1 Goals and objectives

The task in this seminar was to analyze the large amount of data. The simulations on the basis of 84 different scenarios provided an output of 8400 digital elevation models (DEM) and 73.6x106 sediment yield values.

The aim of this data analysis is to identify:

1. Profile and elevation difference along the channel (Tool 2)
2. Mean elevation difference over time (Tool 3)
3. Sediment yield and downstream fill over time (Tool 4)
4. Grain-size distribution over time (Tool 5)
5. Assessment of reproducibility (Tool 6)

2 Methods

To process this large amount of data, a program was written to automate the import of the model output files and analyzing and comparing the model output data. The script was written in Python (version 3.6) with the Pycharm editor and the libraries NumPy and Matplotlib.

Different tools have been developed to accomplish the tasks. The following table (Table 1) lists the different tools including a short description.

Tools	**Description**	**Page**
Tool 1	Load in DEMs	2
Tool 2	Spatially distinct change in channel elevation	5
Tool 3	Mean change in channel elevation (over time)	12
Tool 4	Sediment yield with potential channel filling (over time)	27
Tool 5	Grain-size distribution (over time)	42
Tool 6	Comparison and synthetic input	45

Table 1. Different Tools to analyze the provided data

3 Description of tools, scripts and output

The tools are described and explained in more detail in the following chapter. In addition, the scripts and the outputs that have been obtained are presented.

The following abbrevations are used in the script:

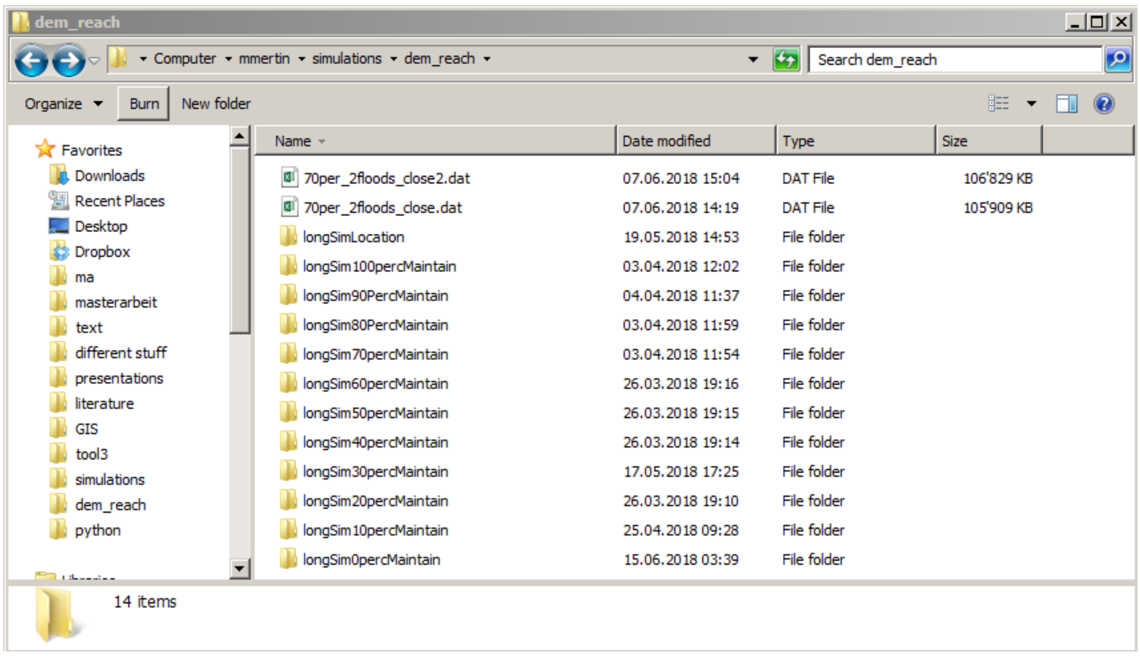
Abbreviation	**Definition**
DEM	Digital elevation model
scn	Scenarios
maint	Maintenance
loc	Location
yrs	Years

Table 2. Used abbreviation in the scripts

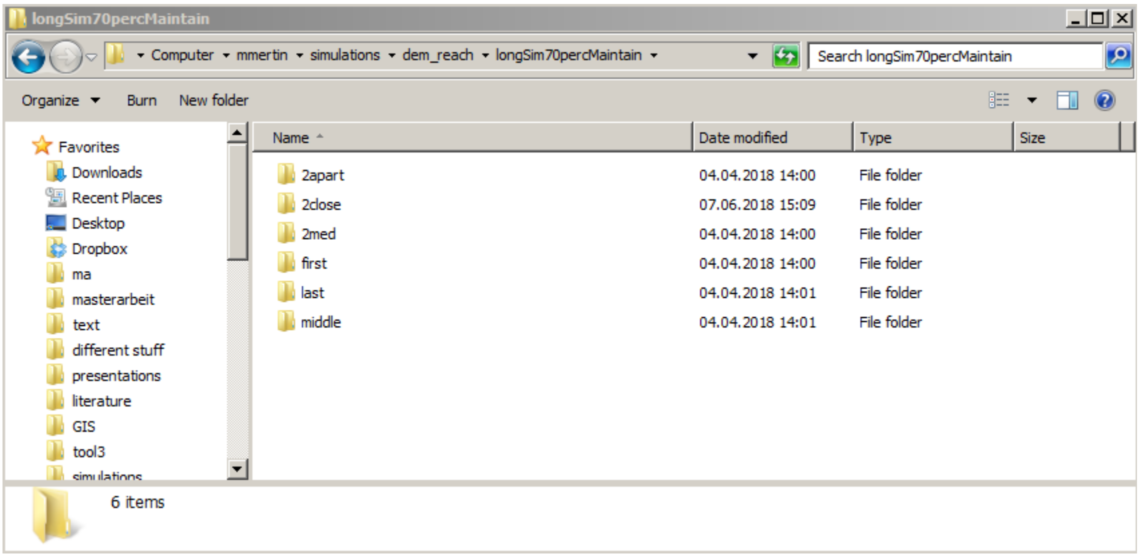
3.1 Tool 1: Load in DEMs

Access files from folders & subfolders: This script imports all files that are stored in different folders and subfolders. The name of these folders and files follow a pattern and can be read in automatically after creating a path to each file. The folders and subfolders represent different scenarios from a simulation and the files are digital elevation models (DEM). The folders differ by the numbers 0-100 with the increment of 10 and three location scenarios. The name of the subfolders is written in the list “floods”.

