Anejos

Anejo 1: Script de modelo de flujo

a. IMPORTAR BIBLIOTECAS

```
import os
import pandas as pd
import flopy
import matplotlib
import numpy as np
import matplotlib.pyplot as plt
import flopy.utils.binaryfile as bf
import math
from flopy.utils.gridgen import Gridgen
import geopandas as gpd
import flopy.discretization as fgrid
from flopy.utils.gridintersect import GridIntersect
from shapely.geometry import MultiLineString
from scipy.interpolate import griddata
```

b. MODELO DE FLUJO

i. Asignación de nombre de modelo

```
model_ws='../../Outputs/Resultados_Base/'
modelname= 'RUCM_Base'
exe_name= '../../Solver/mf2005.exe'
mf= flopy.modflow.Modflow(modelname, exe_name=exe_name,
model_ws=model_ws)
```

ii. Shapefile de la cuenca

```
basinDf =
gpd.read_file("../../Inputs/Data_GIS/Shp/Cuenca/RUCM_Cuenca.shp")
basinGeom = basinDf.iloc[0].geometry
```

iii. Discretización temporal

```
import calendar
anio_inicial = 1980
anio_final = 2016
time = []
for anio in range(anio_inicial,anio_final+1): # intervalo de años
    for mes in range(1,13):
        x = calendar.monthrange(anio,mes)
        time.append(x[1]) # extrayendo solo los dias
time_days = time[9:-3:1] #inicia en Octubre y termina en Setiembre

nper = len(time_days) #número de stress period
perlen = time_days  #tiempo total de cada periodo
nstp = [1]+[1]*(nper-1) #time step
tsmult = [1]+[1]*(nper-1) #multiplicador
steady = [False]+[False]*(nper-1) # estado transitorio desde el 1°
stress period
```

iv. Discretización espacial

```
# Discretización espacial preliminar:
ztop = 600
zbot = [-66, -152, -360]
nlay = 3
delr = 500
delc = 500
nrow = 99
ncol = 127

sgr = fgrid.StructuredGrid(delc*np.ones(nrow), delr*np.ones(ncol),
                     xoff=626814.041187938, yoff=4349765.06031221)
ix = GridIntersect(sgr, method='vertex', rtree=True)
```

```
    basin_intersected = ix.intersect_polygon(basinGeom)
```

v. Paquete DIS

```
    dis=flopy.modflow.ModflowDis(mf, nlay=nlay, nrow=nrow, ncol=ncol,
    delr=delr, delc=delc, top=ztop, botm=zbot, nper=nper, nstp=nstp,
    perlen=perlen, tsmult=tsmult, steady=steady, tart_datetime='1/1/1980')
    mf.modelgrid.set_coord_info(xoff=626814.041187938,yoff=4349765.0603122
    1, angrot=0, epsg=25830)
```

vi. Niveles topográficos de las capas del modelo

```
    Data_mtop = pd.read_excel('../../Inputs/Datos_del_modelo.xlsx',
                                sheet_name='mtop')
    mtop = Data_mtop.to_numpy()
    mf.dis.top = mtop
    zbot = np.zeros((nlay, nrow, ncol))
    zbot[0,:,:] = mtop -66
    zbot[1,:,:] = mtop -152
    zbot[2,:,:] = mtop -360
    mf.dis.botm = zbot
```

vii. Alturas iniciales

```
    Data_AlturasIniciales =
    pd.read_excel('../../Inputs/Datos_del_modelo.xlsx',
                                sheet_name='Alturas_Iniciales')
    top_AI = Data_AlturasIniciales.to_numpy()
```

viii. Definición de celdas activas y constantes/ BAS

```
    # Variables del paquete BAS
    idomain = np.zeros((nlay, nrow, ncol))
    for i in basin_intersected.cellids:
        idomain[:,i[0],i[1]]=1
        idomain[:,76,54]=1
        idomain[:,83,65]=1
    mf.modelgrid.set_coord_info(xoff=626814.041187938,yoff=4349765.0603122
    1, angrot=0, epsg=25830)
    bas = flopy.modflow.ModflowBas(mf,ibound=idomain,strt=np.stack([top_AI
    for i in range(nlay)]),ichflg= 'CHTOCH')
```

ix. Conductividad Hidráulica

```
    #eyendo el archivo excel:

    Excel_Kx_Layer1 = pd.read_excel('../../Inputs/Datos_del_modelo.xlsx',
                                sheet_name='Parametros', usecols=[1])
    Excel_Kx_Layer2 = pd.read_excel('../../Inputs/Datos_del_modelo.xlsx',
                                sheet_name='Parametros', usecols=[2])
    Excel_Kx_Layer3 = pd.read_excel('../../Inputs/Datos_del_modelo.xlsx',
                                sheet_name='Parametros', usecols=[3])

    #Convirtiendo de lista a matriz
    Kx_Layer1 = Excel_Kx_Layer1.values.tolist()
    Kx_Layer2 = Excel_Kx_Layer2.values.tolist()
    Kx_Layer3 = Excel_Kx_Layer3.values.tolist()
    #Extrayendo los valores del matriz
    Kx_Layer1_valor=[y for x in Kx_Layer1 for y in x]
    Kx_Layer2_valor=[y for x in Kx_Layer2 for y in x]
    Kx_Layer3_valor=[y for x in Kx_Layer3 for y in x]
    #Leyendo los shapefiles
    Zonificacion_Parametros =
    gpd.read_file("../../Inputs/Data_GIS/Shp/Parametros/RUCM_Zonificacion_
    Parametros.shp")

    # Creando una función para la CH:
```

```
def Funcion_Conductividad_Hidraulica(i):
    Zonificacion_Geom    =    Zonificacion_Parametros.iloc[i].geometry
#Extrayendo la propiedad de geometria de los shapefiles por zonas
    Zona_intersected     =     ix.intersect_polygon(Zonificacion_Geom)
#Intersecando el shapefile y la grilla

    #Integrando las conductividades hidraulicas por zonas:
    Kx_Zona   =   np.zeros((nlay,   nrow,   ncol))  #Creando   matrices
nlay,nrow,ncol por zonas:
    for j in Zona_intersected.cellids:
        Kx_Zona[0,j[0],j[1]] = Kx_Layer1_valor[i]
        Kx_Zona[1,j[0],j[1]] = Kx_Layer2_valor[i]
        Kx_Zona[2,j[0],j[1]] = Kx_Layer3_valor[i]
    return Kx_Zona

# Conductividades hidraulicas por zona:
Kx_Aluvial_1 = Funcion_Conductividad_Hidraulica(0)
Kx_Aluvial_2 = Funcion_Conductividad_Hidraulica(1)
Kx_Aluvial_3 = Funcion_Conductividad_Hidraulica(2)
Kx_CarbonatadaAlta_1 = Funcion_Conductividad_Hidraulica(3)
Kx_CarbonatadaAlta_2 = Funcion_Conductividad_Hidraulica(4)
Kx_CarbonatadaAlta_3 = Funcion_Conductividad_Hidraulica(5)
Kx_CarbonatadaAlta_4 = Funcion_Conductividad_Hidraulica(6)
Kx_CarbonatadaMedia_1 = Funcion_Conductividad_Hidraulica(7)
Kx_CarbonatadaMedia_2 = Funcion_Conductividad_Hidraulica(8)
Kx_CarbonatadaMedia_3 = Funcion_Conductividad_Hidraulica(9)
Kx_CarbonatadaMedia_4 = Funcion_Conductividad_Hidraulica(10)
Kx_CarbonatadaMedia_5 = Funcion_Conductividad_Hidraulica(11)
Kx_DetreticaMedia_1 = Funcion_Conductividad_Hidraulica(12)
Kx_DetreticaMedia_2 = Funcion_Conductividad_Hidraulica(13)
Kx_DetreticaMedia_3 = Funcion_Conductividad_Hidraulica(14)
Kx_DetreticaMedia_4 = Funcion_Conductividad_Hidraulica(15)
Kx_DetreticaMedia_5 = Funcion_Conductividad_Hidraulica(16)
Kx_DetreticaMedia_6 = Funcion_Conductividad_Hidraulica(17)
Kx_DetreticaMuyBaja_1 = Funcion_Conductividad_Hidraulica(18)
Kx_DetreticaMuyBaja_2 = Funcion_Conductividad_Hidraulica(19)

#Conductividad hidráulica total:
Kx_Aluvial = Kx_Aluvial_1+Kx_Aluvial_2+Kx_Aluvial_3
Kx_CarbonatadaAlta=Kx_CarbonatadaAlta_1+Kx_CarbonatadaAlta_2+Kx_Carbon
atadaAlta_3+Kx_CarbonatadaAlta_4
Kx_CarbonatadaMedia=Kx_CarbonatadaMedia_1+Kx_CarbonatadaMedia_2+Kx_Car
bonatadaMedia_3+Kx_CarbonatadaMedia_4+Kx_CarbonatadaMedia_5
Kx_DetreticaMedia=Kx_DetreticaMedia_1+Kx_DetreticaMedia_2+Kx_Detretica
Media_3+Kx_DetreticaMedia_4+Kx_DetreticaMedia_5+Kx_DetreticaMedia_6
Kx_DetreticaMuyBaja = Kx_DetreticaMuyBaja_1+Kx_DetreticaMuyBaja_2
Kx=Kx_Aluvial+Kx_CarbonatadaAlta+Kx_CarbonatadaMedia+Kx_DetreticaMedia
+Kx_DetreticaMuyBaja
```

