# Código de sensibilidad de remoción de capas

```
'''
# -*- coding: utf-8 -*-
#Autores
Fidel Serrano, Víctor Hernández

Afiliación
###############
Área de Planeación Colaborativa, Laboratorio Nacional de Ciencias de la
Sostenibilidad, Instituto de Ecología, Universidad Nacional Autónoma de México

Reconocimiento
#################
Este trabajo fue realizado con financiamiento del Programa de Apoyo a Proyectos de
Investigación e Innovación Tecnológica (PAPIIT IV100118)
del proyecto Análisis integrado de sistemas socio-ambientales acoplados: desarrollo
de capacidades para la evaluación de la vulnerabilidad costera.

'''

import copy
import pprint
import string
import qgis
import qgis.core
from osgeo import gdal
import gdal_calc
import os
import processing as pr
def raster_min_max(path_raster):
    '''
    Esta funcion regresa los valores maximos y minimos de una capa raster

    :param path_raster: ruta de la capa raster
    :type path_raster: str
    '''
    rlayer = QgsRasterLayer(path_raster,"raster")
    extent = rlayer.extent()
    provider = rlayer.dataProvider()
    stats = provider.bandStatistics(1,
                                    QgsRasterBandStats.All,
                                    extent,
                                    0)

    v_min = stats.minimumValue
```

```
    v_max = stats.maximumValue
    return v_min, v_max

def get_region(path_layer):
    '''
    Esta función regresa en forma de cadena de texto
    las coordenadas de la extensión de una capa raster

    param path_layer: ruta de la capa raster
    type path_layer: str
    '''

    layer = QgsRasterLayer(path_layer,"")
    ext = layer.extent()
    xmin = ext.xMinimum()
    xmax = ext.xMaximum()
    ymin = ext.yMinimum()
    ymax = ext.yMaximum()

    region = "%f,%f,%f,%f" % (xmin, xmax, ymin, ymax)
    return region

def nombre_capa(path_capa):
    '''
    Esta función regresa el nombre de una capa sin extensión

    :param path_capa: ruta de la capa
    :type path_capa: str


    '''
    nombre_capa=(path_capa.split("/")[-1:])[0].split(".")[0]
    return nombre_capa

def pesos_factores_vulnerabilidad(dicc):
    lista = []
    for k,v in dicc.items():
        lista.append(dicc[k]['w'])
    return lista

def ecuacion_vulnerabilidad(n):
    '''
    Esta función expresa la ecuación para el cálculo de la vulnerabilidad

    .. math::
        vulnerabilidad = \exp^{( 1 - sus)^{(1 + ca)}}


        | exp = Exposición
        | sus = Susceptibilidad
```

```
        | ca = Capacidad adaptativa

    :returns: str ecuacion
    '''
    if n==1:
        ecuacion = 'pow(A,(1-B))'
    if n==2:
        ecuacion = 'pow(pow(A,(1-B)),(1+C))'
    return ecuacion
def media_raster(path_raster):
    '''
    Esta función regresa el promedio de todos los pixeles válidos
    de un archivo raster

    :param path_raster: Ruta del archivo raster
    :type path_raster: String
    '''
    rlayer = qgis.core.QgsRasterLayer(path_raster,"raster")
    extent = rlayer.extent()
    provider = rlayer.dataProvider()
    stats = provider.bandStatistics(1,
                                    qgis.core.QgsRasterBandStats.All,
                                    extent,
                                    0)
    promedio = stats.mean
    return round(promedio,3)
def lista_criterios(dicc):
    '''
    Esta función regresa una lista de los criterios de un diccionario

    :param dicc: Diccionario que contiene nombres, rutas y pesos para el
    análisis de vulnerabilidad / sensibilidad
    :type dicc: diccionario python
    '''
    criterios = []
    for k1,v1 in dicc.items():
        for k2,v2, in v1['criterios'].items():
            elementos = list(v2.keys())
            if 'ruta' in elementos:
                criterios.append(k2)
            if 'criterios' in elementos:
                for k3,v3, in v2['criterios'].items():
                    elementos = list(v3.keys())
                    if 'ruta' in elementos:
                        criterios.append(k3)
                    if 'criterios' in elementos:
                        for k4,v4, in v3['criterios'].items():
                            elementos = list(v4.keys())
                            if 'ruta' in elementos:
                                criterios.append(k4)
```

```
                            if 'criterios' in elementos:
                                for k5,v5, in v4['criterios'].items():
                                    elementos = list(v5.keys())
                                    if 'ruta' in elementos:
                                        criterios.append(k5)

    return criterios
def crea_capa(ecuacion,rasters_input,salida):

    '''
    Esta función crea una capa mediante la calculadora raster
    de GDAL, esta función esta limitada hasta 14 variables en la ecuación.