The folder and subfolder structure:

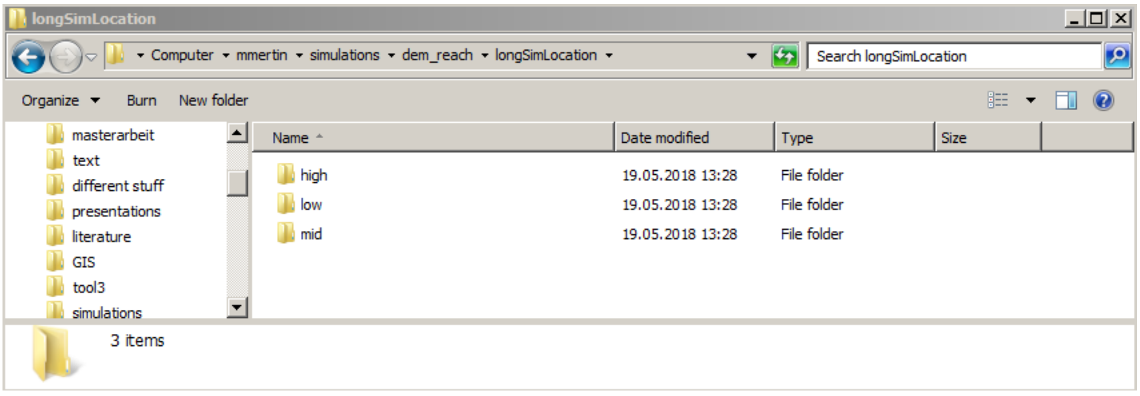


Name	Date modified	Type	Size
70per_2floods_close2.dat	07.06.2018 15:04	DAT File	106'829 KB
70per_2floods_close.dat	07.06.2018 14:19	DAT File	105'909 KB
longSimLocation	19.05.2018 14:53	File folder	
longSim100percMaintain	03.04.2018 12:02	File folder	
longSim90PercMaintain	04.04.2018 11:37	File folder	
longSim80PercMaintain	03.04.2018 11:59	File folder	
longSim70percMaintain	03.04.2018 11:54	File folder	
longSim60percMaintain	26.03.2018 19:16	File folder	
longSim50percMaintain	26.03.2018 19:15	File folder	
longSim40percMaintain	26.03.2018 19:14	File folder	
longSim30percMaintain	17.05.2018 17:25	File folder	
longSim20percMaintain	26.03.2018 19:10	File folder	
longSim10percMaintain	25.04.2018 09:28	File folder	
longSim0percMaintain	15.06.2018 03:39	File folder	



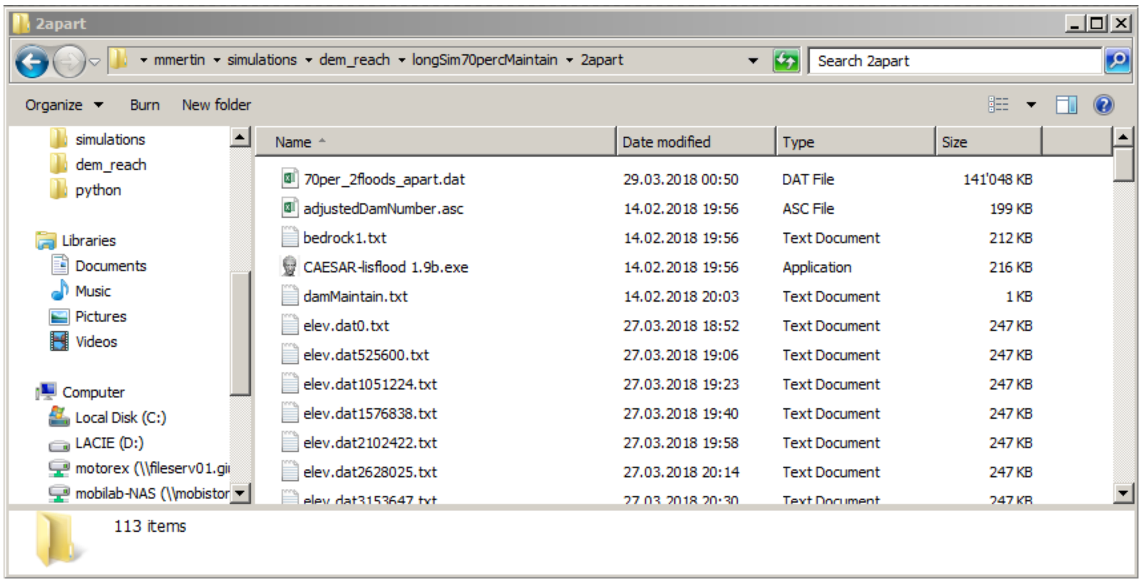
Computer ▸ mmertin ▸ simulations ▸ dem_reach ▸ longSim70percMaintain ▸

Name	Date modified	Type	Size
2apart	04.04.2018 14:00	File folder	
2close	07.06.2018 15:09	File folder	
2med	04.04.2018 14:00	File folder	
first	04.04.2018 14:00	File folder	
last	04.04.2018 14:01	File folder	
middle	04.04.2018 14:01	File folder	



Computer ▸ mmertin ▸ simulations ▸ dem_reach ▸ longSimLocation ▸

Name	Date modified	Type	Size
high	19.05.2018 13:28	File folder	
low	19.05.2018 13:28	File folder	
mid	19.05.2018 13:28	File folder	



mmertin ▸ simulations ▸ dem_reach ▸ longSim70percMaintain ▸ 2apart

Name	Date modified	Type	Size
70per_2floods_apart.dat	29.03.2018 00:50	DAT File	141'048 KB
adjustedDamNumber.asc	14.02.2018 19:56	ASC File	199 KB
bedrock1.txt	14.02.2018 19:56	Text Document	212 KB
CAESAR-lisflood 1.9b.exe	14.02.2018 19:56	Application	216 KB
damMaintain.txt	14.02.2018 20:03	Text Document	1 KB
elev.dat0.txt	27.03.2018 18:52	Text Document	247 KB
elev.dat525600.txt	27.03.2018 19:06	Text Document	247 KB
elev.dat1051224.txt	27.03.2018 19:23	Text Document	247 KB
elev.dat1576838.txt	27.03.2018 19:40	Text Document	247 KB
elev.dat2102422.txt	27.03.2018 19:58	Text Document	247 KB
elev.dat2628025.txt	27.03.2018 20:14	Text Document	247 KB
elev.dat3153647.txt	27.03.2018 20:30	Text Document	247 KB

# ACCESS FILES FROM FOLDERS & SUBFOLDERS: This script imports all files that are
# stored in different folders and subfolders. The name of these folders and files
# follow a pattern and can be read in automatically after creating a path to each
# file. The folders and subfolders represent different scenarios from a simulation
# and the files are digital elevation models (DEM). The folders differ by the num-
# bers 0-100 with the increment of 10 and three location scenarios. the name of the