x. Almacenamiento Especifico

```
# leyendo el archivo excel:

Excel_Ss_Layer1 = pd.read_excel('../../Inputs/Datos_del_modelo.xlsx',
                                sheet_name='Parametros', usecols=[4])
Excel_Ss_Layer2 = pd.read_excel('../../Inputs/Datos_del_modelo.xlsx',
                                sheet_name='Parametros', usecols=[5])
Excel_Ss_Layer3 = pd.read_excel('../../Inputs/Datos_del_modelo.xlsx',
                                sheet_name='Parametros', usecols=[6])

#Convirtiendo de lista a matriz
Ss_Layer1 = Excel_Ss_Layer1.values.tolist()
Ss_Layer2 = Excel_Ss_Layer2.values.tolist()
Ss_Layer3 = Excel_Ss_Layer3.values.tolist()
#Extrayendo los valores del matriz
Ss_Layer1_valor=[y for x in Ss_Layer1 for y in x]
Ss_Layer2_valor=[y for x in Ss_Layer2 for y in x]
```

```
Ss_Layer3_valor=[y for x in Ss_Layer3 for y in x]

# Creando una funcion para el Almacenamiento Especifico :
def Funcion_Ss(i):
    Zonificacion_Geom    =    Zonificacion_Parametros.iloc[i].geometry
#Extrayendo la propiedad de geometria de los shapefiles por zonas
    Zona_intersected     =     ix.intersect_polygon(Zonificacion_Geom)
#Intersecando el shapefile y la grilla

    #Integrando las conductividades hidraulicas por zonas:
    Ss_Zona   =   np.zeros((nlay,   nrow,   ncol))  #Creando   matrices
nlay,nrow,ncol por zonas:
    for j in Zona_intersected.cellids:
        Ss_Zona[0,j[0],j[1]] = Ss_Layer1_valor[i]
        Ss_Zona[1,j[0],j[1]] = Ss_Layer2_valor[i]
        Ss_Zona[2,j[0],j[1]] = Ss_Layer3_valor[i]
    return Ss_Zona

# Almacenamiento Especifico por zonas:
Ss_Aluvial_1 = Funcion_Ss(0)
Ss_Aluvial_2 = Funcion_Ss(1)
Ss_Aluvial_3 = Funcion_Ss(2)
Ss_CarbonatadaAlta_1 = Funcion_Ss(3)
Ss_CarbonatadaAlta_2 = Funcion_Ss(4)
Ss_CarbonatadaAlta_3 = Funcion_Ss(5)
Ss_CarbonatadaAlta_4 = Funcion_Ss(6)
Ss_CarbonatadaMedia_1 = Funcion_Ss(7)
Ss_CarbonatadaMedia_2 = Funcion_Ss(8)
Ss_CarbonatadaMedia_3 = Funcion_Ss(9)
Ss_CarbonatadaMedia_4 = Funcion_Ss(10)
Ss_CarbonatadaMedia_5 = Funcion_Ss(11)
Ss_DetreticaMedia_1 = Funcion_Ss(12)
Ss_DetreticaMedia_2 = Funcion_Ss(13)
Ss_DetreticaMedia_3 = Funcion_Ss(14)
Ss_DetreticaMedia_4 = Funcion_Ss(15)
Ss_DetreticaMedia_5 = Funcion_Ss(16)
Ss_DetreticaMedia_6 = Funcion_Ss(17)
Ss_DetreticaMuyBaja_1 = Funcion_Ss(18)
Ss_DetreticaMuyBaja_2 = Funcion_Ss(19)

# Almacenamiento Especifico total:
Ss_Aluvial = Ss_Aluvial_1+Ss_Aluvial_2+Ss_Aluvial_3
Ss_CarbonatadaAlta=Ss_CarbonatadaAlta_1+Ss_CarbonatadaAlta_2+Ss_Carbon
atadaAlta_3+Ss_CarbonatadaAlta_4
Ss_CarbonatadaMedia=Ss_CarbonatadaMedia_1+Ss_CarbonatadaMedia_2+Ss_Car
bonatadaMedia_3+Ss_CarbonatadaMedia_4+Ss_CarbonatadaMedia_5
Ss_DetreticaMedia=Ss_DetreticaMedia_1+Ss_DetreticaMedia_2+Ss_Detretica
Media_3+Ss_DetreticaMedia_4+Ss_DetreticaMedia_5+Ss_DetreticaMedia_6
Ss_DetreticaMuyBaja = Ss_DetreticaMuyBaja_1+Ss_DetreticaMuyBaja_2
Ss=Ss_Aluvial+Ss_CarbonatadaAlta+Ss_CarbonatadaMedia+Ss_DetreticaMedia
+Ss_DetreticaMuyBaja
```

xi. Rendimiento Especifico

```
# leyendo el archivo excel:
Excel_Sy_Layer1 = pd.read_excel('../../Inputs/Datos_del_modelo.xlsx',
                                sheet_name='Parametros', usecols=[7])
Excel_Sy_Layer2 = pd.read_excel('../../Inputs/Datos_del_modelo.xlsx',
                                sheet_name='Parametros', usecols=[8])
Excel_Sy_Layer3 = pd.read_excel('../../Inputs/Datos_del_modelo.xlsx',
                                sheet_name='Parametros', usecols=[9])

# Convirtiendo de lista a matriz
Sy_Layer1 = Excel_Sy_Layer1.values.tolist()
Sy_Layer2 = Excel_Sy_Layer2.values.tolist()
Sy_Layer3 = Excel_Sy_Layer3.values.tolist()
#Extrayendo los valores del matriz
```

```
Sy_Layer1_valor=[y for x in Sy_Layer1 for y in x]
Sy_Layer2_valor=[y for x in Sy_Layer2 for y in x]
Sy_Layer3_valor=[y for x in Sy_Layer3 for y in x]

# Creando una funcion para el Rendimiento Especifico :
def Funcion_Sy(i):
    Zonificacion_Geom = Zonificacion_Parametros.iloc[i].geometry
#Extrayendo la propiedad de geometria de los shapefiles por zonas
    Zona_intersected = ix.intersect_polygon(Zonificacion_Geom)
#Intersecando el shapefile y la grilla

    #Integrando las conductividades hidraulicas por zonas:
    Sy_Zona = np.zeros((nlay, nrow, ncol)) #Creando matrices
nlay,nrow,ncol por zonas:
    for j in Zona_intersected.cellids:
        Sy_Zona[0,j[0],j[1]] = Sy_Layer1_valor[i]
        Sy_Zona[1,j[0],j[1]] = Sy_Layer2_valor[i]
        Sy_Zona[2,j[0],j[1]] = Sy_Layer3_valor[i]
    return Sy_Zona

# Rendimiento Especifico por zonas:
Sy_Aluvial_1 = Funcion_Sy(0)
Sy_Aluvial_2 = Funcion_Sy(1)
Sy_Aluvial_3 = Funcion_Sy(2)
Sy_CarbonatadaAlta_1 = Funcion_Sy(3)
Sy_CarbonatadaAlta_2 = Funcion_Sy(4)
Sy_CarbonatadaAlta_3 = Funcion_Sy(5)
Sy_CarbonatadaAlta_4 = Funcion_Sy(6)
Sy_CarbonatadaMedia_1 = Funcion_Sy(7)
Sy_CarbonatadaMedia_2 = Funcion_Sy(8)
Sy_CarbonatadaMedia_3 = Funcion_Sy(9)
Sy_CarbonatadaMedia_4 = Funcion_Sy(10)
Sy_CarbonatadaMedia_5 = Funcion_Sy(11)
Sy_DetreticaMedia_1 = Funcion_Sy(12)
Sy_DetreticaMedia_2 = Funcion_Sy(13)
Sy_DetreticaMedia_3 = Funcion_Sy(14)
Sy_DetreticaMedia_4 = Funcion_Sy(15)
Sy_DetreticaMedia_5 = Funcion_Sy(16)
Sy_DetreticaMedia_6 = Funcion_Sy(17)
Sy_DetreticaMuyBaja_1 = Funcion_Sy(18)
Sy_DetreticaMuyBaja_2 = Funcion_Sy(19)

# Rendimiento Especifico total:
Sy_Aluvial = Sy_Aluvial_1+Sy_Aluvial_2+Sy_Aluvial_3
Sy_CarbonatadaAlta=Sy_CarbonatadaAlta_1+Sy_CarbonatadaAlta_2+Sy_Carbon
atadaAlta_3+Sy_CarbonatadaAlta_4
Sy_CarbonatadaMedia=Sy_CarbonatadaMedia_1+Sy_CarbonatadaMedia_2+Sy_Car
bonatadaMedia_3+Sy_CarbonatadaMedia_4+Sy_CarbonatadaMedia_5
Sy_DetreticaMedia=Sy_DetreticaMedia_1+Sy_DetreticaMedia_2+Sy_Detretica
Media_3+Sy_DetreticaMedia_4+Sy_DetreticaMedia_5+Sy_DetreticaMedia_6
Sy_DetreticaMuyBaja = Sy_DetreticaMuyBaja_1+Sy_DetreticaMuyBaja_2
Sy=Sy_Aluvial+Sy_CarbonatadaAlta+Sy_CarbonatadaMedia+Sy_DetreticaMedia
+Sy_DetreticaMuyBaja
```

xii. Paquete LPF - Parametros

```
laytyp = [1,1,0] # Capa1=libre, Capa2=Libre, Capa3=Confinado
lpf = flopy.modflow.ModflowLpf(mf, hk=Kx, vka=Kx/10, laytyp=laytyp,
#hdry=-1e+30,ss=Ss,sy=Sy,ipakcb=53, chani=-1)
```

xiii. Paquete GHB - Condiciones de borde

```
#Leyendo los shapefiles
Limite_Mira=gpd.read_file("../../Inputs/Data_GIS/Shp/CondiciondeBorde/
GHB_LaMira.shp")
Limite_Bunol_Cheste=gpd.read_file("../../Inputs/Data_GIS/Shp/Condicion
deBorde/GHB_BunolCheste.shp")
```

```
#Intersecando el shapefile y la grilla
Limite_Mira_intersected=ix.intersect_linestring(MultiLineString(Limite
_Mira.geometry.values))
Limite_Bunol_Cheste_intersected=ix.intersect_linestring(MultiLineStrin
g(Limite_Bunol_Cheste.geometry.values))
Data_GHB_Mira=pd.read_excel('../../Inputs/Datos_del_modelo.xlsx',sheet
_name='GHB_Mira')
Data_GHB_Mira = Data_GHB_Mira.values.tolist()

GHB_list = []
for i in Limite_Bunol_Cheste_intersected.cellids:
    GHB_list.append([0,i[0],i[1],300,2.5])
    GHB_list.append([1,i[0],i[1],300,2.5])
    GHB_list.append([2,i[0],i[1],300,2.5])
for i in range(len(Data_GHB_Mira)):
        GHB_list.append([0,Data_GHB_Mira[i][0]-1,Data_GHB_Mira[i][1]-1
                          ,Data_GHB_Mira[i][2],Data_GHB_Mira[i][3]])
        GHB_list.append([1,Data_GHB_Mira[i][0]-1,Data_GHB_Mira[i][1]-1
                          ,Data_GHB_Mira[i][2],Data_GHB_Mira[i][3]])
        GHB_list.append([2,Data_GHB_Mira[i][0]-1,Data_GHB_Mira[i][1]-1
                          , Data_GHB_Mira[i][2],Data_GHB_Mira[i][3]])

GHB_total = {i:GHB_list for i in range(nper)}
ghb =flopy.modflow.mfghb.ModflowGhb(mf, stress_period_data=GHB_total,
ipakcb=53)
```

xiv. Recarga por infiltración

```
# leyendo el archivo excel:
def Funcion_Data_Recargas(ncol):
    Data_Recargas=pd.read_excel('../../Inputs/Datos_del_modelo.xlsx',s
    heet_name='Recargas', usecols=[ncol])
    Data_Recargas = Data_Recargas.values.tolist()
    return Data_Recargas

Data_8103310 = Funcion_Data_Recargas(6)
Data_8103320 = Funcion_Data_Recargas(7)
Data_8103330 = Funcion_Data_Recargas(8)
Data_8103340 = Funcion_Data_Recargas(9)
Data_8103350 = Funcion_Data_Recargas(10)
Data_8103360 = Funcion_Data_Recargas(11)
Data_8103365 = Funcion_Data_Recargas(12)
Data_8103370 = Funcion_Data_Recargas(13)
Data_8103375 = Funcion_Data_Recargas(14)
Data_8103380 = Funcion_Data_Recargas(15)
Data_8103385 = Funcion_Data_Recargas(16)
Data_8103390 = Funcion_Data_Recargas(17)
Data_8103920 = Funcion_Data_Recargas(18)
Data_8103940 = Funcion_Data_Recargas(19)
Data_8103960 = Funcion_Data_Recargas(20)

#Leyendo los shapefiles
GIS_Recarga=gpd.read_file("../../Inputs/Data_GIS/Shp/Recargas/RUCM_Rec
argas.shp")

def Funcion_Recarga(i):  # i: poligono , j:
    Recarga_Geom = GIS_Recarga.iloc[i].geometry
    Recarga_intersected = ix.intersect_polygon(Recarga_Geom)
    Recarga_spd = np.zeros([nrow,ncol])

    for j in Recarga_intersected.cellids:
        Recarga_spd[j] = 1
    return Recarga_spd

# Matrices de ones para conocer la fila y columna en la que se ubica
el valor de recarga
Recarga_8103310 = Funcion_Recarga(0)
```

```
Recarga_8103320 = Funcion_Recarga(1)
Recarga_8103330 = Funcion_Recarga(2)
Recarga_8103340 = Funcion_Recarga(3)
Recarga_8103350 = Funcion_Recarga(4)
Recarga_8103360 = Funcion_Recarga(5)
Recarga_8103365 = Funcion_Recarga(6)
Recarga_8103370 = Funcion_Recarga(7)
Recarga_8103375 = Funcion_Recarga(8)
Recarga_8103380 = Funcion_Recarga(9)
Recarga_8103385 = Funcion_Recarga(10)
Recarga_8103390 = Funcion_Recarga(11)
Recarga_8103920 = Funcion_Recarga(12)
Recarga_8103940 = Funcion_Recarga(13)
Recarga_8103960 = Funcion_Recarga(14)
```

