Boundary_Conditions][Largest river channel
along model edge = upstream boundary, river discharge conditions]
.......\n'%(step))
    rivers = gpd.read_file("%s%s_rivers_%s.shp" %(folders[7],delta,xres))
    rivers = gpd.overlay(rivers,extentpoly,how='intersection')

    intersections = gpd.overlay(df_line,rivers,how='intersection')
    intersections['mean'] = pd.DataFrame(zonal_stats(vectors =
intersections['geometry'],raster = "%s/%s_%s_%s.tif" %(folders[4] +
elev_name,delta,elev_name,xres),stats='mean',nodata = '-9999'))['mean']
    intersections['areadepth'] = intersections.area *
intersections['mean']
```

```
    try:
        #inlet =
intersections.loc[intersections['mean']==min(intersections['mean'])].iloc
[0]
        #inlet =
intersections.loc[intersections.area==max(intersections.area)]
        inlet = intersections.loc[intersections['areadepth'] ==
min(intersections['areadepth'])]
    except:
        print('no intersections between river and model boundary')
        upstreamX = 0
        upstreamY = 0
    else:
        upstreamX= int(inlet['geometry'].centroid.x)
        upstreamY= int(inlet['geometry'].centroid.y)
        print('##################### Discharge boundary conditions set at
%s,%s' %(upstreamX,upstreamY))

    #boundaries = ['BoundaryType1'] * (len(df_line)-1) # Boundary types:
Bi tides, Bi2 transmissive, but no stage set
    df_line['boundary'] = ['BoundaryType1'] * (len(df_line))
    if os.path.isfile('%s%s_tidebnd.shp' %(folders[0],delta)) == True:
        print('\n[Step %s][Set_Boundary_Conditions][%s_tidebnd.shp will
determine downstream boundary, tidal conditions] .......\n'%(step,delta))
        tide_bnd = gpd.read_file('%s%s_tidebnd.shp' %(folders[0],delta))
        tide_bnd = tide_bnd.to_crs(df_line.crs)
        tideboundary = gpd.overlay(df_line,tide_bnd,how='intersection')
        tide = tideboundary['index']
        #b#oundaries =
np.where(boundaries[.index].isin(tide),'BoundaryType2',boundaries)
        #boundaries =
np.where(df_line.iloc[tide],'BoundaryType2','BoundaryType1')
        df_line['boundary'].loc[tide] = 'BoundaryType2'

        tide_bnd = tide_bnd['geometry'].centroid
        tide_bnd = tide_bnd.to_crs("EPSG:4326")
        tidex= float(tide_bnd.x)
        tidey= float(tide_bnd.y)
        if tidex < 0:
            tidex = 360+tidex
    else:
        print('\n[Step %s][Set_Boundary_Conditions][Deepest model side =
downstream boundary, tidal conditions] .......\n'%(step))
        df_line['mean'] = pd.DataFrame(zonal_stats(vectors =
df_line['geometry'],raster = "%s%s_bathy_%s.tif"
%(folders[8],delta,xres),stats='mean',nodata = '-9999'))['mean']
        print('\n[Step %s][Set_Boundary_Conditions][Deepest model side =
downstream boundary, tidal conditions] .......\n'%(step))
        tide = df_line.loc[df_line['mean']==min(df_line['mean'])]
        df_line['boundary'].loc[tide] = 'BoundaryType2'
        #deepest =
int(df_line['id'][df_line['mean']==min(df_line['mean'])])
```

```
#########################################################################
#####

#########################################################################
#####
        oceans = gpd.read_file("%s%s_fulloceans_%s.shp"
%(folders[7],delta,xres))
        oceans = oceans.to_crs(df_line.crs)
        oceanboundary = gpd.overlay(df_line,oceans,how='intersection')
        oceanboundary['mean'] = pd.DataFrame(zonal_stats(vectors =
oceanboundary['geometry'],raster = "%s/%s_%s_%s.tif" %(folders[4] +
elev_name,delta,elev_name,xres),stats='mean',nodata = '-9999'))['mean']
        tideboundary =
oceanboundary.loc[oceanboundary['mean']==min(oceanboundary['mean'])]
        tideboundary = tideboundary['geometry'].centroid
        tideboundary = tideboundary.to_crs("EPSG:4326")
        tidex= float(tideboundary.x)
        tidey= float(tideboundary.y)
        if tidex < 0:
            tidex = 360+tidex
    ## Get Tide Data
    ## Run pyTMD to get global tidal predictions
    ## Set downstream boundary conditions

