Anexo 2. xmf6.py

Código utilizado para personalizar el estilo de los datos de salida y para la simplificación de los parámetros de entrada en la simulación de flujo y transporte.

```
import os, sys   # Intefaces con el sistema operativo.
import numpy as np # Manejo de arreglos numéricos
↪  multidimensionales
import matplotlib.pyplot as plt # Graficación

# Biblioteca y módulos de flopy
import flopy
from flopy.plot.styles import styles

from colorama import Fore, Style, Back

class MeshDis():
    def __init__(self, nrow = 1, ncol = 1, nlay = 1,
                 column_length = 1.0, row_length = 1.0,
                 top = 0.0, bottom = 0.0):
        self.__nx = ncol  # Number of columns
        self.__ny = nrow  # Number of rows
        self.__nz = nlay  # Number of layers
        self.__lx = row_length
        self.__ly = column_length
        self.__lz = np.abs(top - bottom)
        self.__dx = self.__calc_dx()
        self.__dy = self.__calc_dy()
        self.__dz = 1.0
        self.__top = top       # Top of the model
        self.__bottom = bottom # Layer bottom elevation

    def print(self):
        print('NX : {:8d} (ncol)'.format(self.__nx))
        print('NY : {:8d} (nrow)'.format(self.__ny))
        print('NZ : {:8d} (nlay)'.format(self.__nz))
        print('LX : {:8.5} (row)'.format(self.__lx))
        print('LY : {:8.5} (col)'.format(self.__ly))
        print('LZ : {:8.5} (lay)'.format(self.__lz))
        print('DX : {:8.5} (delr)'.format(self.__dx))
        print('DY : {:8.5} (delc)'.format(self.__dy))
        print('DZ : {:8.5} (top-botm)'.format(self.__dz))


    @property
    def ncol(self):
```

```
    return self.__nx

@ncol.setter
def ncol(self, ncol):
    if ncol > 0:
        self.__nx = ncol
        self.__calc_dx()
    else:
        print('El valor {} no es válido para ncol, debe
        ↪  ser mayor que cero'.format(ncol))

@property
def nrow(self):
    return self.__ny

@nrow.setter
def nrow(self, nrow):
    if nrow > 0:
        self.__ny = nrow
        self.__calc_dy()
    else:
        print('El valor {} no es válido para nrow, debe
        ↪  ser mayor que cero'.format(nrow))

@property
def nlay(self):
    return self.__nz

@nlay.setter
def nlay(self, nlay):
    if nlay > 0:
        self.__nz = nlay
    ## TODO: calcular un delta para la direccion z
    else:
    print('El valor {} no es válido para nlay, debe ser
    ↪  mayor que cero'.format(nlay))

@property
def row_length(self):
    return self.__lx

@row_length.setter
def row_length(self, r_l):
    if r_l > 0:
        self.__lx = r_l
    else:
```

```
            print('El valor {} no es válido para row_length,
            ↪  debe ser mayor que cero'.format(r_l))

    @property
    def col_length(self):
        return self.__ly

    @col_length.setter
    def col_length(self, c_l):
        if c_l > 0:
            self.__ly = c_l
        else:
            print('El valor {} no es válido para col_length,
            ↪  debe ser mayor que cero'.format(c_l))

    @property
    def lay_length(self):
        return self.__lz

    @lay_length.setter
    def lay_length(self, l_l):
        if l_l > 0:
            self.__lz = l_l
        else:
            print('El valor {} no es válido para lay_length,
            ↪  debe ser mayor que cero'.format(l_l))

    @property
    def delr(self):
        return self.__dx

    @property
    def delc(self):
        return self.__dy

    @property
    def delz(self):
        return self.__dz

    @property
    def top(self):
        return self.__top

    @property
    def bottom(self):
        return self.__bottom
```

```
    def __calc_dx(self):
        return self.__lx / self.__nx

    def __calc_dy(self):
        return self.__ly / self.__ny

    def __calc_dz(self):
        return self.__lz / self.__nz

    def get_coords(self):
            return (np.linspace(0.5 * self.__dx, self.__lx -
            ↪  0.5 *self.__dx, self.__nx),
                    np.linspace(0.5 * self.__dy, self.__ly -
                    ↪  0.5 *self.__dy, self.__ny),
                    np.linspace(0.5 * self.__dz, self.__lz -
                    ↪  0.5 *self.__dz, self.__nz))

    def get_grid(self):
        xg, yg, zg = np.meshgrid(np.linspace(0, self.__lx,
        ↪  self.__nx),
                                 np.linspace(0, self.__ly,
                                 ↪  self.__ny),
                                 np.linspace(0, self.__lz,
                                 ↪  self.__nz))

    def get_dict(self):
        return {'row_length': self.__lx, 'col_length':
        ↪  self.__ly, 'lay_length': self.__lz,
                'ncol': self.__nx, 'nrow': self.__ny, 'nlay':
                ↪  self.__nz,
                'delr': self.__dx, 'delc': self.__dy, 'dell':
                ↪  self.__dz,
                'top': self.__top, 'bottom': self.__bottom}

def plot_flow_1D(gwf, mesh, os_par, oc_par, savefig = False):
    """
    Función para graficar los resultados.