xv. Retornos por riego

```
# leyendo el archivo excel:
def Funcion_Data_Retornos(ncol):
    Data_Retornos=pd.read_excel('../../Inputs/Datos_del_modelo.xlsx',s
    heet_name='Retornos', usecols=[ncol])
    Data_Retornos = Data_Retornos.values.tolist()
    return Data_Retornos

#Convirtiendo de lista a matriz
Retornos_AltoMagro = Funcion_Data_Retornos(6)
Retornos_PlanaUtiel = Funcion_Data_Retornos(7)
Retornos_BunolChiva = Funcion_Data_Retornos(8)

#Leyendo los shapefiles
rch_AltoMagro=gpd.read_file("../../Inputs/Data_GIS/Shp/Retornos/RUCM_R
etorno_AltoMagro_label.shp")
rch_PlanaUtiel=gpd.read_file("../../Inputs/Data_GIS/Shp/Retornos/RUCM_
Retorno_PlanaUtiel_label.shp")
rch_BunolChiva=gpd.read_file("../../Inputs/Data_GIS/Shp/Retornos/RUCM_
Retorno_BunolChiva_label.shp")

def Funcion_Retorno(Retorno_GIS):
    retorno_Geom=[]
    retorno_intersected=[]
    retorno_list=np.zeros([nrow,ncol])
    a=len(Retorno_GIS)
    for i in range(a):
        retorno_Geom.append(Retorno_GIS.iloc[i].geometry)
    for i in range(a):
       retorno_intersected.append(ix.intersect_point(retorno_Geom[i]))
    for i in range(a):
        for j in retorno_intersected[i].cellids:
            retorno_list[j] = 1
    return retorno_list

AltoMagro = Funcion_Retorno(rch_AltoMagro)
PlanaUtiel = Funcion_Retorno(rch_PlanaUtiel)
BunolChiva = Funcion_Retorno(rch_BunolChiva)

retorno_total={i:[Recarga_8103310*Data_8103310[i]+Recarga_8103320*Data
_8103320[i]+Recarga_8103330*Data_8103330[i]+Recarga_8103340*Data_81033
40[i]+Recarga_8103350*Data_8103350[i]+Recarga_8103360*Data_8103360[i]+
Recarga_8103365*Data_8103365[i]+Recarga_8103370*Data_8103370[i]+Recarg
a_8103375*Data_8103375[i]+Recarga_8103380*Data_8103380[i]+Recarga_8103
385*Data_8103385[i]+Recarga_8103390*Data_8103390[i]+Recarga_8103920*Da
ta_8103920[i]+Recarga_8103940*Data_8103940[i]+Recarga_8103960*Data_810
3960[i]+AltoMagro*Retornos_AltoMagro[i]+PlanaUtiel*Retornos_PlanaUtiel
[i]+BunolChiva*Retornos_BunolChiva[i]] for i in range(nper)}
```

xvi. Paquete RCH

```
rch = flopy.modflow.ModflowRch(mf,rech=retorno_total, ipakcb=53)
```

xvii. Paquete RIV

```
#Leyendo los shapefiles
rioDf_Bunol=gpd.read_file("../../Inputs/Data_GIS/Shp/Rios/RUCM_Rio_Bun
ol.shp")
rioDf_LaTorre=gpd.read_file("../../Inputs/Data_GIS/Shp/Rios/RUCM_Rio_L
aTorre.shp")
rioDf_LaMadre=gpd.read_file("../../Inputs/Data_GIS/Shp/Rios/RUCM_Rio_L
aMadre.shp")
rioDf_Magro=gpd.read_file("../../Inputs/Data_GIS/Shp/Rios/RUCM_Rio_Mag
ro.shp")

def Funcion_Rio(Rio_GIS):
    rio río_Geom=[]
    rio río_Elev=[]
    rio río_Cond=[]
    rio río_intersected=[]
    rio río_list=[]
    a=len(Rio_GIS)

    for i in range(a):
        rio río_Geom.append(Rio_GIS.iloc[i].geometry)
        rio río_Elev.append(Rio_GIS.iloc[i].Elevation)
        rio río_Cond.append(Rio_GIS.iloc[i].Conductanc)

    for i in range(a):
        rio río_intersected.append(ix.intersect_point(rio_Geom[i]))

    for j in range(a):
        for i in rio río_intersected[j].cellids:
            if mtop[i]-66 < rio río_Elev[j]: #valores para el layer 1
                          rio
                   río_list.append([0,i[0],i[1],rio_Elev[j]+1,rio_Con
                   d[j],rio_Elev[j]])
            if mtop[i]-66 > rio río_Elev[j]: #valores para el layer 2
                                    río
                   río_list.append([1,i[0],i[1],rio_Elev[j]+1,rio_Con
                   d[j],rio_Elev[j]])
    return rio río_list

rio_Bunol = Funcion_Rio(rioDf_Bunol)
rio_LaTorre = Funcion_Rio(rioDf_LaTorre)
rio_LaMadre= Funcion_Rio(rioDf_LaMadre)
rio_Magro = Funcion_Rio(rioDf_Magro)

rio1 = np.concatenate((rio_Bunol,rio_LaTorre,rio_LaMadre,rio_Magro),0)
rio = {i:rio1 for i in range(nper)}
riv = flopy.modflow.ModflowRiv(mf, stress_period_data=rio, ipakcb=53)
```

xviii. Paquete WELL

```
#Leyendo los shapefiles
RU_Industriales=gpd.read_file("../../Inputs/Data_GIS/Shp/Pozos/RU_Indu
striales.shp")
RU_Urbanas=gpd.read_file("../../Inputs/Data_GIS/Shp/Pozos/RU_Urbanas.s
hp")
RU_Agricolas=gpd.read_file("../../Inputs/Data_GIS/Shp/Pozos/RU_Agricol
as.shp")
CM_Industriales=gpd.read_file("../../Inputs/Data_GIS/Shp/Pozos/CM_Indu
striales.shp")
CM_Urbanas=gpd.read_file("../../Inputs/Data_GIS/Shp/Pozos/CM_Urbanas.s
hp")
CM_Agricolas=gpd.read_file("../../Inputs/Data_GIS/Shp/Pozos/CM_Agricol
as.shp")
RUCM_PozosTotales=gpd.read_file("../../Inputs/Data_GIS/Shp/Pozos/RUCM_
PozosTotales.shp")
```

```
#Leyendo los bombeos desde el excel:
wel_excel = pd.read_csv('../../Inputs/Data_Pozos/Bombeo_RUCM.csv')

#Convirtiendo de lista a matriz
Wel_RU_I = wel_excel['Bombeo'].values.tolist()
b = len(RU_Industriales)
n = len(Wel_RU_I)
Wel_RU=[]
for i in range(nper):
    Wel_RU.append(Wel_RU_I[i:n:nper])

RU_I_Geom=[]
RU_I_Layer=[]
RU_I_intersected=[]
RU_list=[]
RU_F=[]

def Funcion_Well(RU_GIS):
    a=len(RU_GIS)
    for i in range(a):
        RU_I_Geom.append(RU_GIS.iloc[i].geometry)
        RU_I_Layer.append(RU_GIS.iloc[i].Layer)

    for i in range(a):
        RU_I_intersected.append(ix.intersect_point(RU_I_Geom[i]))

    for z in range(nper):
        W = Wel_RU[z]
        for j in range(a):
            for i in RU_I_intersected[j].cellids:
                RU_list.append([RU_I_Layer[j]-1,i[0],i[1],W[j]])

    for i in range(0,len(RU_list),a):
        RU_F.append(RU_list[i:i+a])

    wel_spd = {i:RU_F[i] for i in range(nper)}

    return wel_spd
wel_spd_RU = Funcion_Well(RUCM_PozosTotales)

wel=flopy.modflow.ModflowWel(mf,stress_period_data=wel_spd_RU,ipakcb=5
3)
```