    :param ecuacion: ecuación expresada en formato gdal,\
                    es este caso es la salida de la funcion *ecuacion_clp*
    :type ecuacion: String
    :param rasters_input: lista de los paths de los archivos rasters, salida de la
función *separa_ruta_pesos*
    :type rasters_input: lista
    :param salida: ruta con extensión tiff de la salida
    :type salida: String
    '''
    path_A=''
    path_B=''
    path_C=''
    path_D=''
    path_E=''
    path_F=''
    path_G=''
    path_H=''
    path_I=''
    path_J=''
    path_K=''
    path_L=''
    path_M=''
    path_N=''


    total_raster = len(rasters_input)

    for a,b in zip(range(total_raster), rasters_input):
        if a == 0:
            path_A=b
        elif a == 1:
            path_B=b
        elif a == 2:
            path_C=b
        elif a == 3:
            path_D=b
        elif a == 4:
```

```
            path_E=b
        elif a == 5:
            path_F=b
        elif a == 6:
            path_G=b
        elif a == 7:
            path_H=b
        elif a == 8:
            path_I=b
        elif a == 9:
            path_J=b
        elif a == 10:
            path_K=b
        elif a == 11:
            path_L=b
        elif a == 12:
            path_M=b
        elif a == 13:
            path_N=b




    if total_raster == 1:
        gdal_calc.Calc(calc=ecuacion,
                           A=path_A,
                           outfile=salida,
                           NoDataValue=-9999.0,
                           quiet=True)

    if total_raster == 2:
        gdal_calc.Calc(calc=ecuacion,
                       A=path_A,
                       B=path_B,
                       outfile=salida,
                       NoDataValue=-9999.0,
                       quiet=True)

    if total_raster == 3:
            gdal_calc.Calc(calc=ecuacion,
                           A=path_A,
                           B=path_B,
                           C=path_C,
                           outfile=salida,
                           NoDataValue=-9999.0,
                           quiet=True)

    if total_raster == 4:
        gdal_calc.Calc(calc=ecuacion,
                       A=path_A,
```

```
                        B=path_B,
                        C=path_C,
                        D=path_D,
                        outfile=salida,
                        NoDataValue=-9999.0,
                        quiet=True)

    if total_raster == 5:
            gdal_calc.Calc(calc=ecuacion,
                            A=path_A,
                            B=path_B,
                            C=path_C,
                            D=path_D,
                            E=path_E,
                            outfile=salida,
                            NoDataValue=-9999.0,
                            quiet=True)

    if total_raster == 6:
        gdal_calc.Calc(calc=ecuacion,
                        A=path_A,
                        B=path_B,
                        C=path_C,
                        D=path_D,
                        E=path_E,
                        F=path_F,
                        outfile=salida,
                        NoDataValue=-9999.0,
                        quiet=True)

    if total_raster == 7:
            gdal_calc.Calc(calc=ecuacion,
                            A=path_A,
                            B=path_B,
                            C=path_C,
                            D=path_D,
                            E=path_E,
                            F=path_F,
                            G=path_G,
                            outfile=salida,
                            NoDataValue=-9999.0,
                            quiet=True)

    if total_raster == 8:
        gdal_calc.Calc(calc=ecuacion,
                        A=path_A,
                        B=path_B,
                        C=path_C,
                        D=path_D,
                        E=path_E,
```

```
                        F=path_F,
                        G=path_G,
                        H=path_H,
                        outfile=salida,
                        NoDataValue=-9999.0,
                        quiet=True)

    if total_raster == 9:
        gdal_calc.Calc(calc=ecuacion,
                        A=path_A,
                        B=path_B,