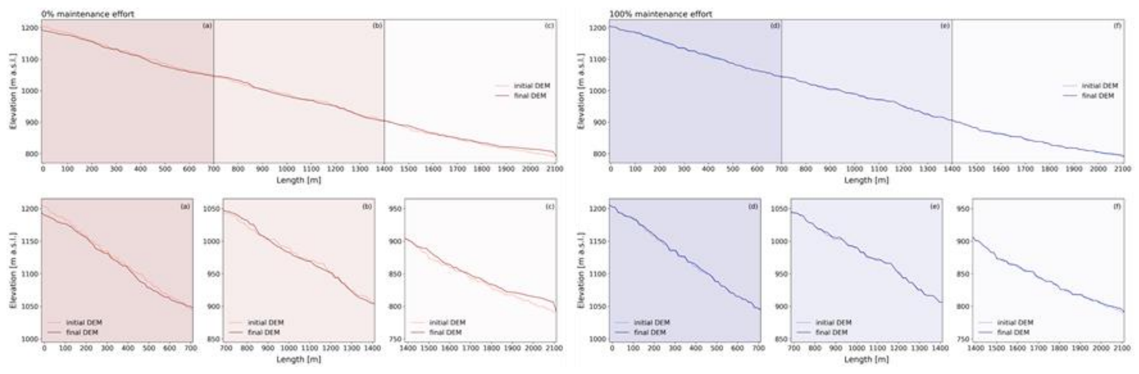
```
# subfolders is written in the list "floods".

# -IMPORT LIBRARIES & VARIABLES HERE-
import glob
import numpy as np

# -DEFINE FUNCTIONS HERE-
def doPaths(floods, maintenances, locations, path, path2):
    '''create list of paths to specific folders and subfolders by changing certain parts of a path'''
    scenario_list = []
    for maintenance in maintenances:                         # loop over maint scn
        for flood in floods:                         # loop over flood scn
            paths = path.format(maintenance, flood)                   # include two arguments to the path
            scenario_list.append(paths)
    for location in locations:
        for flood in floods:                         # loop over loc scn
            paths2 = path2.format(location, flood)
            scenario_list.append(paths2)
    print('\n"path list created"\n')
    return scenario_list
def doElev(length, scenariolist):
    '''get all the files within the specific paths. sort them by string length'''
    elev_list = []
    sorted_list = []
    for x in length:                              # loop over length of list
        elev_list.append(glob.glob(scenariolist[x]))    # import all DEMs within one folder. all DEMs end with .dat(num-
ber)
        sort = sorted(elev_list[x], key=len)
        sorted_list.append(sort)
    elev = np.array(sorted_list)      # change list into array with 101 cols, 66 rows
    print('\n"elev list created"\n"\n')
    return elev
def doDEM(scenario, year, array):
    '''read in all files from created path array. store them in a 4D array'''
    DEM = []
    for row in range(scenario):
        inter_list = []       # use intermediate list to store all files from nested
                                                   loop and later append them to "DEM"
        # skip ArcGIS information. load only cells with c.d.
        for col in range(year):
            read_files = np.genfromtxt(array[row][col], skip_header=6,skip_footer=52, usecols=range(76, 203), delim-
iter=' ')
            inter_list.append(read_files)
        DEM.append(inter_list)
        if (row % 6 == 0):
            print('DEM ' + str(row) + ' is created. Only ' + str(scenario-row) + ' to go!')
    DEM = np.array(DEM)                                                  # convert list into 4D array
    print('\n"YES, all done!"\n')
    return DEM

# -DEFINE GLOBAL VARIABLES HERE-
# define folders and subfolders names. names represent the different flood, maintenance and location scenarios
(snc).
maintenances = range(0, 110, 10)
floods = ['2apart', '2close', '2med', 'first', 'middle', 'last']
locations = ['high', 'mid', 'low']
# define paths of where folders are located
path = 'U:simulations/dem_reach/longSim{}percMaintain/{}/elev.dat*.txt'    # maintenance path which needs
to be adjusted
path2 = 'U:simulations/dem_reach/longSimLocation/{}/{}/elev.dat*.txt'        # location path which needs to be
adjusted

# -CALL FUNCTIONS HERE-
# create path list to each subfolder
path_list = doPaths(floods, maintenances, locations, path, path2)

# get all the DEM files within the subfolder, sort them and store them in a 4D array (scenarios, years, x-elev, y-
elev)
```

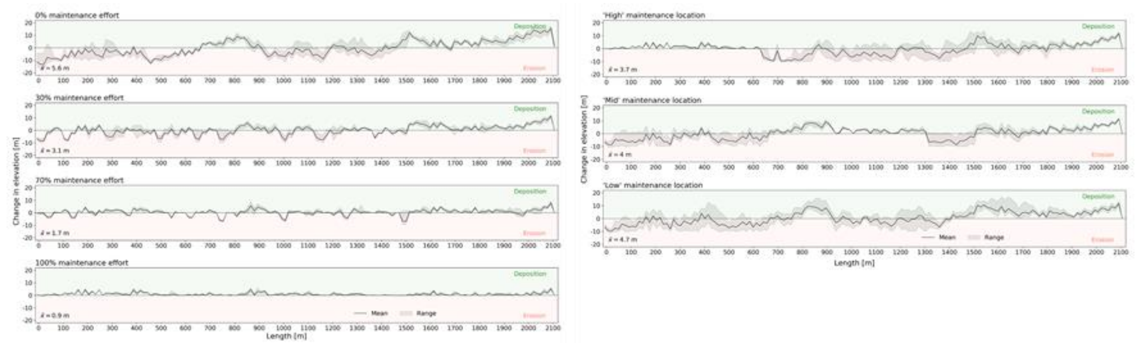
```
#  scenarios: maintenance&location(14)*flood(6)
# years: 100 years of simulation, 1 DEM per year -> 101 DEMs in total
# x-&y-elev: elevation at x-coord, at y-coord
elev = doElev(range(len(path_list)), path_list)

# read in files from the paths created in the array "elev". iterate through ech row and col. nested for-loop procedure:
# take 1st row of the "elev_list" and iterate through all cols, then go on to the 2nd row and iterate rough each col
# again etc. 1st line of output represents the scenario at elev_list[0,0], 2nd line the scenario at elev_list[0,1] etc.
DEM = doDEM(elev.shape[0], elev.shape[1], elev)

# delete variables that are not needed anymore
del(maintenances, floods, locations, path, path2, path_list)
```

3.2 Tool 2: Spatially distinct change in channel elevation

Profile and elevation difference along the channel: This script firstly creates the longitudinal profile of the channel at the beginning and at the end of the simulation. It also cuts the profile into three parts so the differences are better visible. The profiles are generated for 2 maintenance scenarios. More could be added: change in doPlot_prof the index in the variable finalDEM[i] to the requested scenario (0=0% maint, 1=10% maint etc.). Secondly, the spatially distributed elevation differences are calculated. This for 4 different maint scn. Same here, the *number of these can be changed in the doPlot_diff function (DEMdiff[i]). Additionally, the number and the relative share of cells which are below a certain erosion/deposition threshold can be calculated (e.g. x% of all values lie below n)*