xix. Paquete HOB

```
# leyendo el archivo excel:
Data_Piezometros =
pd.read_csv('../../Inputs/Data_Piezometros/RUCM_Piezometros.csv')

# Función para obtener los registros de medición de piezómetros:
def Funcion_HOB(y):
    PO_tsd = []
    PO = Data_Piezometros[y].values.tolist()
    Perlen_PO = Data_Piezometros['Inicio'].values.tolist()
    for i in range(nper):
        if PO[i]>0:
            PO_tsd.append([Perlen_PO[i],PO[i]])
    return PO_tsd

tsd_1 = Funcion_HOB('08_18_003')
tsd_2 = Funcion_HOB('08_18_005')
tsd_3 = Funcion_HOB('08_18_086')
tsd_4 = Funcion_HOB('08_24_005')
tsd_5 = Funcion_HOB('08_24_007')
tsd_6 = Funcion_HOB('08_24_008')
```

```
tsd_7 = Funcion_HOB('08_24_010')
tsd_8 = Funcion_HOB('08_24_031')
tsd_9 = Funcion_HOB('08_24_032')
tsd_10 = Funcion_HOB('08_24_033')

def Funcion_Mediciones(n,m):
    L = [n]
    k = len(m)
    L1 = ['{}{}'.format(x, y) for y in range(1, k+1) for x in L]
    return L1

Name_1 = Funcion_Mediciones('P0818003.',tsd_1)
Name_2 = Funcion_Mediciones('P0818005.',tsd_2)
Name_3 = Funcion_Mediciones('P0818086.',tsd_3)
Name_4 = Funcion_Mediciones('P0824005.',tsd_4)
Name_5 = Funcion_Mediciones('P0824007.',tsd_5)
Name_6 = Funcion_Mediciones('P0824008.',tsd_6)
Name_7 = Funcion_Mediciones('P0824010.',tsd_7)
Name_8 = Funcion_Mediciones('P0824031.',tsd_8)
Name_9 = Funcion_Mediciones('P0824032.',tsd_9)
Name_10 = Funcion_Mediciones('P0824033.',tsd_10)

#Leyendo los shapefiles
GIS_Piezometros=gpd.read_file("../../Inputs/Data_GIS/Shp/Piezometros/R
UCM_Piezometros.shp")

HOB_Geom=[]
HOB_intersected=[]
HOB_list=[]
HOB_altura=[]

a=len(GIS_Piezometros)
for i in range(a):
    HOB_Geom.append(GIS_Piezometros.iloc[i].geometry)
for i in range(a):
    HOB_intersected.append(ix.intersect_point(HOB_Geom[i]))
for j in range(a):
    for i in HOB_intersected[j].cellids:
        HOB_list.append((i[0],i[1]))

obs_data = []
obs_data.append(flopy.modflow.HeadObservation(mf,layer=0,row=HOB_list[
0][0],column=HOB_list[0][1],time_series_data=tsd_1,names=Name_1,obsnam
e='08_18_003'))
obs_data.append(flopy.modflow.HeadObservation(mf,layer=1,row=HOB_list[
1][0],column=HOB_list[1][1],time_series_data=tsd_2,names=Name_2,obsnam
e='08_18_005'))
obs_data.append(flopy.modflow.HeadObservation(mf,layer=2,row=HOB_list[
2][0],column=HOB_list[2][1],time_series_data=tsd_3,names=Name_3,obsnam
e='08_18_086'))
obs_data.append(flopy.modflow.HeadObservation(mf,layer=0,row=HOB_list[
3][0],column=HOB_list[3][1],time_series_data=tsd_4,names=Name_4,obsnam
e='08_24_005'))
obs_data.append(flopy.modflow.HeadObservation(mf,layer=0,row=HOB_list[
4][0],column=HOB_list[4][1],time_series_data=tsd_5,names=Name_5,
obsname='08_24_007'))
obs_data.append(flopy.modflow.HeadObservation(mf,layer=0,row=HOB_list[
5][0],column=HOB_list[5][1],time_series_data=tsd_6,names=Name_6,
obsname='08_24_008'))
obs_data.append(flopy.modflow.HeadObservation(mf,layer=0,row=HOB_list[
6][0],column=HOB_list[6][1],time_series_data=tsd_7,names=Name_7,
obsname='08_24_010'))
obs_data.append(flopy.modflow.HeadObservation(mf,layer=0,row=HOB_list[
7][0],column=HOB_list[7][1],time_series_data=tsd_8,names=Name_8,
obsname='08_24_031'))
```

```
        obs_data.append(flopy.modflow.HeadObservation(mf,layer=0,row=HOB_list[
        8][0],column=HOB_list[8][1],time_series_data=tsd_9,names=Name_9,
        obsname='08_24_032'))
        obs_data.append(flopy.modflow.HeadObservation(mf,layer=0,row=HOB_list[
        9][0],column=HOB_list[9][1],time_series_data=tsd_10,names=Name_10,obsn
        ame='08_24_033'))

        hob=flopy.modflow.ModflowHob(mf, iuhobsv=42, hobdry=-1e+6, tomulth=1.0,
        obs_data=obs_data)

    xx. Paquete Output Control
        spd = {}
        for i in range(nper):
            for j in range(nstp[i]):
              spd[(i,j)]=['print head', 'print budget', 'save head', 'save
              budget']
        oc= flopy.modflow.ModflowOc(mf,stress_period_data=spd, compact=True)

   xxi. Paquete DE4
        de4 = flopy.modflow.ModflowDe4(mf, itmx=5, mxup=0, mxlow=0, mxbw=0,
                                       ifreq=3, mutd4=0, accl=1, hclose=3e-1,
                                       iprd4=1, extension='de4')

  xxii. Ejecutando el modelo
        mf.write_input()
        mf.run_model(silent=True)

c.     BALANCE POR ZONAS
       # Zona 1: Contorno con Mira.
       # Zona 2: Contorno con Buñol Cheste.
       # Zona 3: Rambla de la Torre.
       # Zona 4: Río Madre.
       # Zona 5: Río Magro.
       # Zona 6: Río Buñol.

       def Funcion_Contorno_zb(zb,nzb): #zb: zonebuget geometry, #nzb: numero de
       zona
           Zona_Contorno = np.zeros((nlay, nrow, ncol), dtype=np.int32)
           for i in zb.cellids:
               Zona_Contorno[0,i[0],i[1]] = nzb
               Zona_Contorno[1,i[0],i[1]] = nzb
               Zona_Contorno[2,i[0],i[1]] = nzb
           return Zona_Contorno

       def Funcion_Rio_zb(Rio_GIS,nzb):
           rio río_Geom=[]
           rio río_Elev=[]
           rio río_intersected=[]
           a=len(Rio_GIS)

           for i in range(a):
               rio río_Geom.append(Rio_GIS.iloc[i].geometry)
               rio río_Elev.append(Rio_GIS.iloc[i].Elevation)
           for i in range(a):
               rio río_intersected.append(ix.intersect_point(rio_Geom[i]))

           Zona_rio = np.zeros((nlay, nrow, ncol),dtype=np.int32)
           for j in range(a):
               for i in rio río_intersected[j].cellids:
                   if mtop[i]-66 < rio río_Elev[j]: #valores para el layer 1
                       Zona_rio[0,i[0],i[1]]=nzb
                   if mtop[i]-66 > rio río_Elev[j]: #valores para el layer 2
                       Zona_rio[1,i[0],i[1]]=nzb
           return Zona_rio
```

```
Zona_1 = Funcion_Contorno_zb(Limite_Mira_intersected, 1)
Zona_2 = Funcion_Contorno_zb(Limite_Bunol_Cheste_intersected, 2)
Zona_3 = Funcion_Rio_zb(rioDf_LaTorre, 3)
Zona_4 = Funcion_Rio_zb(rioDf_LaMadre, 4)
Zona_5 = Funcion_Rio_zb(rioDf_Magro, 5)
Zona_6 = Funcion_Rio_zb(rioDf_Bunol, 6)
zones= (Zona_1+((Zona_6-Zona_6.max())/-Zona_6.max())*Zona_2+((Zona_1-
Zona_1.max())/-Zona_1.max())*Zona_3+Zona_4+Zona_5+Zona_6).astype(int)

# Convirtiendo la matriz en lista
zon_lst = zones.tolist()
zon_lst2 = [num for elem in zon_lst for num in elem]
mylst = [num for elem in zon_lst2 for num in elem]


# Creando el archiVo .zon

def chunk_list(alist, n):
    for i in range(0, len(alist), n):
        yield alist[i:i + n]

def block_gen(mylst, rows, cols, file_cols, field_len):
    frmt = '{:>%d}' % field_len
    zon_str = ''
    for lay in chunk_list(mylst, (rows * cols)):
        zon_str += 'INTERNAL      ({:}I{})\n'.format(file_cols,field_len)
        for block in chunk_list(lay, cols):
            for line in chunk_list(block, file_cols):

                zon_str += ''.join([frmt.format(cell) for cell in line]) +
                '\n'

    return zon_str

def write_zb_zonefile(filepath, lays, rows, cols, zon_str):
    with open(filepath, 'w+') as file:
        file.write('{:<6}{:<6}{:<6}\n'.format(lays, rows, cols))
        file.write(zon_str)
    return

zon_str = block_gen(mylst, nrow, ncol, 10, 3)
write_zb_zonefile((os.path.join(model_ws,modelname+'.zon')), nlay, nrow,
ncol, zon_str)
```

d. BALANCE TOTAL

```
# Convirtiendo la matriz en lista
idomain_zone = idomain.astype(int)
zon_lst_total = idomain_zone.tolist() # Se usa matriz de dominio
zon_lst2_total = [num for elem in zon_lst_total for num in elem]
mylst_total = [num for elem in zon_lst2_total for num in elem]

# Creando el archivo .zon
zon_str_total = block_gen(mylst_total, nrow, ncol, 10, 3)
write_zb_zonefile((os.path.join(model_ws,modelname+'.zontotal')),   nlay,
nrow, ncol, zon_str_total)
```

e. BALANCE POR MASAS SUBTERRANEAS

```
#Leyendo los shapefiles
Masa_RU_GIS= gpd.read_file("../../Inputs/Data_GIS/Shp/Masas/Masa_RU.shp")
Masa_CM_GIS= gpd.read_file("../../Inputs/Data_GIS/Shp/Masas/Masa_CM.shp")

# Creando una funcion para las masas:
def Funcion_Masas(Masa, n):
    Masa_Geom = Masa.iloc[0].geometry #Extrayendo la propiedad de
    geometría de los shapefiles por zonas
    Masa_intersected = ix.intersect_polygon(Masa_Geom) #Intersecando el
    shapefile y la grilla
    Masas_Zonas = np.zeros((nlay, nrow, ncol),dtype=np.int32) #Creando
    matrices nlay,nrow,ncol por zonas:
    for i in Masa_intersected.cellids:
        Masas_Zonas[0,i[0],i[1]] = n
        Masas_Zonas[1,i[0],i[1]] = n
        Masas_Zonas[2,i[0],i[1]] = n
    return Masas_Zonas

Masa_RU = Funcion_Masas(Masa_RU_GIS,1) # 1: Masa Requena-Utiel
Masa_CM = Funcion_Masas(Masa_CM_GIS,2) # 2: Masa Cabrillas-Malacara
Masa_RUCM = Masa_RU + Masa_CM

# Convirtiendo la matriz en lista
Masas_lst = Masa_RUCM.tolist()
Masas_lst2 = [num for elem in Masas_lst for num in elem]
Masas_mylst = [num for elem in Masas_lst2 for num in elem]

# Creando el archivo .zon
Masas_str_total = block_gen(Masas_mylst, nrow, ncol, 10, 3)
write_zb_zonefile((os.path.join(model_ws,modelname+'.zon_masas')),  nlay,
nrow, ncol, Masas_str_total)
```

Anejo 2: Script de generación de gráficos - Pos procesamiento

```
a.    IMPORT BIBILIOTECAS
      import sys
      import os
      import platform
      import numpy as np
      import matplotlib as mpl
      import matplotlib.pyplot as plt
      import flopy.utils.binaryfile as bf
      import flopy
      from ipywidgets import interact

b.    MODELO DE FLUJO
      i. Cargar resultados de modelo de flujo
         model_ws= "../../Outputs/Resultados_Base/"
         modelname= 'RUCM_Base'
         exe_name= '../../Solver/mf2005.exe'
         ml= flopy.modflow.Modflow.load(modelname+'.nam', exe_name=exe_name,
                                        model_ws=model_ws)
         nper = ml.dis.nper # número de periodos
         nrow = ml.dis.nrow # número de filas
         ncol = ml.dis.ncol # número de columnas

      ii. Cargas de alturas piezométrica
          hds = bf.HeadFile(model_ws + '/' + modelname + '.hds')
          head = hds.get_data(totim=31) #tiempo 1
          head[head==-1e+30]=np.nan
          cpth = os.path.join(model_ws, modelname+'.cbc')
          cobj = flopy.utils.CellBudgetFile(cpth, precision=hds.precision)

c.    GRAFICO DE ELEVACION DEL TERRENO

      fig = plt.figure(figsize=(15, 15))
      ax = fig.add_subplot(1, 1, 1, aspect='equal')
      ax.set_title('Model Top Elevations')
      modelmap = flopy.plot.PlotMapView(model=ml)
      quadmesh = modelmap.plot_array(ml.dis.top.array, alpha=1.)
      linecollection = modelmap.plot_grid(linewidth=0.25)
      modelmap.plot_ibound()
```

```
        cb = plt.colorbar(quadmesh, shrink=0.3)
```