                        C=path_C,
                        D=path_D,
                        E=path_E,
                        F=path_F,
                        G=path_G,
                        H=path_H,
                        I=path_I,
                        outfile=salida,
                        NoDataValue=-9999.0,
                        quiet=True)

    if total_raster == 10:
        gdal_calc.Calc(calc=ecuacion,
                        A=path_A,
                        B=path_B,
                        C=path_C,
                        D=path_D,
                        E=path_E,
                        F=path_F,
                        G=path_G,
                        H=path_H,
                        I=path_I,
                        J=path_J,
                        outfile=salida,
                        NoDataValue=-9999.0,
                        quiet=True)

    if total_raster == 11:
        gdal_calc.Calc(calc=ecuacion,
                        A=path_A,
                        B=path_B,
                        C=path_C,
                        D=path_D,
                        E=path_E,
                        F=path_F,
                        G=path_G,
                        H=path_H,
                        I=path_I,
                        J=path_J,
```

```
                        K=path_K,
                        outfile=salida,
                        NoDataValue=-9999.0,
                        quiet=True)

    if total_raster == 12:
        gdal_calc.Calc(calc=ecuacion,
                        A=path_A,
                        B=path_B,
                        C=path_C,
                        D=path_D,
                        E=path_E,
                        F=path_F,
                        G=path_G,
                        H=path_H,
                        I=path_I,
                        J=path_J,
                        K=path_K,
                        L=path_L,
                        outfile=salida,
                        NoDataValue=-9999.0,
                        quiet=True)

    if total_raster == 13:
        gdal_calc.Calc(calc=ecuacion,
                        A=path_A,
                        B=path_B,
                        C=path_C,
                        D=path_D,
                        E=path_E,
                        F=path_F,
                        G=path_G,
                        H=path_H,
                        I=path_I,
                        J=path_J,
                        K=path_K,
                        L=path_L,
                        M=path_M,
                        outfile=salida,
                        NoDataValue=-9999.0,
                        quiet=True)

    if total_raster == 14:
        gdal_calc.Calc(calc=ecuacion,
                        A=path_A,
                        B=path_B,
                        C=path_C,
                        D=path_D,
                        E=path_E,
                        F=path_F,
```

```
                        G=path_G,
                        H=path_H,
                        I=path_I,
                        J=path_J,
                        K=path_K,
                        L=path_L,
                        M=path_M,
                        N=path_N,
                        outfile=salida,
                        NoDataValue=-9999.0,
                        quiet=True)


def ecuacion_clp(pesos):

    '''
    Esta función recibe una lista de pesos para regresar la ecuación
    en la estructura requerida por gdal para la combinación lineal ponderada.

    :param pesos: lista de los pesos de las capas, salida de la función
*separa_ruta_pesos*
    :type pesos: lista
    '''
    n_variables=len(pesos)
    abc = list(string.ascii_uppercase)
    ecuacion = ''
    for a,b in zip(range(n_variables),pesos):

        if a < n_variables-1:
            ecuacion+= (str(b)+str(' * ')+str(abc[a])+' + ' )
        else:
            ecuacion+= (str(b)+str(' * ')+str(abc[a]))
    return ecuacion
def lista_pesos_ruta(dicc):
    '''
    Funcion para sacar listas por subcriterio
    '''
    exp_fis=[]
    exp_bio=[]
    sus_fis=[]
    sus_bio=[]
    pesos_1n = []
    for k1, v1 in dicc.items():

        for k2, v2 in v1['criterios'].items():
            pesos_1n.append(k1+"|"+str(v1['w'])+"|"+k2+"|"+str(v2['w']))