The following script produces these figures:





```
# PROFILE & ELEVATION DIFFERENCE ALONG THE CHANNEL: This script firstly creates the
# longitudinal profile of the channel at the beginning and at the end of the simu-
# lation. It also cuts the profile into three parts so the differences are better
# visible. The profiles are generated for two maintenance scenarios. more could be
# added: change in 'doPlot_prof' the index in the variable `finalDEM[i]` to the re-
# quested scenario (0=0% maint, 1=10% maint etc.). Secondly, the spatially distrib-
# uted elevation differences are calculated. This for 4 different maintenance scn.
# Same here, the number of these can be changed in the `doPlot_diff` function
# ('DEMdiff[i]'). Additionally, the number and the relative share of cells which
# are below a certain erosion/deposition threshold can be calculated (e.g. x% of
# all values lie below n)

# -IMPORT LIBRARIES & VARIABLES HERE-
import numpy as np
import matplotlib.pyplot as plt
from tool2a_openDEMs import DEM

# -DEFINE FUNCTIONS HERE-
def doDEMdiff(scenarios):
    '''creates the difference for each cell between predefined years for each scenario'''
    DEMdiff = []
    for scenario in range(scenarios):
        DEMdiff_list = DEM[scenario, 100, :] - DEM[scenario, 0, :]
        DEMdiff.append(DEMdiff_list)
    DEMdiff = np.array(DEMdiff)

    DEMdiffzero = []  # tranfer zero values into nan
    for scenario in range(DEM.shape[0]):
        DEMdiff0 = np.where(DEMdiff[scenario, :, :] == 0, np.nan, DEMdiff[scenario, :, :])
        DEMdiffzero.append(DEMdiff0)
    DEMdiffzero = np.array(DEMdiffzero)
    print('difference calculations finished''\n')
    return DEMdiffzero
def doProfile_prof (in1, in2, scenarios, DEM):
    '''mask all generated arrays with the thalweg array, so only the values that belong to the thalweg are analyzed'''
    # load in thalweg file, created in ArcGIS with flow accumulation, which has the same extent as the "cut" DEM
    profile = np.genfromtxt(in1, skip_header=6, delimiter=' ')
    # load in initial DEM in the same extent as the other DEMs
    start = np.genfromtxt(in2, skip_header=6, skip_footer=52, usecols=range(76, 203), delimiter=' ')
    # index array to switch order of rows from last to first
    index = np.arange(profile.shape[0]-1, -1, -1)

    # create profile for the initial DEM (the same for all scenarios)
    thal_start = np.where(profile == True, start, np.nan)         # use thalweg as mask to only get DEM values from
thalweg
    thal_start = thal_start[index, :]                          # switch order of rows with index array
    thal_start = np.array((thal_start[~np.isnan(thal_start)]))   # only get values that are not nan (~ opposite of is.nan)

    # create profile for the final DEM (loop over all 84 scenarios)
    thal = []
    for x in range(scenarios):
        thal_0 = np.where(profile == True, DEM[x, 100, :, :], np.nan)
        thal.append(thal_0)
    thal = np.array(thal)

    thal_i = []
    for x in range(scenarios):
        i = thal[x, index, :]
        thal_i.append(i)
    thal_i = np.array(thal_i)

    thal_end = []
    for x in range(scenarios):
        thal_e = np.array((thal_i[x, :, :][~np.isnan(thal_i[x, :, :])]))
        thal_end.append(thal_e)
    thal_end = np.array(thal_end)
    print('profile built along thalweg''\n')
    return thal_start, thal_end
def doProfile_diff (paths, scenarios, DEMdiff):
    '''mask all generated arrays with the thalweg array, so only the values that belong to the thalweg are analyzed'''
    # load in thalweg file, created in ArcGIS with flow accumulation, which has the same extent as the DEM
    thalweg = np.genfromtxt(paths, skip_header=6, delimiter=' ')

    index = np.arange(thalweg.shape[0]-1, -1, -1)
    # create profile for the final DEM (loop over all 84 scenarios)
    thal = []
    for x in range(scenarios):
        thal_0 = np.where(thalweg == True, DEMdiff[x, :, :], np.nan)
        thal.append(thal_0)
    thal = np.array(thal)

    thal_i = []
    for x in range(scenarios):
        i = thal[x, index, :]
        thal_i.append(i)
    thal_i = np.array(thal_i)

    thal = []
    for x in range(scenarios):
        thal_e = np.array((thal_i[x, :, :][~np.isnan(thal_i[x, :, :])]))
        thal.append(thal_e)
    thal = np.array(thal)

    print('profile built along thalweg''\n')
    return thal
def doNewArray(input):
    '''create arrays to original elev array: specific maintenance scn ("perc"), location scn ("loc"), flood scn ("flood")'''
    perc_loc = np.repeat(np.arange(0, 7, 0.5), 6).reshape(input.shape[0], 1)
    new_array = np.append(np.vstack(input), perc_loc, axis=1)  # combine the new created perc_loc
    floods = np.array(14 * ['2apart', '2close', '2med', 'a_first', 'b_middle', 'c_last']).reshape(input.shape[0], 1)
    new_array = np.append(new_array, floods, axis=1)  # append third column to STD array
    return new_array
def doMean(prof):
    '''create new array which is sorted in the right way for analysis'''
    new_array = doNewArray(prof)

    # sort array by maintenance scn
    a = new_array[:, range(0, prof.shape[1])]
    b = new_array[:, -1]    # flood scn
    c = new_array[:, -2]    # maint scn

    sorts = []
    for x in range(prof.shape[1]):
        ind = np.lexsort((a[:, x], b, c))  # create array with the specified order
        sort = np.array([(a[:, x][i], b[i], c[i]) for i in ind])  # apply the "sorting array" to the original array
        sorts.append(sort)

    sorts = np.array(sorts)
    comb = np.concatenate(sorts[:, :, 0]).reshape(prof.shape[1], prof.shape[0])

    split = np.array(np.split(comb[:, :], 14, axis=1))                  # split array into different flood scenarios (6)

    # calculate mean of flood scn for different maintenance scn
    mean = []
    for x in range(split.shape[0]):  # loop through all scenarios
        interlist = []
        for y in range(split.shape[1]):
            mean_m = np.mean((split[x, y, :]).astype('float'))  # build mean of DEMdiff for each cell during 100yrs
            interlist.append(mean_m)
        mean.append(interlist)
    mean = np.array(mean)

    # calculate min + max of flood scn for different maintenance scn
    min = []
    for x in range(split.shape[0]):  # loop through all scenarios
        interlist = []
        for y in range(split.shape[1]):
            mean_m = np.min((split[x, y, :]).astype('float'))  # build mean of DEMdiff for each cell during 100yrs
            interlist.append(mean_m)
        min.append(interlist)
    min = np.array(min)
    max = []
    for x in range(split.shape[0]):  # loop through all scenarios
        interlist = []
        for y in range(split.shape[1]):
            mean_m = np.max((split[x, y, :]).astype('float'))  # build mean of DEMdiff for each cell during 100yrs
            interlist.append(mean_m)
        max.append(interlist)
    max = np.array(max)

    print('flood mean, min and max calculated.''\n')
    return mean, min, max
def doPlot_prof(initialDEM, finalDEM, xlabel, ylabel, ax_size, l_size, title, save1, save2):
    '''plot the total longitudinal profile of two maintenance scn (0+100% maintenance)'''
    legend = np.array(['final DEM', 'initial DEM'])
    # plot 0% maintenance effort
    plt.figure(figsize=(19, 12))
    # plt.suptitle(title, fontsize=24, fontweight=1, color='black').set_position([.5, 0.96])
    plt.subplots_adjust(hspace=0.25)
    plt.subplots_adjust(left=0.06, bottom=0.07, right=0.9, top=0.9, hspace=0.25)

    plt.subplot(211)
    plt.xticks(np.arange(0.5,  460, 7.1), ('0', '100', '200', '300', '400', '500', '600', '700', '800', '900',
                                   '1000', '1100', '1200', '1300', '1400', '1500', '1600', '1700', '1800',
                                   '1900', '2000', '2100'), fontsize=l_size)
    plt.plot(initialDEM, linewidth=1, color='tomato', linestyle='--', label=legend[1])
    plt.plot(finalDEM[0], linewidth=1, color='maroon', label=legend[0])
    plt.legend(loc='right', fontsize=l_size, frameon=False)
    plt.title('0% maintenance effort', fontsize=ax_size, loc='left')
    plt.xlim(0, 150)
    plt.axvline(x=50.2, color='black', linewidth=0.7)
    plt.axvline(x=99.9, color='black', linewidth=0.7)
    plt.axvspan(0, 50.2, color='maroon', alpha=0.14, lw=0)
    plt.axvspan(50.2, 100, color='maroon', alpha=0.07, lw=0)
    plt.axvspan(100, 150, color='maroon', alpha=0.015, lw=0)
    plt.text(47, 1200, '(a)', fontsize=l_size)
    plt.text(96.5, 1200, '(b)', fontsize=l_size)
    plt.text(146.7, 1200, '(c)', fontsize=l_size)
    plt.xlabel(xlabel, labelpad=9, fontsize=ax_size)
    plt.ylabel(ylabel, labelpad=8, fontsize=ax_size)
    plt.yticks(fontsize=l_size)

    # zoom in on 3 channel sections
    plt.subplot(234)
    plt.xticks(np.arange(1, 60, 6.9), ('0', '100', '200', '300', '400', '500', '600', '700'), fontsize=l_size)
    plt.plot(initialDEM, linewidth=1, color='tomato', linestyle='--', label=legend[1])
    plt.plot(finalDEM[0], linewidth=1, color='maroon', label=legend[0])
    plt.xlim(0, 50)
    plt.ylim(995, 1215)
    plt.legend(loc='lower left', fontsize=l_size, frameon=False)
    plt.text(46, 1200, '(a)', fontsize=l_size)
    plt.ylabel(ylabel, labelpad=8, fontsize=ax_size)
    plt.yticks(np.arange(1000, 1250, 50), np.arange(1000, 1250, 50), fontsize=l_size)
    plt.axvspan(0, 60, color='maroon', alpha=0.14, lw=0)

    plt.subplot(235)
    plt.xticks(np.arange(51, 110, 6.9), ('700', '800', '900', '1000', '1100', '1200', '1300', '1400'), fontsize=l_size)
    plt.plot(initialDEM, linewidth=1, color='tomato', linestyle='--', label=legend[1])
    plt.plot(finalDEM[0], linewidth=1, color='maroon', label=legend[0])
    plt.xlim(50, 100)
    plt.ylim(845, 1065)
    plt.legend(loc='lower left', fontsize=l_size, frameon=False)
    plt.text(96, 1050, '(b)', fontsize=l_size)
    plt.xlabel(xlabel, labelpad=9, fontsize=ax_size)
    plt.yticks(np.arange(850, 1100, 50), range(850, 1100, 50), fontsize=l_size)
    plt.axvspan(50, 120, color='maroon', alpha=0.07, lw=0)

    plt.subplot(236)
    plt.xticks(np.arange(101, 160, 6.9), ('1400', '1500', '1600', '1700', '1800', '1900', '2000', '2100'), fontsize=l_size)
    plt.plot(initialDEM, linewidth=1, color='tomato', linestyle='--', label=legend[1])
    plt.plot(finalDEM[0], linewidth=1, color='maroon', label=legend[0])
    plt.xlim(100, 150)
    plt.ylim(745, 965)
    plt.legend(loc='lower left',  fontsize=l_size, frameon=False)
    plt.text(146.5, 950,'(c)', fontsize=l_size)
    plt.yticks(np.arange(750, 1000, 50), range(750, 1000, 50), fontsize=l_size)
    plt.axvspan(100, 170, color='maroon', alpha=0.015, lw=0)

    plt.savefig(save1, dpi=300, bbox_inches='tight')

    # plot 100% maintenance effort
    plt.figure(figsize=(19, 12))
    # plt.suptitle(title, fontsize=24, fontweight=1, color='black').set_position([.5, 0.96])
    plt.subplots_adjust(left=0.06, bottom=0.07, right=0.9, top=0.9, hspace=0.25)
    plt.subplot(211)
    plt.xticks(np.arange(0.5, 460, 7.1), ('0', '100', '200', '300', '400', '500', '600', '700', '800', '900',
                                          '1000', '1100', '1200', '1300', '1400', '1500', '1600', '1700', '1800',
                                          '1900', '2000', '2100'), fontsize=l_size)
    plt.plot(initialDEM,linewidth=1, color='royalblue', linestyle='--', label=legend[1])
    plt.plot(finalDEM[10], linewidth=1, color='navy', label=legend[0])
    plt.legend(loc='right', fontsize=l_size, frameon=False)
    plt.title('100% maintenance effort', fontsize=ax_size, loc='left')
    plt.axvline(x=50.2, color='black', linewidth=0.6)
    plt.axvline(x=99.9, color='black', linewidth=0.6)
    plt.axvspan(0, 50.2, color='navy', alpha=0.13, lw=0)
    plt.axvspan(50.2, 100, color='navy', alpha=0.07, lw=0)
    plt.axvspan(100, 150, color='navy', alpha=0.015, lw=0)
    plt.text(47, 1200,'(d)', fontsize=l_size)
    plt.text(96.5, 1200,'(e)', fontsize=l_size)
    plt.text(147, 1200,'(f)', fontsize=l_size)
    plt.xlabel(xlabel, labelpad=9, fontsize=ax_size)
    plt.ylabel(ylabel, labelpad=8, fontsize=ax_size)
    plt.xlim(0, 150)
    plt.yticks(fontsize=l_size)

    # zoom in on 3 channel sections
    plt.subplot(234)
    plt.xticks(np.arange(1, 60, 6.9), ('0', '100', '200', '300', '400', '500', '600', '700'), fontsize=l_size)
    plt.plot(initialDEM, linewidth=1, color='royalblue', linestyle='--', label=legend[1])
    plt.plot(finalDEM[10], linewidth=1, color='navy', label=legend[0])
    plt.xlim(0, 50)
    plt.ylim(995, 1215)
    plt.legend(loc='lower left', fontsize=l_size, frameon=False)
    plt.text(46, 1200, '(d)', fontsize=l_size)
    plt.ylabel(ylabel, labelpad=8, fontsize=ax_size)
    plt.yticks(np.arange(1000, 1250, 50), np.arange(1000, 1250, 50), fontsize=l_size)
    plt.axvspan(0, 60, color='navy', alpha=0.13, lw=0)

    plt.subplot(235)
    plt.xticks(np.arange(51, 110, 6.9), ('700', '800', '900', '1000', '1100', '1200', '1300', '1400'), fontsize=l_size)
    plt.plot(initialDEM, linewidth=1, color='royalblue', linestyle='--', label=legend[1])
    plt.plot(finalDEM[10], linewidth=1, color='navy', label=legend[0])
    plt.xlim(50, 100)
    plt.ylim(845, 1065)
    plt.legend(loc='lower left', fontsize=l_size, frameon=False)
    plt.text(96, 1050, '(e)', fontsize=l_size)
    plt.xlabel(xlabel, labelpad=9, fontsize=ax_size)
    plt.yticks(np.arange(850, 1100, 50), range(850, 1100, 50), fontsize=l_size)
    plt.axvspan(50, 120, color='navy', alpha=0.07, lw=0)

    plt.subplot(236)
    plt.xticks(np.arange(101, 160, 6.9), ('1400', '1500', '1600', '1700', '1800', '1900', '2000', '2100'), fontsize=l_size)
    plt.plot(initialDEM, linewidth=1, color='royalblue', linestyle='--', label=legend[1])
    plt.plot(finalDEM[10], linewidth=1, color='navy', label=legend[0])
    plt.xlim(100, 150)
    plt.ylim(745, 965)
    plt.legend(loc='lower left', fontsize=l_size, frameon=False)
```