    Paramaters
    ----------
    gwf: ModflowGwf
    Objeto del modelo de flujo GWF

    mesh: MeshDis
```

```
Objeto que gestiona atributos y métodos de una malla
↪  rectangular
estructurada y uniforme.

os_par: dict
Parámetros para ejecución de MODFLOW 6, archivos de
↪  salida y
path del workspace.
"""
# Obtenemos los resultados de la carga hidráulica
head = flopy.utils.HeadFile(
    os.path.join(os_par["ws"],
                 oc_par["head_file"])).get_data()

print('head_L = {} \t head_R = {}'.format(head[0,0,0],
↪  head[0,0,-1]))

# Obtenemos los resultados del BUDGET
bud  = flopy.utils.CellBudgetFile(
    os.path.join(os_par["ws"],
                 oc_par["fbudget_file"]),
    precision='double'
)
# Obtenemos las velocidades
spdis = bud.get_data(text='DATA-SPDIS')[0]
qx, qy, qz =
↪  flopy.utils.postprocessing.get_specific_discharge(spdis,
↪  gwf)

with styles.USGSPlot():
    plt.rcParams['font.family'] = 'DeJavu Sans'
    x, _, _ = mesh.get_coords()
    plt.figure(figsize=(10,3))
    plt.plot(x, head[0, 0], marker=".", ls ="-",
    ↪  mec="blue", mfc="none", markersize="1", label =
    ↪  'Head')
    plt.xlim(0, 12)
    plt.xticks(ticks=np.linspace(0, mesh.row_length,13))
    plt.xlabel("Distance (cm)")
    plt.ylabel("Head (unitless)")
    plt.legend()
    plt.grid()

    if savefig:
        plt.savefig('head.pdf')
    else:
```

```
            plt.show()

from wexler1 import sol_analytical_t
def plot_tran_1D(sim, mesh, tm_par, ph_par, os_par, oc_par,
↪  savefig = False):
    """
    Función para graficar los resultados.

    Paramaters
    ----------
    gwf: ModflowGwf
    Objeto del modelo de flujo GWF

    mesh: MeshDis
    Objeto que gestiona atributos y métodos de una malla
    ↪  rectangular
    estructurada y uniforme.

    os_par: dict
    Parámetros para ejecución de MODFLOW 6, archivos de
    ↪  salida y
    path del workspace.
    """
    mf6gwt_ra =
    ↪  sim.get_model("transport").obs.output.obs().data
    ucnobj_mf6 = sim.transport.output.concentration()
    simtimes = mf6gwt_ra["totim"]
    obsnames = ["X005", "X405", "X1105"]

    with styles.USGSPlot():
        plt.rcParams['font.family'] = 'DeJavu Sans'

        fig, axs = plt.subplots(2, 1, figsize=(5,6),
        ↪  tight_layout=True)

        iskip = 5

        atimes = np.arange(0, tm_par["total_time"], 0.1)

        for i, x in enumerate([0.05, 4.05, 11.05]):
            a1, idx_filter = sol_analytical_t(i, x,
            ↪  atimes,mesh, ph_par)

            axs[0].plot(atimes[idx_filter], a1[idx_filter],
            ↪  color="k", label="ANALYTICAL")
```

```
            axs[0].plot(simtimes[::iskip],
            ↪  mf6gwt_ra[obsnames[i]][::iskip],
                        marker="o", ls="none", mec="blue",
                        ↪  mfc="none", markersize="4",
                        label="MODFLOW 6")
            axs[0].set_ylim(-0.05, 1.2)
            axs[0].set_xlim(0, 120)
            axs[0].set_xlabel("Time (seconds)")
            axs[0].set_ylabel("Normalized Concentration
            ↪  (unitless)")

        ctimes = [6.0, 60.0, 120.0]
        x, _, _ = mesh.get_coords()
        for i, t in enumerate(ctimes):
            a1, idx_filter = sol_analytical_t(i, x, t, mesh,
            ↪  ph_par, False)

            axs[1].plot(x, a1, color="k", label="ANALYTICAL")
            simconc = ucnobj_mf6.get_data(totim=t).flatten()
            axs[1].plot(x[::iskip], simconc[::iskip],
                        marker="o", ls="none", mec="blue",
                        ↪  mfc="none", markersize="4",
                        label="MODFLOW 6")
            axs[1].set_ylim(0, 1.1)
            axs[1].set_xlim(0, 12)
            axs[1].set_xlabel("Distance (cm)")
            axs[1].set_ylabel("Normalized Concentration
            ↪  (unitless)")

        if savefig:
            plt.savefig('conc.pdf')
        else:
            plt.show()

def nice_print(dic, message = ''):
    print(Fore.GREEN)
    print(message)
    print('{:^30}'.format(30*'-') + Style.RESET_ALL)
    for k,v in dic.items():
        print('{:>20} = {:<10}'.format(k, v))


if __name__ == '__main__':
    mesh = MeshDis(
        nrow = 1,    # Number of rows
        ncol = 120,  # Number of columns
```

```
    nlay = 1,     # Number of layers
    row_length = 12.0,   # Length of rows
    column_length = 0.1, # Length of columns
    top = 1.0,    # Top of the model
    bottom = 0,  # Layer bottom elevation
)
mesh.print()
nice_print(mesh.get_dict(), 'Space discretization')
```