            for k3, v3 in v2['criterios'].items():
                if k1=='exposicion' and k2=='fisico':
                    exp_fis.append(v3['ruta'] + "|" + str(v3['w']))
                    #print (k1,v1['w'],k2,v2['w'],k3,v3['w'],v3['ruta'])
```

```
                elif k1=='exposicion' and k2=='biologico':
                    exp_bio.append(v3['ruta'] + "|" + str(v3['w']))
                    #print (k1,v1['w'],k2,v2['w'],k3,v3['w'],v3['ruta'])
                elif k1=='susceptibilidad' and k2=='fisico':
                    sus_fis.append(v3['ruta'] + "|" + str(v3['w']))
                    #print (k1,v1['w'],k2,v2['w'],k3,v3['w'],v3['ruta']
                elif k1=='susceptibilidad' and k2=='biologico':
                    sus_bio.append(v3['ruta'] + "|" + str(v3['w']))
                    #print (k1,v1['w'],k2,v2['w'],k3,v3['w'],v3['ruta']
    return exp_fis, exp_bio, sus_fis, sus_bio,pesos_1n

def separa_ruta_pesos(lista):
    rutas =[]
    pesos= []
    for capa in lista:
        rutas.append(capa.split("|")[0])
        pesos.append(capa.split("|")[1])
    return rutas,pesos


def rutas_pesos_globales(factor,dicc):
    lista = []   # se tiene que pasar a nivel recursiva
    for k1, v1 in dicc[factor].items():
        if type(v1) is dict:
            for k2, v2 in v1.items(): # nivel de biologicos o fisico
                if type(v2) is dict:
                    for k3,v3 in v2.items():
                        if type(v3) is dict:
                             for k4,v4 in v3.items(): #k4 e sel nombre de los
criterios  como v_acuatica_exp
                                elementos = list(v4.keys())
                                if 'ruta' in elementos:

lista.append(v4['ruta']+"|"+str(round(v2['w']*v4['w'],3)))
                                elif 'criterios' in elementos and  type(v4) is dict:
                                    for k5,v5, in v4.items():
                                        if type(v5) is dict:
                                            for k6,v6 in v5.items():
                                                elementos = list(v6.keys())
                                                if 'ruta' in elementos:

lista.append(v6['ruta']+"|"+str(round(v2['w']*v4['w']*v6['w'],3)))
                                                elif 'criterios' in elementos and
type(v6) is dict:
                                                    for k7,v7, in v6.items():
                                                        if type(v7) is dict:
                                                            for k8,v8 in v7.items():
                                                                elementos =
list(v8.keys())
                                                                if 'ruta' in
```

```
elementos:

lista.append(v8['ruta']+"|"+str(round(v2['w']*v4['w']*v6['w']*v8['w'],3)))

                                                                    elif 'criterios' in
elementos and  type(v8) is dict:
                                                                            for k9,v9,
in v8.items():
                                                                                if
type(v9) is dict:
                                                                                    for
k10,v10 in v9.items():

elementos = list(v10.keys())

if 'ruta' in elementos:

lista.append(v10['ruta']+"|"+str(round(v2['w']*v4['w']*v6['w']*v8['w']*v10['w'],3)))
    return lista


def find_key(d, key):

    for k,v in d.items():

        if isinstance(v, dict):
            p = find_key(v, key)
            if p:
                return [k] + p
        if k == key:
            return [k]
def quita(dicc,key):
    '''
    Esta función retira un elemento del diccionario y regresa un nuevo diccionario
    sin dicho elemento <<dicc_q>>.

    :param dicc: Diccionario con la estructura requerida
    :type dicc: diccionario
    :param key: nombre de la variable a quitar
    :type key: String

    '''
    dicc_q = copy.deepcopy(dicc)

    niveles = find_key(dicc_q,key)

    if len(niveles)==1:
        dicc_q.pop(niveles[0])
    if len(niveles)==3:
        dicc_q[niveles[0]]['criterios'].pop(niveles[2])
```

```
        if len(dicc_q[niveles[0]]['criterios'])==0:
            dicc_q.pop(niveles[0])