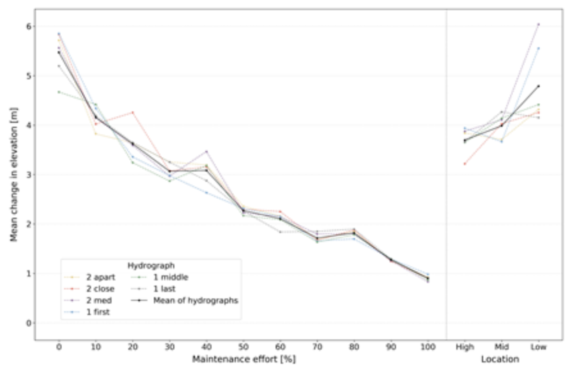
```
    plt.text(146.5, 950, '(f)', fontsize=l_size)
    plt.axvspan(100, 170, color='navy', alpha=0.015, lw=0)
    plt.yticks(np.arange(750, 1000, 50), range(750, 1000, 50), fontsize=l_size)
    plt.savefig(save2, dpi=300, bbox_inches='tight')
    print('longitudinal profile plotted''\n')
def doPlot_diff(mean, min, max, xlabel, ylabel, l_size, ax_size, title1, title2, title3, save1, save2):
    '''plot the elevation difference along the longitudinal profile of the channel for four different maintenance scn'''
    # maint scn
    fig = plt.figure(figsize=(19, 12))
    # fig.suptitle(title1, fontsize=24, fontweight=1, color='black').set_position([.5, 0.96])
    ax = plt.axes([0, 0, 1, 1], frameon=False)
    ax.axes.get_xaxis().set_visible(False)
    ax.axes.get_yaxis().set_visible(False)
    fig.text(0.5, 0.051, xlabel, ha='center', fontsize=ax_size)
    fig.text(0.006, 0.5, ylabel, va='center', rotation='vertical', fontsize=ax_size)
    # fig.text(0.5, 0.91, title2, ha='center', fontsize=ax_size, style='italic')

    maint = [0, 3, 7, 10]
    maintenance = ['0% maintenance effort', '30% maintenance effort', '70% maintenance effort', '100% 
maintenance effort']
    sigma = [r'$\bar x=5.6$ m', r'$\bar x=3.1$ m', r'$\bar x=1.7$ m', r'$\bar x=0.9$ m']
    label = ['(a)', '(b)', '(c)', '(d)']

    for x in maint:
        y = maint.index(x)
        ax = fig.add_subplot(4, 1, y+1, sharey=ax)
        # fig.tight_layout()
        plt.subplots_adjust(left=0.046, bottom=0.096, right=0.99, top=0.96, hspace=0.5)
        plt.plot(mean[x], linewidth=1, color='black', linestyle='-', label='Mean')
        plt.plot(min[x], linewidth=0.25, color='grey', linestyle='-')
        plt.plot(max[x], linewidth=0.25, color='grey', linestyle='-')
        plt.fill_between(np.arange(min.shape[1]), max[x, range(min.shape[1])], min[x, range(min.shape[1])],
                         color='grey', alpha=0.18, label='Range')
        plt.xticks(np.arange(0.5, 460, 7.1), np.arange(0, 2200, 100), fontsize=l_size)
        plt.yticks(np.arange(-20, 30, 10), np.arange(-20, 30, 10), fontsize=l_size)
        plt.title(maintenance[y], fontsize=ax_size, loc='left')
        plt.xlim(-.5, 151)
        plt.ylim(-22, 22)
        plt.axvspan(0, 50.2, color='grey', alpha=0.0, lw=0)
        plt.axhline(y=0, color='black', linewidth=0.6)
        plt.text(1, -18, sigma[y], fontsize=l_size)
        plt.axhspan(-.5, 22, color='green', alpha=0.04, lw=0)
        plt.axhspan(-.5, -22, color='tomato', alpha=0.04, lw=0)
        plt.text(141, -18, 'Erosion', alpha=0.85, color='tomato', fontsize=l_size)
        plt.text(138.3, 15.5, 'Deposition', alpha=0.85, color='green', fontsize=l_size)
    plt.legend(ncol=2, fontsize=l_size, framealpha=0, bbox_to_anchor=(0.6, 0.012), loc=3)
    plt.savefig(save1, dpi=300, bbox_inches='tight')

    # loc scn
    fig = plt.figure(figsize=(19, 12))
    # fig.suptitle(title, fontsize=24, fontweight=1, color='black').set_position([.5, 0.96])
    ax = plt.axes([0, 0, 1, 1], frameon=False)
    ax.axes.get_xaxis().set_visible(False)
    ax.axes.get_yaxis().set_visible(False)
    fig.text(0.5, 0.26, xlabel, ha='center', fontsize=ax_size)
    fig.text(0.006, 0.62, ylabel, va='center', rotation='vertical', fontsize=ax_size)
    # fig.text(0.5, 0.91, title3, ha='center', fontsize=ax_size, style='italic')

    loc = [11, 12, 13]
    location = ['\'High\' maintenance location ', '\'Mid\' maintenance location ', '\'Low\' maintenance location ']
    sigma = [r'$\bar x=3.7$ m', r'$\bar x=4$ m', r'$\bar x=4.7$ m']
    for x in loc:
        y = loc.index(x)
        ax = fig.add_subplot(4, 1, y+1, sharey=ax)
        # fig.tight_layout()
        plt.plot(mean[x], linewidth=1, color='black', linestyle='-', label='Mean')
        plt.plot(min[x], linewidth=0.25, color='grey', linestyle='-')
        plt.plot(max[x], linewidth=0.25, color='grey', linestyle='-')
        plt.subplots_adjust(left=0.046, bottom=0.07, right=0.99, top=0.96, hspace=0.5)
        plt.fill_between(np.arange(min.shape[1]), max[x, range(min.shape[1])], min[x, range(min.shape[1])],
                         color='grey', alpha=0.18, label='Range')
        plt.xticks(np.arange(0.5, 460, 7.1), np.arange(0, 2200, 100), fontsize=l_size)
        plt.yticks(np.arange(-20, 30, 10), np.arange(-20, 30, 10), fontsize=l_size)
        plt.title(location[y], fontsize=ax_size, loc='left')
        plt.xlim(-.5, 151)
        plt.ylim(-22, 22)
        plt.axvspan(0, 50.2, color='grey', alpha=0.0, lw=0)
        plt.axhline(y=0, color='black', linewidth=0.6)
        plt.text(1, -18, sigma[y], fontsize=l_size)
        plt.axhspan(-.5, 22, color='green', alpha=0.04, lw=0)
        plt.axhspan(-.5, -22, color='tomato', alpha=0.04, lw=0)
        plt.text(141.1, -18, 'Erosion', alpha=0.85, color='tomato', fontsize=l_size)
        plt.text(138.4, 15.6, 'Deposition', alpha=0.85, color='green', fontsize=l_size)
        # plt.text(0.5, 16, label[y], fontsize=l_size)
    plt.legend(ncol=2, fontsize=l_size, framealpha=0, bbox_to_anchor=(0.6, 0.012), loc=3)
    plt.savefig(save2, dpi=300, bbox_inches='tight')
    print('mean and range of elevation change plotted''\n')
def doHigherThan(n):
    highervalues=[]
    relvalues=[]
    for x in range(diff_mean.shape[0]):
        hv = len(diff_mean[x][np.where(diff_mean[x] < n)])
        rv = hv/151*100
        highervalues.append(hv)
        relvalues.append(rv)
    relvalues = np.round(relvalues, 1)
    print('\n''How many values are smaller than ' + str(n) + ' for each maintenance scenario (100%, 90%,
...)?''\n'
          + str(highervalues))
    print('\n''What is the relative share?''\n' + str(relvalues))

# -READ IN FILES HERE-
# define where profile and initial DEM are located to load in and where outputs should be saved at
in_path = 'U:simulations/analysis/python/profile/profile_old.txt'
start_DEM = 'U:simulations/analysis/python/profile/elevSlide2.txt'
out_path1 = 'U:simulations/analysis/python/profile/DEM/profile_DEM{x}.txt'
out_path2 = 'U:simulations/analysis/python/profile/profile.txt'

# -CALL FUNCTIONS HERE-
## ---------------------- 1 longitudinal profile ----------------------
# mask the two different DEMs (initial and final DEM) with the profile line (created in ArcGIS)
startDEM, endDEM = doProfile_prof(in_path, start_DEM, DEM.shape[0], DEM)

# sort the different maintenance scenarios. calculate mean of all flood scenarios for the different maintenance
scenarios
profile_mean, mi, ma = doMean(endDEM)
del(mi, ma)
# plot the longitudinal profile
# define plot properties
xlabel = "Length [m]"
ylabel = "Elevation [m a.s.l.]"
ax_size = 18
l_size = 15
title = "Longitudinal profile of channel after 100 years of simulation"
save0 = 'U:simulations/analysis/python/profile/longitudinalProfile0perc.png'
save100 = 'U:simulations/analysis/python/profile/longitudinalProfile100perc.png'

doPlot_prof(startDEM, profile_mean, xlabel, ylabel, ax_size, l_size, title, save0, save100)

## ---------------------- 2 DEMdiff profile ----------------------
# calculate elevation difference of DEM between year 100 and 0
DEMdiff = doDEMdiff(DEM.shape[0])  # shape of first dimension = 66, shape of second dimension = 100-1

# mask the DEMdiff file with the profile line (created in ArcGIS)
DEMdiff_thal = doProfile_diff(in_path, DEMdiff.shape[0], DEMdiff)
```