#########################################################################
#####

#########################################################################
#####

#########################################################################
#####
    print('\n[Step %s][Set_Boundary_Conditions][Set model run period]
.......\n'%(step))
    startdate = '20100101'
    enddate = '20211001'
    print('##################### Default simulation start and end are %s
- %s' %(startdate, enddate))
    print('\n[Step %s][Set_Boundary_Conditions][Get water stage time
series from TPXO Global Tide Model] .......\n'%(step))

    try:
        boundary_data = np.genfromtxt("%s/%s_tides_lat%s_lon%s_%s.csv"
%(folders[5],delta,np.round(tidey,2),np.round(tidex,2),startdate),delimit
er=',')
    except:
        boundary_data =
np.column_stack((maketides(float(tidey),float(tidex),str(startdate),str(e
nddate),10)))# value every 10 minutes
        np.savetxt("%s/%s_tides_lat%s_lon%s_%s.csv"
%(folders[5],delta,np.round(tidey,2),np.round(tidex,2),startdate),boundar
y_data,delimiter=',')
```

```
    print('##################### Using tide data from file:
%s_tides_lat%s_lon%s_%s.csv'
%(delta,np.round(tidey,2),np.round(tidex,2),startdate))

    print('\n[Step %s][Set_Boundary_Conditions] Finished
.......\n'%(step))

    return df_line, upstreamX,upstreamY,tidex,tidey



def get_inlet(delta,folders,res,parameters,inletx,inlety,elev_name):
    step = '8b'

print('\n\n\n#################################################################
################################')
    print('##############################[Step %s][Get Inlet
Location]##############################'%(step))

print('#################################################################
############################\n')

#########################################################################
##
    ###################### Import Config Parameters
##########################

#########################################################################
##
    xres,yres = res,res
    EPSG = parameters['EPSG'][0]
# Coordinate System must be UTM

    extentpoly = gpd.read_file("%s%s_modeldomain.shp"
%(folders[7],delta))
    ulx,lry,lrx,uly = extentpoly.total_bounds   # Coordinates converted
to UTM coordinate system
    extentpoly_line = extentpoly.geometry.boundary
    print('\n[Step %s][Get Inlet Location][Find River Boundary]
.......\n'%(step))


    x,y = extentpoly_line.geometry[0].coords.xy
    xy = pd.DataFrame(list(zip(x,y)), columns=['LON', 'LAT'])

    geoms = []
    ids = []
    for xy1 in range(len(xy)-1):
        x1 = xy.iloc[xy1]['LON']
        y1 = xy.iloc[xy1]['LAT']
        x2 = xy.iloc[xy1+1]['LON']
        y2 = xy.iloc[xy1+1]['LAT']
        geoms.append(LineString([(x1,y1),(x2,y2)]))
        ids.append(xy1)
```

```
    sides = {'id':ids,'geometry':geoms}
    df_line = gpd.GeoDataFrame(sides,columns=['id','geometry'])
    df_line['geometry'] = df_line.geometry.buffer(500)
    df_line.crs = extentpoly.crs
    df_line['index'] = df_line.index
    if (inlety == -9999) & (inletx == -9999) :

        print('\n[Step %s][Get Inlet Location][Largest river channel
along model edge = upstream boundary, river discharge conditions]
.......\n'%(step))
        rivers = gpd.read_file("%s%s_rivers_%s.shp"
%(folders[7],delta,xres))
        rivers = gpd.overlay(rivers,extentpoly,how='intersection')

        intersections = gpd.overlay(df_line,rivers,how='intersection')
        intersections['mean'] = pd.DataFrame(zonal_stats(vectors =
intersections['geometry'],raster = "%s/%s.tif"
%(folders[4],elev_name),stats='mean',nodata = '-9999'))['mean']
        intersections['areadepth'] = intersections.area *
intersections['mean']
        intersections.to_file('test2.shp')
        try:
            #inlet =
intersections.loc[intersections['mean']==min(intersections['mean'])].iloc
[0]
            #inlet =
intersections.loc[intersections.area==max(intersections.area)]
            inlet = intersections.loc[intersections['areadepth'] ==
np.nanmin(intersections['areadepth'])]
        except:
            print('no intersections between river and model boundary')
            inletx = 0
            inlety = 0
        else:
            inletx= int(inlet['geometry'].centroid.x)
            inlety= int(inlet['geometry'].centroid.y)
            print('#################### Discharge boundary conditions
set at %s,%s' %(inletx,inlety))

    #boundaries = ['BoundaryType1'] * (len(df_line)-1) # Boundary types:
Bi tides, Bi2 transmissive, but no stage set

    print('\n[Step %s][Get Inlet Location] Finished .......\n'%(step))

    return df_line, round(inletx,0),round(inlety,0)


def
get_tidal_boundary(delta,folders,res,parameters,tide_bnd,tidey,tidex,elev
_name):
    step = '8A'

print('\n\n\n##################################################################
###################################')
```