    if len(niveles)==5:
        dicc_q[niveles[0]]['criterios'][niveles[2]]['criterios'].pop(niveles[4])
        if len(dicc_q[niveles[0]]['criterios'][niveles[2]]['criterios'])==0:
            dicc_q[niveles[0]]['criterios'].pop(niveles[2])
        if len(dicc_q[niveles[0]]['criterios'])==0:
            dicc_q.pop(niveles[0])
    if len(niveles)==7:
        dicc_q[niveles[0]]['criterios'][niveles[2]]['criterios'][niveles[4]]
['criterios'].pop(niveles[6])
        if len(dicc_q[niveles[0]]['criterios'][niveles[2]]['criterios'][niveles[4]]
['criterios'])==0:
            dicc_q[niveles[0]]['criterios'][niveles[2]]['criterios'].pop(niveles[4])
        if len(dicc_q[niveles[0]]['criterios'][niveles[2]]['criterios'])==0:
            dicc_q[niveles[0]]['criterios'].pop(niveles[2])
        if len(dicc_q[niveles[0]]['criterios'])==0:
            dicc_q.pop(niveles[0])
    if len(niveles)==9:
        dicc_q[niveles[0]]['criterios'][niveles[2]]['criterios'][niveles[4]]
['criterios'][niveles[6]]['criterios'].pop(niveles[8])
        if len(dicc_q[niveles[0]]['criterios'][niveles[2]]['criterios'][niveles[4]]
['criterios'][niveles[6]]['criterios'])==0:
            dicc_q[niveles[0]]['criterios'][niveles[2]]['criterios'][niveles[4]]
['criterios'].pop(niveles[6])
        if len(dicc_q[niveles[0]]['criterios'][niveles[2]]['criterios'][niveles[4]]
['criterios'])==0:
            dicc_q[niveles[0]]['criterios'][niveles[2]]['criterios'].pop(niveles[4])
        if len(dicc_q[niveles[0]]['criterios'][niveles[2]]['criterios'])==0:
            dicc_q[niveles[0]]['criterios'].pop(niveles[2])
        if len(dicc_q[niveles[0]]['criterios'])==0:
            dicc_q.pop(niveles[0])


    return dicc_q


def reescala(dicc_q):
    '''
    Esta función rescala un diccionario que se le a quitado un criterio
    y regresa el diccionario con los pesos rescalados

    :param dicc_q: salida de la función *quita*

    '''

    dicc_r = copy.deepcopy(dicc_q)
    suma = 0
    for k1, v1 in dicc_r.items():
        suma += v1['w']
```

```
for k1, v1 in dicc_r.items():
    v1['w'] = v1['w']/suma

for k1, v1 in dicc_r.items():
    suma = 0
    elementos = list(v1.keys())
    if 'criterios' in elementos:
        for k2, v2 in v1['criterios'].items():
            suma += v2['w']
        for k2, v2 in v1['criterios'].items():
            v2['w'] = v2['w'] / suma


for k1, v1 in dicc_r.items():
    elementos = list(v1.keys())
    if 'criterios' in elementos:
        for k2, v2 in v1['criterios'].items():
            suma = 0
            elementos = list(v2.keys())
            if 'criterios' in elementos:
                for k3, v3 in v2['criterios'].items():
                    suma += v3['w']
                for k3, v3 in v2['criterios'].items():
                    v3['w'] = v3['w'] / suma


for k1, v1 in dicc_r.items():
    elementos = list(v1.keys())
    if 'criterios' in elementos:
        for k2, v2 in v1['criterios'].items():
            elementos = list(v2.keys())
            if 'criterios' in elementos:
                for k3, v3 in v2['criterios'].items():
                    suma = 0
                    elementos = list(v3.keys())
                    if 'criterios' in elementos:
                        for k4, v4 in v3['criterios'].items():
                            suma += v4['w']
                        for k4, v4 in v3['criterios'].items():
                            v4['w'] = v4['w'] / suma

for k1, v1 in dicc_r.items():
    elementos = list(v1.keys())
    if 'criterios' in elementos:
        for k2, v2 in v1['criterios'].items():
            elementos = list(v2.keys())
            if 'criterios' in elementos:
                for k3, v3 in v2['criterios'].items():
                    elementos = list(v3.keys())
                    if 'criterios' in elementos:
```

```
                            for k4, v4 in v3['criterios'].items():
                                suma = 0
                                elementos = list(v4.keys())
                                if 'criterios' in elementos:
                                    for k5, v5 in v4['criterios'].items():
                                        suma += v5['w']
                                    for k5, v5 in v4['criterios'].items():