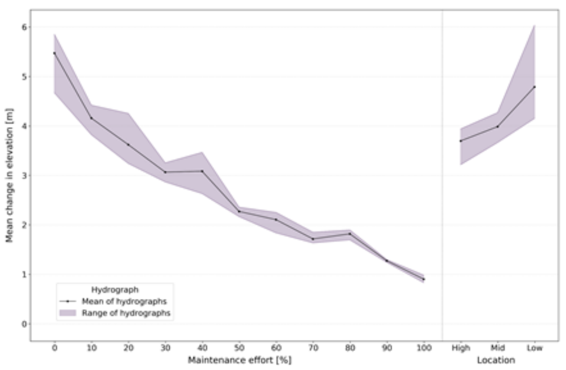
```
# sort the different maintenance scenarios. calculate mean of all flood scenarios for the different maintenance scenarios
diff_mean, diff_min, diff_max = doMean(DEMdiff_thal)

# calculate number of values smaller than x for different maintenance scenarios to get the p-quantile of the dataset
n = 5.5   # threshold value, how many values are smaller than this number?
doHigherThan(n)

# plot the elevation difference
# define plot properties
ylabel = "Change in elevation [m]"
title1 = "Change in channel elevation after 100 years of simulation"
title2 = "Maintenance effort"
title3 = "Maintenance location"
save1 = 'U:simulations/analysis/python/profile/ElevDiff_maint.png'
save2 = 'U:simulations/analysis/python/profile/ElevDiff_loc.png'
doPlot_diff(diff_mean, diff_min, diff_max, xlabel, ylabel, l_size, ax_size, title1, title2, title3, save1, save2)
```