```
    print('##############################[Step %s][Get Tidal
Boundary]##############################'%(step))

print('#########################################################################
############################\n')

#########################################################################
##
    ###################### Import Config Parameters
#########################

#########################################################################
##
    xres,yres = res,res
    EPSG = parameters['EPSG'][0]
# Coordinate System must be UTM

    extentpoly = gpd.read_file("%s%s_modeldomain.shp"
%(folders[7],delta))
    ulx,lry,lrx,uly = extentpoly.total_bounds   # Coordinates converted
to UTM coordinate system
    extentpoly_line = extentpoly.geometry.boundary

    print('\n[Step %s][Get Tidal Boundary][Find Tidal Boundary]
.......\n'%(step))
    x,y = extentpoly_line.geometry[0].coords.xy
    xy = pd.DataFrame(list(zip(x,y)), columns=['LON', 'LAT'])

    geoms = []
    ids = []
    for xy1 in range(len(xy)-1):
        x1 = xy.iloc[xy1]['LON']
        y1 = xy.iloc[xy1]['LAT']
        x2 = xy.iloc[xy1+1]['LON']
        y2 = xy.iloc[xy1+1]['LAT']
        geoms.append(LineString([(x1,y1),(x2,y2)]))
        ids.append(xy1)
    sides = {'id':ids,'geometry':geoms}
    df_line = gpd.GeoDataFrame(sides,columns=['id','geometry'])
    df_line['geometry'] = df_line.geometry.buffer(500)
    df_line.crs = extentpoly.crs
    df_line['index'] = df_line.index

    df_line['boundary'] = ['NonTidal'] * (len(df_line))
    df_line.to_file('test.shp')
    if len(tide_bnd) > 0:
        df_line_tide = gpd.sjoin(df_line,tide_bnd, how='inner',
op='within')
        df_line_tide['id_left'].values
        df_line['boundary'][df_line_tide['id_left'].values] = 'Tide'
    else:
        df_line['mean'] = pd.DataFrame(zonal_stats(vectors =
df_line['geometry'],raster = "%s/%s.tif"
%(folders[4],elev_name),stats='mean',nodata = '-9999'))['mean']
```

```
        print('\n[Step %s][Set_Boundary_Conditions][Deepest model side =
downstream boundary, tidal conditions] .......\n'%(step))
        # tide = df_line.loc[df_line['mean']==min(df_line['mean'])]
        df_line['boundary'][df_line['mean']==np.nanmin(df_line['mean'])]
= 'Tide'
        #deepest =
int(df_line['id'][df_line['mean']==min(df_line['mean'])])
    if (tidex == -9999) & (tidey == -9999) :
        tide_lines = df_line[df_line['boundary']=='Tide'].geometry
        #tide_lines = tide_lines.to_crs('EPSG:4326')

        tidex = tide_lines[tide_lines.length ==
np.nanmax(tide_lines.length)].centroid.iloc[0].coords.xy[0][0]
        tidey = tide_lines[tide_lines.length ==
np.nanmax(tide_lines.length)].centroid.iloc[0].coords.xy[1][0]
    print('\n[Step %s][Get Tidal Boundary] Finished .......\n'%(step))

    return df_line,round(tidex,2),round(tidey,2)


def get_tide_data_pytmd(delta,path,tidex,tidey):
    step = '9'
    if tidex < 0:
        tidex = 360+tidex
    ## Get Tide Data
    ## Run pyTMD to get global tidal predictions
    ## Set downstream boundary conditions

##########################################################################
#####

##########################################################################
#####

##########################################################################
#####
    print('\n[Step %s][Set_Boundary_Conditions][Set model run period]
.......\n'%(step))
    startdate = '20100101'
    enddate = '20211001'
    print('##################### Default simulation start and end are %s
- %s' %(startdate, enddate))
    print('\n[Step %s][Set_Boundary_Conditions][Get water stage time
series from TPXO Global Tide Model] .......\n'%(step))

    try:
        tide_data = np.genfromtxt("%s/%s_tides_lat%s_lon%s_%s.csv"
%(path,delta,np.round(tidey,2),np.round(tidex,2),startdate),delimiter=','
)
    except:
        tide_data =
np.column_stack((maketides(float(tidey),float(tidex),str(startdate),str(e
nddate),10)))# value every 10 minutes
```

```
        np.savetxt("%s/%s_tides_lat%s_lon%s_%s.csv"
%(path,delta,np.round(tidey,2),np.round(tidex,2),startdate),tide_data,del
imiter=',')
    print('#################### Using tide data from file:
%s_tides_lat%s_lon%s_%s.csv'
%(delta,np.round(tidey,2),np.round(tidex,2),startdate))

    print('\n[Step %s][Set_Boundary_Conditions] Finished
.......\n'%(step))

    return tide_data
```