                                        v5['w'] = v5['w'] / suma




    return dicc_r

def quita_reescala(dicc,key):
    '''
    Función que integra las funciones quita y reescala y regresa
    un diccionario sin la variable y con los pesos reescalados.

    :param dicc: Diccionario con la estructura requerida
    :type dicc: diccionario
    :param key: nombre de la variable a quitar
    :type key: String
    '''
    dicc_q = quita(dicc,key)
    dicc_r = reescala(dicc_q)
    return dicc_r
def pesos_superiores(dicc):
    pesos=[]
    for k1,v1 in dicc.items():
        #print (k1,v1['w'])
        pesos.append([k1,str(v1['w'])])
    return pesos


def ideales(path_raster, path_raster_n):
    min,max = raster_min_max(path_raster)
    no_data =raster_nodata(path_raster)
    ec_norm ='(A' + ') / (' + str(max) +')'  # llevar a ideal
    #ec_norm ='(A - '+str(min) + ') / (' + str(max)+'-'+str(min) +')'  # normalizar
    dicc ={
        'INPUT_A':path_raster,
        'BAND_A':1,
        'FORMULA':ec_norm,
        'NO_DATA': no_data,
        'RTYPE':5,
        'OUTPUT':path_raster_n}
    pr.run("gdal:rastercalculator",dicc)
```

```
def raster_nodata(path_raster):

    rlayer = QgsRasterLayer(path_raster,"raster")
    extent = rlayer.extent()


    provider = rlayer.dataProvider()
    rows = rlayer.rasterUnitsPerPixelY()
    cols = rlayer.rasterUnitsPerPixelX()
    block = provider.block(1, extent,  rows, cols)

    no_data = block.noDataValue()


    return no_data

def lineal_decreciente(path_raster, path_raster_n):
    min,max = raster_min_max(path_raster)
    no_data =raster_nodata(path_raster)
    xmax_menos_xmin = max-min
    xmax_menos_xmin_xmax = xmax_menos_xmin / max
    xmax_mas_xmin = max + min
    xmax_mas_xmin_xmax = xmax_mas_xmin / max

    ec_norm ='((-A * '+str(xmax_menos_xmin_xmax)+') / '+str(xmax_menos_xmin)+') +
'+str(xmax_mas_xmin_xmax)  # llevar a ideal

    dicc ={
        'INPUT_A':path_raster,
        'BAND_A':1,
        'FORMULA':ec_norm,
        'NO_DATA': no_data,
        'RTYPE':5,
        'OUTPUT':path_raster_n}
    pr.run("gdal:rastercalculator",dicc)

''' # <- Quitar comillas para usar código
## rutas de entrada y salida

p_sig_exp= 'C:/Dropbox (LANCIS)/SIG/desarrollo/sig_papiit/entregables/\
exposicion/'
p_sig_sens= 'C:/Dropbox (LANCIS)/SIG/desarrollo/sig_papiit/entregables/\
susceptibilidad/'
p_sig_res= 'C:/Dropbox (LANCIS)/SIG/desarrollo/sig_papiit/entregables/\
resiliencia/'
p_procesamiento = 'C:/vulnerabilidad13/'
zonas_ambiente_construido  = 'C:/Dropbox
(LANCIS)/SIG/desarrollo/sig_papiit/entregables/mascaras/zonas_ambiente_construido.ti
f'
```

```
dicc = {
    'exposicion': {'w':0.333,
                        'criterios':{'biologico':{'w':0.50,
                                                            'criterios':
{'v_acuatica_exp':
{'w':0.16,'ruta':p_sig_exp+'biologica/v_acuatica_yuc/fv_v_acuatica_yuc_100m.tif'},

'v_costera_exp':{ 'w':0.84,

'criterios':{'distancia_manglar_exp':{'w':0.75,'ruta':p_sig_exp +
'biologica/v_costera_yuc/fv_distancia_manglar_yuc_100m.tif'},

'distancia_dunas_exp':{'w':0.25,'ruta':p_sig_exp +
'biologica/v_costera_yuc/fv_distancia_dunas_yuc_100m.tif'}}}}},
                                          'fisico':{'w':0.50,
                                                        'criterios':{'elevacion_exp':{
'w':0.87,'ruta':p_sig_exp+ 'fisica/elev_yuc/fv_elevacion_yuc_100m.tif'},

'ancho_playa_exp':{'w':0.13,'ruta':p_sig_exp+
'fisica/ancho_playa_yuc/fv_distancia_playa_yuc_100m.tif'}
                                                                        }
                                                    }
                                    }
                    },
    'susceptibilidad': {'w':0.333,
                            'criterios':{'biologico':{'w':0.50 ,
                                                            'criterios':
{'v_costera_sus':{ 'w':1.0,'ruta':p_sig_sens +
'biologica/v_costera_yuc/fv_v_costera_presencia_yuc_100m.tif'}}},
                                                'fisico':{'w':0.50,
                                                            'criterios':
{'elevacion_sus':{ 'w':0.26,'ruta':p_sig_sens +
'fisica/elev_yuc/fv_elevacion_yuc_100m.tif' },

'duna_costera_sus':{'w':0.10,'ruta':p_sig_sens +