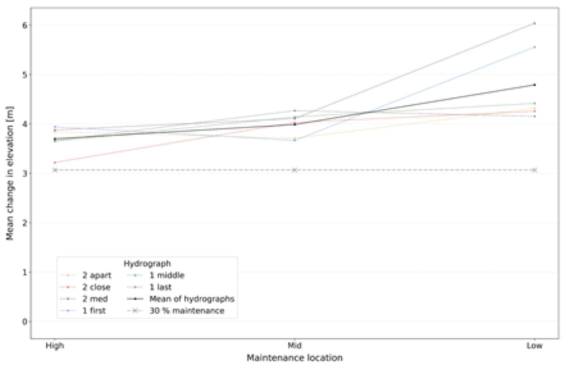
3.3 Tool 3: Mean change in channel elevation over time

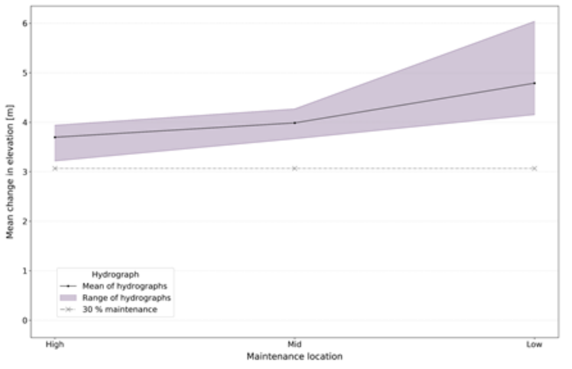
Mean elevation difference: This script calculates the mean elevation difference of the total channel after a certain number of years. You can choose to calculate the total difference after a 100 years of simulation (diff_yrs = [100]) or define the years of difference you want to look at (e.g. always calculate the difference after 20 years, calculate the difference after the flood events, or only after the big flood events). Depending if you chose the first option (difference after 100 years) or the second option (continuous difference during the 100 years), different figures will be plotted. The figures present either the mean channel change after the whole simulation time for the different scenarios or the evolution of the channel change within the 100 years of simulation. If diff_yrs = [100], all arrays exported to ArcGIS files, which include geometric information. Additionally, a function for a unique cholor scheme is developed.

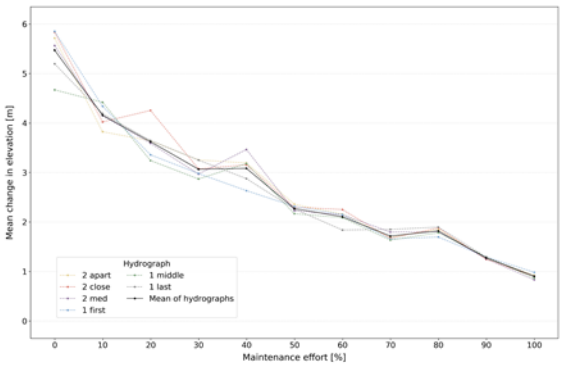
The script produces following figures (for reasons of simplicity not all figures are presented):