d. GRAFICO DE NIVELES PIEZOMETRICO POR CAPA Y PERIODO EN PLANO HORIZONTAL

```
@interact(Capa=(0,2,1),Periodo=[i for i in range(nper)])
def f(Capa,Periodo):
    head=hds.get_data(kstpkper=(0,Periodo))
    head[head==-1e+30]=np.nan
    head[head==-999.99]=np.nan

    fig = plt.figure(figsize=(15, 15))
    ax = fig.add_subplot(1, 1, 1, aspect='equal')
    ax.set_title('Model Heads')
    mapview = flopy.plot.PlotMapView(model=ml,layer=Capa)
    quadmesh = mapview.plot_array(head, alpha= 1.,masked_values=[-999.99,
                -1E+30])
    quadmesh2 = mapview.plot_ibound()
    levels = np.linspace(np.nanmin(head),np.nanmax(head),num=20)
    contour_set  =  mapview.contour_array(head,  levels=levels,  alpha=1.,
                colors='white',masked_values=[-999.99, -1E+30])
    linecollection = mapview.plot_grid(linewidth=0.25)
    plt.clabel(contour_set, fmt='%2d', fontsize=10)
    plt.colorbar(quadmesh, shrink=0.3, format='%2.1f')
    plt.show()
```

e. GRAFICO DE DIRECCNION DEL FLUJO POR CAPA Y PERIODO

```
@interact(Capa=(0,2,1),Periodo=[i for i in range(nper)])
def f(Capa,Periodo):
    head=hds.get_data(kstpkper=(0,Periodo))

    fname = os.path.join(model_ws, modelname+'.cbc')
    cbb = flopy.utils.CellBudgetFile(fname)
    frf = cbb.get_data(text='FLOW RIGHT FACE')[0]
    fff = cbb.get_data(text='FLOW FRONT FACE')[0]

    fig = plt.figure(figsize=(15, 15))

    ax = fig.add_subplot(1, 1, 1, aspect='equal')
    ax.set_title('Volumetric discharge (' + r'$L^3/T$' + ')')
    mapview = flopy.plot.PlotMapView(model=ml,layer=Capa)
    quadmesh = mapview.plot_ibound()
    contour_set = mapview.contour_array(head, masked_values=[-999.99])
    quiver = mapview.plot_discharge(frf, fff)
    linecollection = mapview.plot_grid(linewidth=0.5)
    plt.colorbar(contour_set, shrink=0.5)
```

f. GRAFICO DE CURVAS HIDROISOHIPSAS

```
fig, ax = plt.subplots(figsize=(10,10))
ax.set(aspect='equal')
mapview=flopy.plot.PlotMapView(model=ml,ax=ax,extent=(645000,680000,43600
00,4390000),layer=2)
linecolletion = mapview.plot_grid(linewidth=0.5)
mapview.plot_ibound()
mapview.plot_array(head, masked_values=[-999.99, -1E+30], alpha= 0.5)
levels = np.linspace(np.nanmin(head), np.nanmax(head), num=200)
c = mapview.contour_array(head, levels=levels, colors='white',
masked_values=[-999.99, -1E+30])
plt.clabel(c, fmt='%3d')
plt.show()
plt.clf()
```

g. GRAFICO DE NIVELES PIEZOMETRICOS POR CAPA Y PERIODO EN PLANO VERTICAL

```
@interact(Periodo=[i  for  i  in  range(nper)],  Columna=[i  for  i  in
range(ncol)])
def f(Periodo,Columna):
```

```
head=hds.get_data(kstpkper=(0,Periodo))
head[head==1e+30]=np.nan
head[head==-999.99]=np.nan
    fig = plt.figure(figsize=(20,6))
ax = fig.add_subplot(1, 1, 1)
levels = np.linspace(np.nanmin(500), np.nanmax(head), num=30)
modelxsect=flopy.plot.PlotCrossSection(model=ml,line={'Column':Column
a})
contour_set = modelxsect.contour_array(head, masked_values=[-999.99,-
1E+30], levels=levels,colors='white')
plt.clabel(contour_set, fmt='%1.f', colors='white', fontsize=10)
hv = modelxsect.plot_array(head, head=head, masked_values=[-999.99, -
1E+30])
patches = modelxsect.plot_ibound(head=head)
linecollection = modelxsect.plot_grid(linewidth=1.)
actives = modelxsect.plot_ibound()
wt = modelxsect.plot_surface(head[2], masked_values=[-999.99,-1e+30],
color='blue', lw=1.5)
t = ax.set_title('CROSS SECTION')
fig.colorbar(hv, orientation='vertical', format='%2.1f',shrink=0.75)
```

h. GRAFICO DE DISCRETIZACION ESPACIAL

```
fig = plt.figure(figsize=(12,12))
ax = fig.add_subplot(1,1,1, aspect='equal')
modelmap = flopy.plot.PlotMapView(model=ml, ax=ax)
actives = modelmap.plot_ibound()
linecollection = modelmap.plot_grid(linewidth=0.5)
t = ax.set_title('IBOUND')
```

i. GRAFICO DE DISCRETIZACION VERTICAL

```
fig, ax = plt.subplots(figsize=(15,3))
modelxsect = flopy.plot.PlotCrossSection(model=ml, line={'Row':20})
linecollection = modelxsect.plot_grid()
```

j. GRAFICO GHB

```
fig = plt.figure(figsize=(25,25))
ax = fig.add_subplot(1,2,1, aspect='equal')
modelmap = flopy.plot.PlotMapView(model=ml, ax=ax)
actives = modelmap.plot_ibound()
quadmesh = modelmap.plot_array(ml.ghb.stress_period_data.array['cond'][-
1,0,:,:], cmap='plasma',ax=ax, alpha=1)
linecollection = modelmap.plot_grid(linewidth=0.2)
plt.colorbar(quadmesh, shrink=0.15)
t = ax.set_title('GHB - Conductance')
ax = fig.add_subplot(1,2,2, aspect='equal')
modelmap = flopy.plot.PlotMapView(model=ml, ax=ax)
actives = modelmap.plot_ibound()
quadmesh = modelmap.plot_array(ml.ghb.stress_period_data.array['bhead'][-
1,0,:,:],cmap='plasma',ax=ax, alpha=1)
linecollection = modelmap.plot_grid(linewidth=0.2)
plt.colorbar(quadmesh, shrink=0.15)
t = ax.set_title('GHB - Head')
```

k. GRAFICO RIVER

```
fig = plt.figure(figsize=(25,25))
ax = fig.add_subplot(1, 2, 1, aspect='equal')
modelmap = flopy.plot.PlotMapView(model=ml)
actives = modelmap.plot_ibound()
linecollection = modelmap.plot_grid(linewidth=0.1)
quadmesh = modelmap.plot_array(ml.riv.stress_period_data.array['cond'][-
1,0,:,:], cmap='plasma',ax=ax, alpha=0.5)
plt.colorbar(quadmesh, shrink=0.15)
t = ax.set_title('RIVER - Conductance - Layer 1')
ax = fig.add_subplot(1, 2, 2, aspect='equal')
modelmap = flopy.plot.PlotMapView(model=ml)
```

```
actives = modelmap.plot_ibound()
linecollection = modelmap.plot_grid(linewidth=0.1)
quadmesh = modelmap.plot_array(ml.riv.stress_period_data.array['cond'][-
1,1,:,:], cmap='plasma',ax=ax, alpha=0.5)
plt.colorbar(quadmesh, shrink=0.15)
t = ax.set_title('RIVER - Conductance - Layer 2')
```

l. GRAFICO DE CONDUCTIVIDAD HIDRAULICO

```
fig = plt.figure(figsize=(30,30))
ax = fig.add_subplot(1, 3, 1, aspect='equal')
modelmap = flopy.plot.PlotMapView(model=ml)
linecollection = modelmap.plot_grid(linewidth=0.1)
quadmesh =
modelmap.plot_array(ml.lpf.hk.array[0,:,:],cmap='plasma',ax=ax,
alpha=0.75)
actives = modelmap.plot_ibound()
plt.colorbar(quadmesh, shrink=0.15)
t = ax.set_title('K - Layer 1')
ax = fig.add_subplot(1, 3, 2, aspect='equal')
modelmap = flopy.plot.PlotMapView(model=ml)
linecollection = modelmap.plot_grid(linewidth=0.1)
quadmesh =
modelmap.plot_array(ml.lpf.hk.array[1,:,:],cmap='plasma',ax=ax,
alpha=0.75)
actives = modelmap.plot_ibound()
plt.colorbar(quadmesh, shrink=0.15)
t = ax.set_title('K - Layer 2')
ax = fig.add_subplot(1, 3, 3, aspect='equal')
modelmap = flopy.plot.PlotMapView(model=ml)
linecollection = modelmap.plot_grid(linewidth=0.1)
quadmesh =
modelmap.plot_array(ml.lpf.hk.array[2,:,:],cmap='plasma',ax=ax,
alpha=0.75)
actives = modelmap.plot_ibound()
plt.colorbar(quadmesh, shrink=0.15)
t = ax.set_title('K - Layer 3')
```

m. GRAFICO DE ALMACENAMIENTO ESPECIFICO

```
fig = plt.figure(figsize=(25,25))
ax = fig.add_subplot(1, 3, 1, aspect='equal')
modelmap = flopy.plot.PlotMapView(model=ml)
linecollection = modelmap.plot_grid(linewidth=0.1)
quadmesh=modelmap.plot_array(ml.lpf.ss.array[0,:,:],cmap='plasma',ax=ax,
alpha=1.)
actives = modelmap.plot_ibound()
plt.colorbar(quadmesh, shrink=0.15)
t = ax.set_title('Ss - Layer 1')

ax = fig.add_subplot(1, 3, 2, aspect='equal')
modelmap = flopy.plot.PlotMapView(model=ml)
linecollection = modelmap.plot_grid(linewidth=0.1)
quadmesh=modelmap.plot_array(ml.lpf.ss.array[1,:,:],cmap='plasma',ax=ax,a
lpha=0.75)
actives = modelmap.plot_ibound()
plt.colorbar(quadmesh, shrink=0.15)
t = ax.set_title('Ss - Layer 2')
ax = fig.add_subplot(1, 3, 3, aspect='equal')
modelmap = flopy.plot.PlotMapView(model=ml)
linecollection = modelmap.plot_grid(linewidth=0.1)
quadmesh=modelmap.plot_array(ml.lpf.ss.array[2,:,:],cmap='plasma',ax=ax,
alpha=0.75)
actives=modelmap.plot_ibound()
plt.colorbar(quadmesh, shrink=0.15)
t = ax.set_title('Ss - Layer 3')
```

```
n.     GRAFICO DE RENDIMIENTO ESPECIFICO
       fig = plt.figure(figsize=(25,25))
       ax = fig.add_subplot(1, 3, 1, aspect='equal')
       modelmap = flopy.plot.PlotMapView(model=ml)
       linecollection = modelmap.plot_grid(linewidth=0.1)
       quadmesh =modelmap.plot_array(ml.lpf.sy.array[0,:,:],cmap='plasma',ax=ax,
       alpha=0.75)
       actives = modelmap.plot_ibound()
       plt.colorbar(quadmesh, shrink=0.15)
       t = ax.set_title('Sy - Layer 1')
       ax = fig.add_subplot(1, 3, 2, aspect='equal')
       modelmap = flopy.plot.PlotMapView(model=ml)
       linecollection = modelmap.plot_grid(linewidth=0.1)
       quadmesh =modelmap.plot_array(ml.lpf.sy.array[1,:,:],cmap='plasma',ax=ax,
       alpha=0.75)
       actives = modelmap.plot_ibound()
       plt.colorbar(quadmesh, shrink=0.15)
       t = ax.set_title('Sy - Layer 2')
       ax = fig.add_subplot(1, 3, 3, aspect='equal')
       modelmap = flopy.plot.PlotMapView(model=ml)
       linecollection = modelmap.plot_grid(linewidth=0.1)
       quadmesh=modelmap.plot_array(ml.lpf.sy.array[2,:,:],cmap='plasma',ax=ax,
       alpha=0.75)
       actives = modelmap.plot_ibound()
       plt.colorbar(quadmesh, shrink=0.15)
       t = ax.set_title('Ac. Conf. - Layer 3')
```