'fisica/duna_yuc/fv_duna_yuc_100m.tif'},

'tipo_litoral_sus':{'w':0.64,'ruta':p_sig_sens +
'fisica/t_litoral_yuc/fv_tipo_litoral_yuc_100m_corregida.tif'}
                                                                        }
                                                    }
                                    }
                    },
    'resiliencia': {'w':0.333,
                            'criterios':{'biologico':{'w':0.50 ,
                                                            'criterios':
{'biodiversidad_res':{'w':0.50,'ruta':p_sig_res +
'biologica/biodiversidad/fv_biodiversidad_yuc_100m.tif'},
```

```
'servicios_ambientales_res':{'w':0.50,

'criterios':{'proteccion_costera_res':{'w':0.75,

'criterios':{'entidades_protectoras_res':{'w':0.84,'ruta':p_sig_res +
'biologica/serv_ambientales/prot_costera_yuc/fv_entidades_protectoras.tif'},

'v_acuatica_res':{'w':0.16,'ruta':p_sig_res +
'biologica/serv_ambientales/prot_costera_yuc/fv_v_acuatica_yuc_100m.tif'}

}

},

'provision_res':{'w':0.25,'ruta':p_sig_res +
'biologica/serv_ambientales/provision_yuc/fv_provision_yuc_100m.tif'}}

}
                                                                    }
                                                 },
                                        'fisico':{'w':0.50,
                                                        'criterios':
{'elevacion_res':{'w':0.60,'ruta':p_sig_res + 'fisica/elev_yuc/fv_elev_yuc.tif'},

'tipo_litoral_res':{'w':0.40,'ruta':p_sig_res +
'fisica/t_litoral_yuc/fv_tipo_litoral_yuc.tif'}
                                                                }
                                                    }
                                        }
                    }

    }
# ----- GENERACIÓN DE LAS CAPAS CONSIDERANDO TODOS LOS ELEMENTOS ----- #

exposicion_total,susceptibilidad_total,resiliencia_total =
rutas_pesos_globales('exposicion',dicc),rutas_pesos_globales('susceptibilidad',dicc)
,rutas_pesos_globales('resiliencia',dicc)
path_exp_t,w_exp_t = separa_ruta_pesos(exposicion_total)
path_sus_t,w_sus_t = separa_ruta_pesos(susceptibilidad_total)
path_res_t,w_res_t = separa_ruta_pesos(resiliencia_total)

ecuacion_exp_t = ecuacion_clp(w_exp_t)
ecuacion_sus_t = ecuacion_clp(w_sus_t)
ecuacion_res_t = ecuacion_clp(w_res_t)



dir_tmp = p_procesamiento+"tmp"
if "tmp" not in os.listdir(p_procesamiento):
```

```
    os.mkdir(dir_tmp)


exposicion = dir_tmp+"/tp_exposicion.tif"
susceptibilidad = dir_tmp+"/tp_susceptibilidad.tif"
tp_resiliencia = dir_tmp+"/tp_resiliencia.tif"


exposicion_ze = dir_tmp+"/tp_exposicion_ze.tif"
susceptibilidad_ze = dir_tmp+"/tp_susceptibilidad_ze.tif"
tp_resiliencia_ze = dir_tmp+"/tp_resiliencia_ze.tif"


crea_capa(ecuacion_exp_t,path_exp_t,exposicion)
crea_capa(ecuacion_sus_t,path_sus_t,susceptibilidad)
crea_capa(ecuacion_res_t,path_res_t,tp_resiliencia)

crea_capa('A*B',[exposicion,zonas_ambiente_construido],exposicion_ze)
crea_capa('A*B',[susceptibilidad, zonas_ambiente_construido],susceptibilidad_ze)
crea_capa('A*B',[tp_resiliencia, zonas_ambiente_construido],tp_resiliencia_ze)

resiliencia = dir_tmp+"/tp_resiliencia_dec.tif"




# --- Creación de la capa de vulnerabilidad ---##



vulnerabilidad= dir_tmp +"/tp_vulnerabilidad.tif"

lista_exp_sus_res = [exposicion,susceptibilidad,resiliencia]
#lista_exp_sus_res = [exposicion_ze,susceptibilidad_ze,resiliencia]  ## CAPAS CON
ZONAS DE EXCLUSION

w_vulnerabilidad = pesos_factores_vulnerabilidad(dicc)
ecuacion_vul_lineal = ecuacion_clp(w_vulnerabilidad)
crea_capa(ecuacion_vul_lineal,lista_exp_sus_res,vulnerabilidad)


### Datos para el analisis de sensibilidad ####

vulnerabilidad_total_media = media_raster(vulnerabilidad)

exp_media_total = media_raster(exposicion)
sus_media_total = media_raster(susceptibilidad)
res_media_total = media_raster(tp_resiliencia)

## PROMEDIOS DE CAPAS CON ZONAS DE EXCLUSION
```

```
# exp_media_total = media_raster(exposicion_ze)
# sus_media_total = media_raster(susceptibilidad_ze)
# res_media_total = media_raster(tp_resiliencia_ze)



#---- TERMINA ---#

#--- ANALISIS DE SENSIBILIDAD ----#


archivo = open(p_procesamiento+"sensibilidad_por_criterio_ze.csv","w")
archivo.write("criterio,exposicion,sensibilidad_exp,suceptibilidad,sensibilidad_sus,