Anejo 3: Script de reporte de balance de aguas del sistema

```
a.     IMPORT BIBILIOTECAS
       import flopy, os
       import sys
       import platform
       import numpy as np
       import matplotlib as mpl
       import matplotlib.pyplot as plt
       import pandas as pd
       from seaborn import load_dataset
       from pandas import ExcelWriter
       from pandas import ExcelFile
       from openpyxl import Workbook
       import flopy.utils.binaryfile as bf

b.     MODELO DE FLUJO
     i.  Cargar resultados de modelo de flujo
         model_ws= "../../Outputs/Resultados_Base/"
         modelname= 'RUCM_Base'
         exe_name= '../../Solver/mf2005.exe'
         wb_name = 'WaterBudget_Base'
         ml= flopy.modflow.Modflow.load(modelname+'.nam', exe_name=exe_name,
                                        model_ws=model_ws, check=False)
    ii.  Cargar paquete ZoneBudget
         from flopy.utils.zonbud import ZoneBudget, read_zbarray
         zona_total = read_zbarray(os.path.join(model_ws + '/' + modelname +
                                                   '.zontotal'))
         balance_total = ZoneBudget(model_ws + '/' + modelname + '.cbc',
                                                  zona_total, kstpkper=None)
         df_total = balance_total.get_dataframes()

c.     BALANCE POR MASAS SUBTERRANEAS (HM3)
       zona_masas = read_zbarray(os.path.join(model_ws + '/' + modelname +
       '.zon_masas'))
       balance_masas = ZoneBudget(model_ws + '/' + modelname + '.cbc',
       zona_masas, kstpkper=None)
```

```
df_masas = balance_masas.get_dataframes()

i. BALANCE DE MASAS TOTAL (m3/d)
   # Arreglo del dataframe Zonebudget:
   entrada_m = []
   salida_m = []
   balance_m = []
   error_m = []
   discrep_m = []
   for i in time:
       entrada_m.append(df_masas.loc[(i,'TOTAL_IN'), ['ZONE_1',
                                           'ZONE_2']])
       salida_m.append(df_masas.loc[(i,'TOTAL_OUT'), ['ZONE_1',
                                           'ZONE_2']])
       error_m.append(df_masas.loc[(i,'PERCENT_DISCREPANCY'), ['ZONE_1',
                                           'ZONE_2']])
   for i in range(len(entrada_m)):
       balance_m.append(entrada_m[i]-salida_m[i])
       discrep_m.append((entrada_m[i]-salida_m[i])/entrada_m[i]*100)

   arrays_inout_m = [time, np.array(['IN-OUT']*nper)]
   df1_m = pd.DataFrame(balance_m, index=arrays_inout_m)

   arrays_discrep_m = [time, np.array(['PERCENT_DISCREPANCY']*nper)]
   df2_m = pd.DataFrame(discrep_m, index=arrays_discrep_m)

   # Reemplazando en df:
   for i in time:
       df_masas.loc[(i,'IN-OUT'),['ZONE_1','ZONE_2']] = df1_m.loc[(i,'IN-
                                           OUT'),['ZONE_1','ZONE_2']]
   for i in time:
       df_masas.loc[(i,'PERCENT_DISCREPANCY'),['ZONE_1','ZONE_2']] =
               df2_m.loc[(i,'PERCENT_DISCREPANCY'),['ZONE_1','ZONE_2']]
       df_masas=df_masas.loc[(time,['FROM_STORAGE','FROM_CONSTANT_HEAD',
               'FROM_WELLS','FROM_RIVER_LEAKAGE','FROM_HEAD_DEP_BOUNDS',
               'FROM_RECHARGE','FROM_ZONE_1','FROM_ZONE_2','TOTAL_IN','TO
               _STORAGE','TO_CONSTANT_HEAD','TO_WELLS','TO_RIVER_LEAKAGE'
               , 'TO_HEAD_DEP_BOUNDS', 'TO_RECHARGE','TO_ZONE_1',
               'TO_ZONE_2', 'TOTAL_OUT','IN-OUT','PERCENT_DISCREPANCY']),
               ['ZONE_1','ZONE_2']]

   def fm(x):
       df_1 = df_masas.loc[(time,[x]), :]
       df_2 = df_1.values.tolist()
       df_3 = [y for x in df_2 for y in x]
       return df_3

   perlen_m = []
   for i in range(nper):
       perlen_m.append(ml.dis.perlen[i])
       perlen_m.append(ml.dis.perlen[i])
   Periodo_m = []
   for i in range(nper):
       Periodo_m.append(i+1)
       Periodo_m.append(i+1)
   time_m = []
   for i in range(nper):
       time_m.append(time[i])
       time_m.append(time[i])
   list_zone = []
   for i in range(nper):
       list_zone.append('ZONE 1')
       list_zone.append('ZONE 2')

   lista_m1 = list(zip(Periodo_m, perlen_m, time_m, list_zone,
               fm('FROM_STORAGE'), fm('FROM_CONSTANT_HEAD'),
                fm('FROM_WELLS'),fm('FROM_RIVER_LEAKAGE'),fm('FROM_HEAD_DE
                P_BOUNDS'),fm('FROM_RECHARGE'),fm('FROM_ZONE_1'),
                fm('FROM_ZONE_2'),fm('TOTAL_IN'), fm('TO_STORAGE'),
                fm('TO_CONSTANT_HEAD'), fm('TO_WELLS'),
                ('TO_RIVER_LEAKAGE'), fm('TO_HEAD_DEP_BOUNDS'),
                fm('TO_RECHARGE'),fm('TO_ZONE_1'),fm('TO_ZONE_2'),fm('TOTA
                L_OUT'),fm('IN-OUT'), fm('PERCENT_DISCREPANCY')))
ship_columns_m1 = ['Stress Period', 'Time','Total Time','ZONES','FROM
                STORAGE', 'FROM CONSTANT HEAD', 'FROM WELLS', 'FROM RIVER
                LEAKAGE', 'FROM HEAD DEP BOUNDS', 'FROM RECHARGE',
                 'FROM ZONE 1', 'FROM ZONE 2', 'TOTAL IN (m3/day)', 'TO
                STORAGE','TO CONSTANT HEAD', 'TO WELLS', 'TO RIVER
                LEAKAGE', 'TO HEAD DEP BOUNDS', 'TO RECHARGE', 'TO ZONE
                1', 'TO ZONE 2', 'TOTAL OUT (m3/day)', 'IN-OUT (m3/day)',
                'PERCENT DISCREPANCY']
Balance_masas = pd.DataFrame(columns=ship_columns_m1, data=lista_m1)
```

ii. BALANCE DE MASAS ANUAL - RU (hm3)

```
masa_RU =['Requena-Utiel']*int(len(perlen)/12)
lista_RU = list(zip(Years, masa_RU, fm_anual('FROM_STORAGE','ZONE_1'),
       fm_anual('FROM_CONSTANT_HEAD','ZONE_1'),fm_anual('FROM_WELLS','ZO
       NE_1'),fm_anual('FROM_RIVER_LEAKAGE','ZONE_1'),fm_anual('FROM_HEA
       D_DEP_BOUNDS','ZONE_1'),fm_anual('FROM_RECHARGE','ZONE_1'),fm_anu
       al('FROM_ZONE_2','ZONE_1'),fm_anual('TOTAL_IN','ZONE_1'),fm_anual
       ('TO_STORAGE','ZONE_1'),fm_anual('TO_CONSTANT_HEAD','ZONE_1'),fm_
       anual('TO_WELLS','ZONE_1'),fm_anual('TO_RIVER_LEAKAGE','ZONE_1'),
       fm_anual('TO_HEAD_DEP_BOUNDS','ZONE_1'),fm_anual('TO_RECHARGE','Z
       ONE_1'),fm_anual('TO_ZONE_2','ZONE_1'),fm_anual('TOTAL_OUT','ZONE
       _1'),fm_anual('IN-OUT','ZONE_1'),
       fm_anual('PERCENT_DISCREPANCY','ZONE_1')))
ship_columns_RU = ['Year', 'Masa','FROM STORAGE', 'FROM CONSTANT HEAD',
       'FROM WELLS','FROM RIVER LEAKAGE', 'FROM HEAD DEP BOUNDS', 'FROM
       RECHARGE', 'FROM ZONE 2','TOTAL IN (hm3/year)', 'TO STORAGE', 'TO
       CONSTANT  HEAD',  'TO  WELLS','TO  RIVER  LEAKAGE',  'TO  HEAD  DEP
       BOUNDS', 'TO RECHARGE', 'TO ZONE 2','TOTAL OUT (hm3/year)', 'IN-
       OUT (hm3/year)', 'PERCENT DISCREPANCY']
Balance_RU = pd.DataFrame(columns=ship_columns_RU, data=lista_RU)
```

iii. BALANCE DE MASAS ANUAL - CM (hm3)

```
lista_CM = list(zip(Years, masa_CM, fm_anual('FROM_STORAGE','ZONE_2'),
fm_anual('FROM_CONSTANT_HEAD','ZONE_2'),
                    fm_anual('FROM_WELLS','ZONE_2'),
fm_anual('FROM_RIVER_LEAKAGE','ZONE_2'),fm_anual('FROM_HEAD_DEP_BOUNDS
','ZONE_2'),

fm_anual('FROM_RECHARGE','ZONE_2'),fm_anual('FROM_ZONE_1','ZONE_2'),fm
_anual('TOTAL_IN','ZONE_2'),
                    fm_anual('TO_STORAGE','ZONE_2'),
fm_anual('TO_CONSTANT_HEAD','ZONE_2'), fm_anual('TO_WELLS','ZONE_2'),
                    fm_anual('TO_RIVER_LEAKAGE','ZONE_2'),
fm_anual('TO_HEAD_DEP_BOUNDS','ZONE_2'),
fm_anual('TO_RECHARGE','ZONE_2'),
                    fm_anual('TO_ZONE_1','ZONE_2'),
fm_anual('TOTAL_OUT','ZONE_2'), fm_anual('IN-OUT','ZONE_2'),
                    fm_anual('PERCENT_DISCREPANCY','ZONE_2')))
ship_columns_CM = ['Year', 'Masa','FROM STORAGE', 'FROM CONSTANT
HEAD', 'FROM WELLS',
                'FROM RIVER LEAKAGE', 'FROM HEAD DEP BOUNDS', 'FROM
RECHARGE', 'FROM ZONE 1',
                'TOTAL IN (hm3/year)', 'TO STORAGE', 'TO CONSTANT
HEAD', 'TO WELLS',
                'TO RIVER LEAKAGE', 'TO HEAD DEP BOUNDS', 'TO
RECHARGE', 'TO ZONE 1',
                'TOTAL OUT (hm3/year)', 'IN-OUT (hm3/year)', 'PERCENT
DISCREPANCY']
```

```
    Balance_CM = pd.DataFrame(columns=ship_columns_CM, data=lista_CM)

iv. BALANCE DE MASAS - PROMEDIO ANUAL (hm3)
    # Funcion para los promedios anuales:
    def f_masa_prom(entrada,salida,zone):
        masa_entrada = fm_anual(entrada,zone)
        masa_salida = fm_anual(salida,zone)
        masa_entrada_prom = sum(masa_entrada)/len(masa_entrada)
        masa_salida_prom = sum(masa_salida)/len(masa_salida)
        masa_delta_prom = masa_entrada_prom - masa_salida_prom

        return [masa_entrada_prom, masa_salida_prom, masa_delta_prom]

v.  RU - PROMEDIO ANUAL (hm3)
    RU_prom = [f_masa_prom('FROM_RECHARGE','TO_RECHARGE','ZONE_1'),

    f_masa_prom('FROM_HEAD_DEP_BOUNDS','TO_HEAD_DEP_BOUNDS','ZONE_1'),

    f_masa_prom('FROM_RIVER_LEAKAGE','TO_RIVER_LEAKAGE','ZONE_1'),
               f_masa_prom('FROM_WELLS','TO_WELLS','ZONE_1')]

    entrada_RU_prom = []
    for i in range(len(RU_prom)):
        entrada_RU_prom.append(round((RU_prom[i][0]),2))
    entrada_RU_prom.append(sum(entrada_RU_prom))

    salida_RU_prom = []
    for i in range(len(RU_prom)):
        salida_RU_prom.append(round((RU_prom[i][1]),2))
    salida_RU_prom.append(sum(salida_RU_prom))

    delta_RU_prom = []
    for i in range(len(RU_prom)):
        delta_RU_prom.append(round((RU_prom[i][2]),2))
    delta_RU_prom.append(sum(delta_RU_prom))

    descr_RU_prom = ['Recarga', 'Otras Masas', 'Río - Acuífero',
                 'Bombeo', 'TOTAL']
    lista_RU_prom = list(zip(descr_RU_prom,entrada_RU_prom,
    salida_RU_prom, delta_RU_prom))
    ship_columns_RU_prom = ['Descripción','Entrada (hm3)','Salida
    (hm3)','Delta (hm3)']

    RU_resumen = pd.DataFrame(columns=ship_columns_RU_prom,
    data=lista_RU_prom)
vi. CM - PROMEDIO ANUAL (hm3)
    CM_prom = [f_masa_prom('FROM_RECHARGE','TO_RECHARGE','ZONE_2'),

    f_masa_prom('FROM_HEAD_DEP_BOUNDS','TO_HEAD_DEP_BOUNDS','ZONE_2'),

    f_masa_prom('FROM_RIVER_LEAKAGE','TO_RIVER_LEAKAGE','ZONE_2'),
               f_masa_prom('FROM_WELLS','TO_WELLS','ZONE_2')]

    entrada_CM_prom = []
    for i in range(len(CM_prom)):
        entrada_CM_prom.append(round((CM_prom[i][0]),2))
    entrada_CM_prom.append(sum(entrada_CM_prom))

    salida_CM_prom = []
    for i in range(len(CM_prom)):
        salida_CM_prom.append(round((CM_prom[i][1]),2))
    salida_CM_prom.append(sum(salida_CM_prom))

    delta_CM_prom = []
    for i in range(len(CM_prom)):
        delta_CM_prom.append(round((CM_prom[i][2]),2))
        delta_CM_prom.append(sum(delta_CM_prom))

        descr_CM_prom = ['Recarga', 'Otras Masas', 'Río - Acuífero',
                     'Bombeo', 'TOTAL']
        lista_CM_prom = list(zip(descr_CM_prom,entrada_CM_prom,
        salida_CM_prom, delta_CM_prom))
        ship_columns_CM_prom = ['Descripción','Entrada (hm3)','Salida
        (hm3)','Delta (hm3)']

        CM_resumen = pd.DataFrame(columns=ship_columns_CM_prom,
        data=lista_CM_prom)
```

```
d.    BALANCE TOTAL - Exportando a Excel
      writer_WB = pd.ExcelWriter(model_ws + '/' + wb_name +'_Total.xlsx')
      WB_resumen.to_excel(writer_WB,'PromedioAnual',index=True)
      Balance.to_excel(writer_WB,'m3-d',index=True)
      Balance_2.to_excel(writer_WB,'m3-mes',index=True)
      Balance_3.to_excel(writer_WB,'acum_m3',index=True)
      Balance_4.to_excel(writer_WB,'acum_hm3',index=True)
      Balance_5.to_excel(writer_WB,'hm3-year',index=True)
      writer_WB.save()

e.    BALANCE DE MASAS - Exportando a Excel
      writer_BM = pd.ExcelWriter(model_ws + '/' + wb_name +'_Masas.xlsx')
      Balance_masas.to_excel(writer_BM,'masas_m3-d',index=True)
      RU_resumen.to_excel(writer_BM,'RU_PromedioAnual',index=True)
      CM_resumen.to_excel(writer_BM,'CM_PromedioAnual',index=True)
      Balance_RU.to_excel(writer_BM,'RU_hm3-year',index=True)
      Balance_CM.to_excel(writer_BM,'CM_hm3-year',index=True)
      writer_BM.save()
```

Anejo 4: Script de reporte de balance por zonas

```
a.    IMPORT BIBILIOTECAS
      import flopy, os
      import sys
      import platform
      import numpy as np
      import matplotlib as mpl
      import matplotlib.pyplot as plt
      import pandas as pd
      import flopy.utils.binaryfile as bf

b.    MODELO DE FLUJO
     iii.Cargar resultados de modelo de flujo
        model_ws= "../../Outputs/Resultados_Base/"
        modelname= 'RUCM_Base'
        exe_name= '../../Solver/mf2005.exe'
        zb_name = 'ZoneBudget_Base'
        ml= flopy.modflow.Modflow.load(modelname+'.nam', exe_name=exe_name,
                                       model_ws=model_ws, check=False)

     iv. Cargar paquete ZoneBudget
        from flopy.utils.zonbud import ZoneBudget, read_zbarray
        zon = read_zbarray(os.path.join(model_ws + '/' + modelname + '.zon'))
        zb = ZoneBudget(model_ws + '/' + modelname + '.cbc', zon,
                                                 kstpkper=None)
        df = zb.get_dataframes()

c.    ZONEBUDGET TOTAL (m3/d)
      # Arreglo del dataframe Zonebudget:
      time = ml.dis.get_totim()
      nper = ml.dis.nper
      entrada = []
```

```
salida = []
balance = []
error = []
discrep = []
for i in time:
     entrada.append(df.loc[(i,'TOTAL_IN'),
     ['ZONE_1','ZONE_2','ZONE_3','ZONE_4', 'ZONE_5','ZONE_6']])
         salida.append(df.loc[(i,'TOTAL_OUT'),
     ['ZONE_1','ZONE_2','ZONE_3','ZONE_4', 'ZONE_5','ZONE_6']])
         error.append(df.loc[(i,'PERCENT_DISCREPANCY'),
     ['ZONE_1','ZONE_2','ZONE_3','ZONE_4', 'ZONE_5','ZONE_6']])
for i in range(len(entrada)):
    balance.append(entrada[i]-salida[i])
    discrep.append((entrada[i]-salida[i])/entrada[i]*100)

arrays_inout = [time, np.array(['IN-OUT']*nper)]
df1 = pd.DataFrame(balance, index=arrays_inout)

arrays_discrep = [time, np.array(['PERCENT_DISCREPANCY']*nper)]
df2 = pd.DataFrame(discrep, index=arrays_discrep)

# Reemplazando en df:
for i in time:
    df.loc[(i,'IN-OUT'),'ZONE_1'] = df1.loc[(i,'IN-OUT'),'ZONE_1']
    df.loc[(i,'IN-OUT'),'ZONE_2'] = df1.loc[(i,'IN-OUT'),'ZONE_2']
    df.loc[(i,'IN-OUT'),'ZONE_3'] = df1.loc[(i,'IN-OUT'),'ZONE_3']
    df.loc[(i,'IN-OUT'),'ZONE_4'] = df1.loc[(i,'IN-OUT'),'ZONE_4']
    df.loc[(i,'IN-OUT'),'ZONE_5'] = df1.loc[(i,'IN-OUT'),'ZONE_5']
    df.loc[(i,'IN-OUT'),'ZONE_6'] = df1.loc[(i,'IN-OUT'),'ZONE_6']

for i in time:
    df.loc[(i,'PERCENT_DISCREPANCY'),'ZONE_1'] =
    df2.loc[(i,'PERCENT_DISCREPANCY'),'ZONE_1']
    df.loc[(i,'PERCENT_DISCREPANCY'),'ZONE_2']
    =df2.loc[(i,'PERCENT_DISCREPANCY'),'ZONE_2']
    df.loc[(i,'PERCENT_DISCREPANCY'),'ZONE_3'] =
    df2.loc[(i,'PERCENT_DISCREPANCY'),'ZONE_3']
    df.loc[(i,'PERCENT_DISCREPANCY'),'ZONE_4'] =
    df2.loc[(i,'PERCENT_DISCREPANCY'),'ZONE_4']
    df.loc[(i,'PERCENT_DISCREPANCY'),'ZONE_5'] =
    df2.loc[(i,'PERCENT_DISCREPANCY'),'ZONE_5']
    df.loc[(i,'PERCENT_DISCREPANCY'),'ZONE_6'] =
    df2.loc[(i,'PERCENT_DISCREPANCY'),'ZONE_6']
df.drop(['ZONE_0'], axis = 'columns', inplace=True) # eliminando la
columna zona 0
```

d. ZONEBUDGET POR ZONAS

```
perlen = []
for i in range(nper):
    perlen.append(ml.dis.perlen[i])

Years = []
anio_hidro1 = 1980
anio_hidro2 = 81
for i in range(int(len(perlen)/12)):
    if anio_hidro1 < 1999:
        Years.append(str(anio_hidro1)+'/'+str(anio_hidro2))
    else:
        if anio_hidro1 < 2009:
           Years.append(str(anio_hidro1)+'/'+ str(0)+str(anio_hidro2-
           100))
        else:
           Years.append(str(anio_hidro1)+'/'+str(anio_hidro2-100))

    anio_hidro1 = anio_hidro1 +1
    anio_hidro2 = anio_hidro2+1
```

```
# Funcion para obtener los componentes especificos del balance:
def f_zone(x,zone):
    df_1 = df.loc[(time,[x]), zone]
    df_2 = df_1.values.tolist()
    df_3 = []
    for i in range(len(perlen)):
        df_3.append(df_2[i]*perlen[i])

    df_4 = []
    v = int(len(perlen)/12) # Cada 12 meses
    for i in range(v):
        df_4.append(sum(df_3[12*i:12*(i+1)])/1000000)
    return df_4

# Funcion para calcular el delta del balance:
def f_delta(entrada,salida, zone):
    z_entrada = f_zone(entrada,zone)
    z_salida = f_zone(salida,zone)
    z_delta = []
    for i in range(int(len(perlen)/12)):
        z_delta.append(z_entrada[i]-z_salida[i])
    return z_delta

# Funcion para conocer la tipologia del río:
def f_tipologia(entrada,salida, zone):
    z_entrada = f_zone(entrada,zone)
    z_salida = f_zone(salida,zone)
    z_delta = []
    z_tipologia = []
    for i in range(int(len(perlen)/12)):
        z_delta.append(z_entrada[i]-z_salida[i])
    for i in range(int(len(perlen)/12)):
        if z_delta[i]>0:
            z_tipologia.append('Perdedor')
        else:
            z_tipologia.append('Ganador')
    return z_tipologia
```