resiliencia,sensibilidad_res,vulnerabilidad,sensibilidad_vul\n")
archivo.write("total,"+str(round(exp_media_total,3))+",,"+str(round(sus_media_total,
3))+",,"+str(round(res_media_total,3))+",,"+str(round(vulnerabilidad_total_media,3))
+","+"\n")

criterios = lista_criterios(dicc) #obtiene criterios del diccionario principal
cont=0
for criterio in criterios:
    cont +=1
    print ("procensado criterio: ",criterio,"  ",cont,"de",len(criterios))
    dicc2  = quita_reescala(dicc,criterio)


    exposicion,susceptibilidad,resiliencia =
rutas_pesos_globales('exposicion',dicc2),rutas_pesos_globales('susceptibilidad',dicc
2),rutas_pesos_globales('resiliencia',dicc2) #separa los subcriterios por criterios

    path_exp,w_exp = separa_ruta_pesos(exposicion)
    path_sus,w_sus = separa_ruta_pesos(susceptibilidad)
    path_res,w_res = separa_ruta_pesos(resiliencia)

    ecuacion_exp = ecuacion_clp(w_exp)
    ecuacion_sus = ecuacion_clp(w_sus)
    ecuacion_res = ecuacion_clp(w_res)


    salida_exposicion = dir_tmp+"/tp_exposicion_sin_"+criterio+".tif"
    salida_susceptibilidad = dir_tmp+"/tp_suscep_sin_"+criterio+".tif"
    salida_resiliencia = dir_tmp+"/tp_resilencia_sin_"+criterio+".tif"


    salida_exposicion_ze = dir_tmp+"/tp_exposicion_sin_"+criterio+"_ze.tif"
    salida_susceptibilidad_ze = dir_tmp+"/tp_suscep_sin_"+criterio+"_ze.tif"
    salida_resiliencia_ze = dir_tmp+"/tp_resilencia_sin_"+criterio+"_ze.tif"


    crea_capa(ecuacion_exp,path_exp,salida_exposicion)
```

```
    crea_capa(ecuacion_sus,path_sus,salida_susceptibilidad)
    crea_capa(ecuacion_res,path_res,salida_resiliencia)

    crea_capa('A*B',
[salida_exposicion,zonas_ambiente_construido],salida_exposicion_ze)
    crea_capa('A*B',[salida_susceptibilidad,
zonas_ambiente_construido],salida_susceptibilidad_ze)
    crea_capa('A*B',[salida_resiliencia,
zonas_ambiente_construido],salida_resiliencia_ze)


    salida_resiliencia_lineal_dec = dir_tmp+"/tp_resilencia_sin_"+criterio+"_ld.tif"
    #lineal_decreciente(salida_resiliencia_ze,salida_resiliencia_lineal_dec) #
LINEAL DECRECIENTE DE RESILIENCIA A ZONAS DE EXCLUSION

    lineal_decreciente(salida_resiliencia,salida_resiliencia_lineal_dec)
    salida_vulnerabilidad = dir_tmp+"/tp_vulnerabilidad_sin_"+criterio+".tif"



    # --- Creación de la capa de vulnerabilidad --- ##

    # lista_c =
[salida_exposicion_ze,salida_susceptibilidad_ze,salida_resiliencia_lineal_dec]
#VULNERABILIDAD CON ZONAS DE EXCLUSION
    lista_c =
[salida_exposicion,salida_susceptibilidad,salida_resiliencia_lineal_dec] #
VULNERABILIDAD SIN ZONAS DE EXCLUSION

    w_vulnerabilidad = pesos_factores_vulnerabilidad(dicc2)
    ecuacion_vul_lineal = ecuacion_clp(w_vulnerabilidad)
    crea_capa(ecuacion_vul_lineal,lista_c,salida_vulnerabilidad)
    print('pesos vulnerabilidad',w_vulnerabilidad)

    # --- Promedios de los criterios ---- ##
    vulnerabilidad_media = media_raster(salida_vulnerabilidad)

    exp_media = media_raster(salida_exposicion)
    sus_media = media_raster(salida_susceptibilidad)
    res_media = media_raster(salida_resiliencia)
    ### PROEMDIOS DE CRITERIOS CON ZONAS DE EXCLUSION ##
    # exp_media = media_raster(salida_exposicion_ze)
    # sus_media = media_raster(salida_susceptibilidad_ze)
    # res_media = media_raster(salida_resiliencia_ze)

    #---  Cálculo de la sensibilidad ---##
    sensibilidad_exp_calculada = round((abs((exp_media_total-
exp_media))/exp_media_total),3)
    sensibilidad_sus_calculada = round((abs((sus_media_total-
sus_media))/sus_media_total),3)
```

```
    sensibilidad_res_calculada = round((abs((res_media_total-
res_media))/res_media_total),3)
    sensibilidad_calculada = round((abs((vulnerabilidad_total_media-
vulnerabilidad_media))/vulnerabilidad_total_media),3)
    archivo.write(criterio+","+\
                    str(exp_media)+","+\
                    str(sensibilidad_exp_calculada)+","+\
                    str(round(sus_media,3))+","+\
                    str(sensibilidad_sus_calculada)+","+\
                    str(round(res_media,3))+","+\
                    str(sensibilidad_res_calculada)+","+\
                    str(round(vulnerabilidad_media,3))+","+\
                    str(sensibilidad_calculada)+"\n")



archivo.close()

''' # <- Quitar comillas para usar código
```