i. ZONE 01 - Borde Mira (hm3/año)

```
zone_1 =['Zone 01 - Borde Mira']*int(len(perlen)/12)
lista_z1 = list(zip(Years, zone_1,
f_zone('FROM_HEAD_DEP_BOUNDS','ZONE_1'),
                    f_zone('TO_HEAD_DEP_BOUNDS','ZONE_1'),
f_delta('FROM_HEAD_DEP_BOUNDS','TO_HEAD_DEP_BOUNDS','ZONE_1')))
ship_columns_z1 = ['Year','Zone','Entrada (hm3)','Salida (hm3)','Delta
(hm3)']
ZB_1 = pd.DataFrame(columns=ship_columns_z1, data=lista_z1)
```

ii. ZONE 02 - Borde Buñol Cheste (hm3/año)

```
zone_2 =['Zone 02 - Borde Buñol Cheste']*int(len(perlen)/12)
lista_z2 = list(zip(Years, zone_2,
f_zone('FROM_HEAD_DEP_BOUNDS','ZONE_2'),
                    f_zone('TO_HEAD_DEP_BOUNDS','ZONE_2'),
f_delta('FROM_HEAD_DEP_BOUNDS','TO_HEAD_DEP_BOUNDS','ZONE_2')))
ship_columns_z2 = ['Year','Zone','Entrada (hm3)','Salida (hm3)','Delta
(hm3)']
ZB_2 = pd.DataFrame(columns=ship_columns_z2, data=lista_z2)
```

iii. ZONE 03 - Rambla de la Torre (hm3/año)

```
zone_3 =['Zone 03 - Rambla de la Torre']*int(len(perlen)/12)
lista_z3 = list(zip(Years, zone_3,
f_zone('FROM_RIVER_LEAKAGE','ZONE_3'),
                    f_zone('TO_RIVER_LEAKAGE','ZONE_3'),
f_delta('FROM_RIVER_LEAKAGE','TO_RIVER_LEAKAGE','ZONE_3'),
    f_tipologia('FROM_RIVER_LEAKAGE','TO_RIVER_LEAKAGE','ZONE_3')))
    ship_columns_z3 = ['Year','Zone','Entrada (hm3)','Salida (hm3)',
                       'Delta (hm3)', 'Tipología del río']
    ZB_3 = pd.DataFrame(columns=ship_columns_z3, data=lista_z3)

iv. ZONE 04 - Río Madre (hm3/año)
    zone_4 =['Zone 04 - Río Madre']*int(len(perlen)/12)
    lista_z4 = list(zip(Years, zone_4,
    f_zone('FROM_RIVER_LEAKAGE','ZONE_4'),
                        f_zone('TO_RIVER_LEAKAGE','ZONE_4'),
    f_delta('FROM_RIVER_LEAKAGE','TO_RIVER_LEAKAGE','ZONE_4'),

    f_tipologia('FROM_RIVER_LEAKAGE','TO_RIVER_LEAKAGE','ZONE_4')))
    ship_columns_z4 = ['Year','Zone','Entrada (hm3)','Salida (hm3)',
                       'Delta (hm3)', 'Tipología del río']
    ZB_4 = pd.DataFrame(columns=ship_columns_z4, data=lista_z4)

v.  ZONE 05 - Río Magro (hm3/año)
    zone_5 =['Zone 05 - Río Magro']*int(len(perlen)/12)
    lista_z5 = list(zip(Years, zone_5,
    f_zone('FROM_RIVER_LEAKAGE','ZONE_5'),
                        f_zone('TO_RIVER_LEAKAGE','ZONE_5'),
    f_delta('FROM_RIVER_LEAKAGE','TO_RIVER_LEAKAGE','ZONE_5'),

    f_tipologia('FROM_RIVER_LEAKAGE','TO_RIVER_LEAKAGE','ZONE_5')))
    ship_columns_z5 = ['Year','Zone','Entrada (hm3)','Salida (hm3)',
                       'Delta (hm3)','Tipología del río']
    ZB_5 = pd.DataFrame(columns=ship_columns_z5, data=lista_z5)

vi. ZONE 06 - Río Buñol (hm3/año)
    zone_6 =['Zone 06 - Río Buñol']*int(len(perlen)/12)
    lista_z6 = list(zip(Years, zone_6,
    f_zone('FROM_RIVER_LEAKAGE','ZONE_6'),
                        f_zone('TO_RIVER_LEAKAGE','ZONE_6'),

    f_delta('FROM_RIVER_LEAKAGE','TO_RIVER_LEAKAGE','ZONE_6'),

    f_tipologia('FROM_RIVER_LEAKAGE','TO_RIVER_LEAKAGE','ZONE_6')))
    ship_columns_z6 = ['Year','Zone','Entrada (hm3)','Salida (hm3)',
                       'Delta (hm3)','Tipología del río']
    ZB_6 = pd.DataFrame(columns=ship_columns_z6, data=lista_z6)
vii. Resumen de zonas
    # Funcion para los promedios anuales:
    def f_prom(entrada,salida, zone):
        z_entrada = f_zone(entrada,zone)
        z_salida = f_zone(salida,zone)
        z_entrada_prom = sum(z_entrada)/len(z_entrada)
        z_salida_prom = sum(z_salida)/len(z_salida)
        z_delta_prom = z_entrada_prom - z_salida_prom

        return [z_entrada_prom, z_salida_prom, z_delta_prom]

    prom = [f_prom('FROM_HEAD_DEP_BOUNDS','TO_HEAD_DEP_BOUNDS','ZONE_1'),
            f_prom('FROM_HEAD_DEP_BOUNDS','TO_HEAD_DEP_BOUNDS','ZONE_2'),
           f_prom('FROM_RIVER_LEAKAGE','TO_RIVER_LEAKAGE','ZONE_3'),
           f_prom('FROM_RIVER_LEAKAGE','TO_RIVER_LEAKAGE','ZONE_4'),
           f_prom('FROM_RIVER_LEAKAGE','TO_RIVER_LEAKAGE','ZONE_5'),
           f_prom('FROM_RIVER_LEAKAGE','TO_RIVER_LEAKAGE','ZONE_6')]

    # Valores redondeados a 02 decimales:
    entrada_prom = []
    for i in range(len(prom)):
        entrada_prom.append(round((prom[i][0]),2))
    salida_prom = []
    for i in range(len(prom)):
            salida_prom.append(round((prom[i][1]),2))
        delta_prom = []
        for i in range(len(prom)):
            delta_prom.append(round((prom[i][2]),2))
        tipo_prom = []
        for i in range(len(prom)):
            if i<2:
                tipo_prom.append('')
            else:
                if delta_prom[i]>0:
                    tipo_prom.append('Perdedor')
                else:
                    tipo_prom.append('Ganador')

        zone_prom = ['1','2','3','4','5','6']
        descr_prom = ['Borde Mira', 'Borde Buñol Cheste', 'Rambla de la
        Torre', 'Río Madre', 'Río Magro','Río Buñol']

        lista_prom = list(zip(zone_prom, descr_prom,entrada_prom, salida_prom,
        delta_prom, tipo_prom))
        ship_columns_prom = ['Zona', 'Descripción','Entrada (hm3)','Salida
        (hm3)','Delta (hm3)', 'Tipología del río']

        ZB_resumen = pd.DataFrame(columns=ship_columns_prom, data=lista_prom)
```

```
e.     ZONEBUDGET - Exportando a Excel
       writer_ZB = pd.ExcelWriter(model_ws + '/' + zb_name +'_Total.xlsx')
       ZB_resumen.to_excel(writer_ZB, 'PromedioAnual', index=False)
       df.to_excel(writer_ZB, 'Total_m3-d', index=True)
       ZB_1.to_excel(writer_ZB, 'Zone1', index=False)
       ZB_2.to_excel(writer_ZB, 'Zone2', index=False)
       ZB_3.to_excel(writer_ZB, 'Zone3', index=False)
       ZB_4.to_excel(writer_ZB, 'Zone4', index=False)
       ZB_5.to_excel(writer_ZB, 'Zone5', index=False)
       ZB_6.to_excel(writer_ZB, 'Zone6', index=False)
       writer_ZB.save()
```

Anejo 5: Script de reporte de niveles simulados

```
a.     IMPORT BIBILIOTECAS
       import sys
       import os
       import platform
       import numpy as np
       import pandas as pd
       import matplotlib as mpl
       import matplotlib.pyplot as plt
       import flopy.utils.binaryfile as bf
       import flopy

b.     MODELO DE FLUJO
     i.  Cargar resultados de modelo de flujo
         model_ws= "../../Outputs/Resultados_Base/"
         modelname= 'RUCM_Base'
         exe_name= '../../Solver/mf2005.exe'
         ml= flopy.modflow.Modflow.load(modelname+'.nam', exe_name=exe_name,
                                         model_ws=model_ws, check=False)
     ii. Cargar archivo HDS
         hds = bf.HeadFile(model_ws + '/' + modelname + '.hds')

     iii. Funcion para extraer los heads en cada HOB en todos los Stress
         Periods
         totim = ml.dis.get_totim() #matriz de periodos
         nper = ml.dis.nper #numero de periodos
```

```
    def f(Lay, Row, Col):
        head_ob = []
        period_ob = []

        for i in range(nper): #stress periods
            head=hds.get_data(kstpkper=(0,i))
            head_ob.append(head[Lay][Row][Col])
        return head_ob
iv. Extrayendo los niveles piezometricos
    h_0818003 = f(0,55,103)
    h_0818005 = f(1,10,48)
    h_0818086 = f(2,64,103)
    h_0824005 = f(0,39,60)
    h_0824007 = f(0,43,21)
    h_0824008 = f(0,67,35)
    h_0824010 = f(0,45,71)
    h_0824031 = f(0,36,41)
    h_0824032 = f(0,65,63)
    h_0824033 = f(0,25,45)

    Periodo = [i+1 for i in range(nper)]
    perlen = []
    for i in range(nper):
        perlen.append(ml.dis.perlen[i])

    lista=list(zip(Periodo,perlen,totim,h_0818003,h_0818005,h_0818086,h_08
        24005,h_0824007,h_0824008,h_0824010,h_0824031,
        h_0824032,h_0824033))
    ship_columns = ['Stress Period','Time','Total Time','08_18_003',
        '08_18_005', '08_18_086','08_24_005', '08_24_007', '08_24_008',
        '08_24_010', '08_24_031','08_24_032', '08_24_033']
    Sim = pd.DataFrame(columns=ship_columns, data=lista)

v.  Exportando a CSV
    Sim.to_csv(os.path.join(model_ws, 'Valores_Simulados.csv'))
```