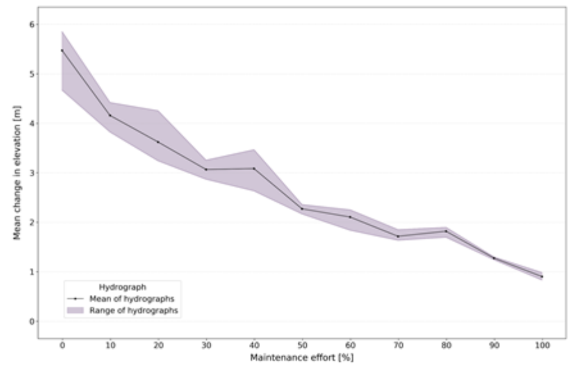


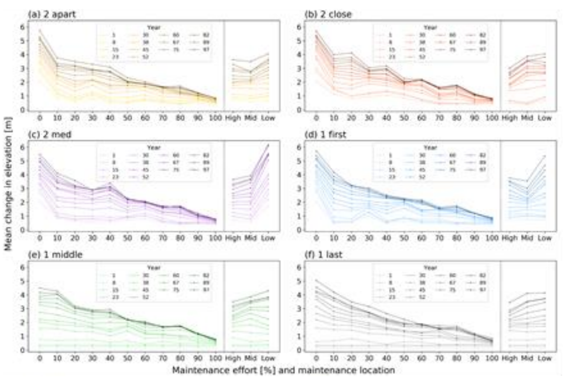


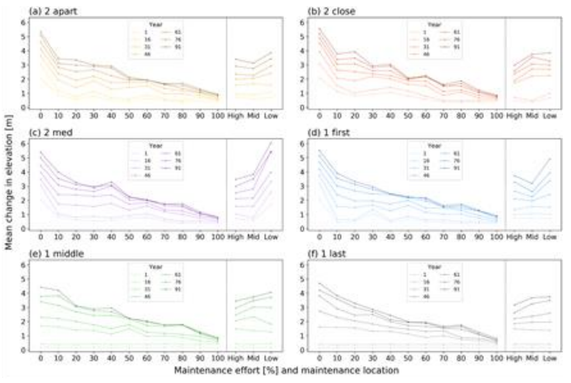

Mean change in elevation [m]
Maintenance location
High
Mid
Low
Hydrograph
2 apart
2 close
2 med
1 first
1 middle
1 last
Mean of hydrographs
30 % maintenance

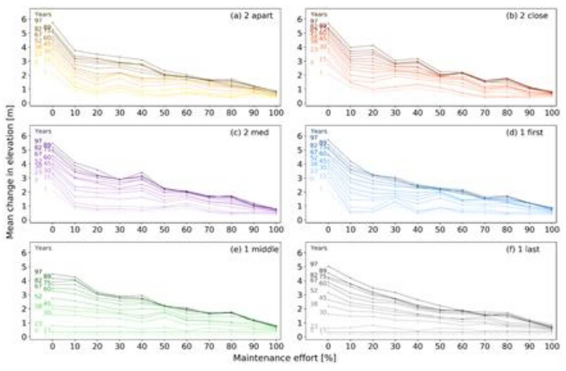

Mean change in elevation [m]
Maintenance location
High
Mid
Low
Hydrograph
Mean of hydrographs
Range of hydrographs
30 % maintenance

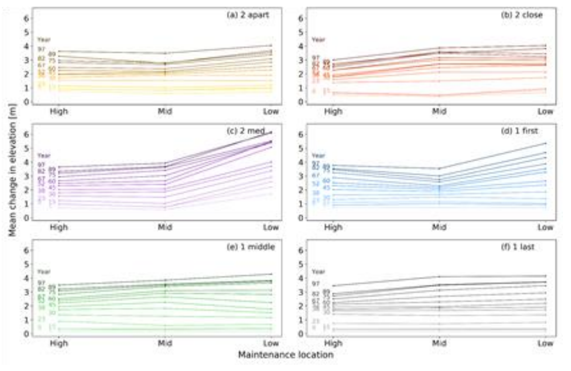

Mean change in elevation [m]
Maintenance effort [%]
Hydrograph
2 apart
2 close
2 med
1 first
1 middle
1 last
Mean of hydrographs

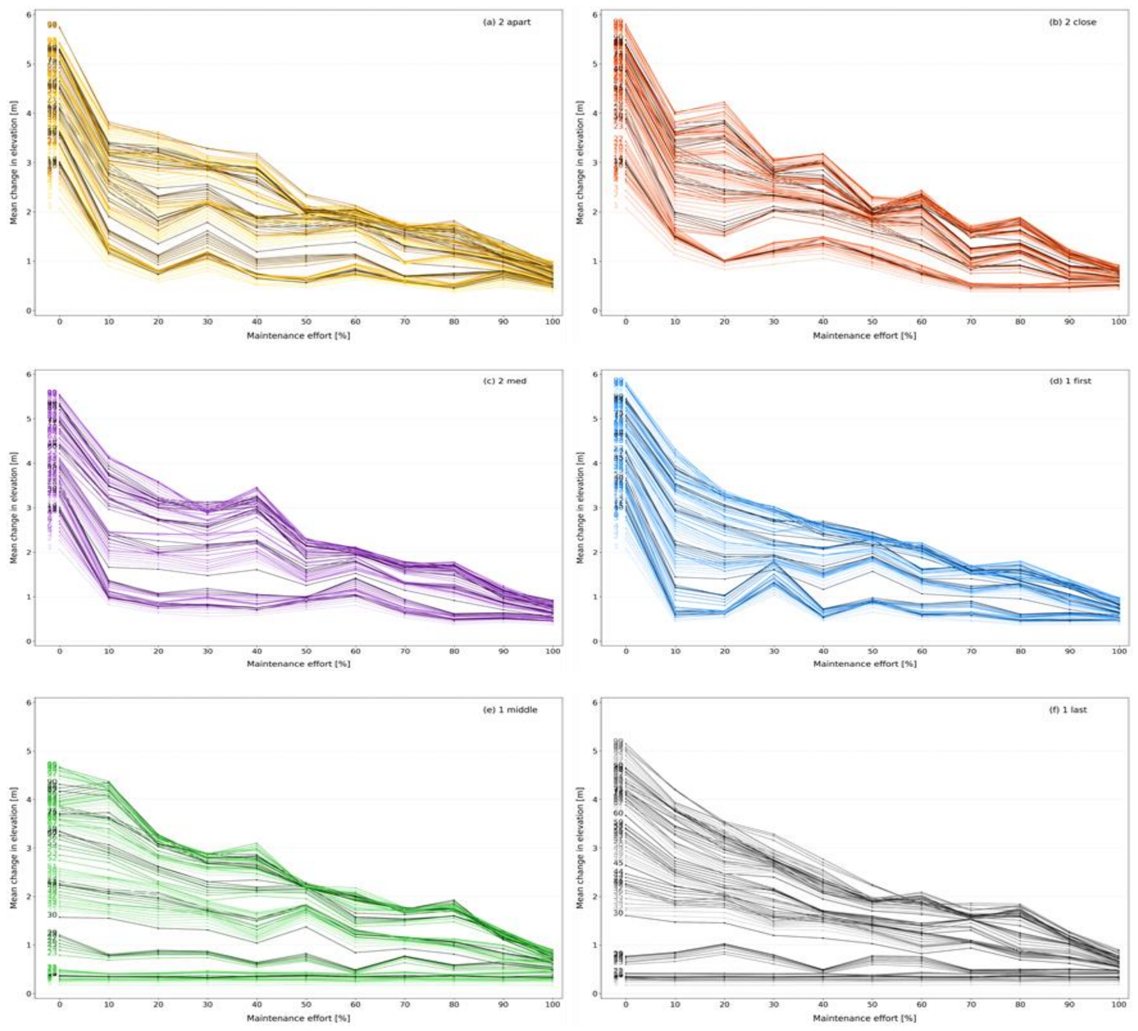

Mean change in elevation [m]
Maintenance effort [%]
Hydrograph
Mean of hydrographs
Range of hydrographs


(a) 2 apart
(b) 2 close
(c) 2 med
(d) 1 first
(e) 1 middle
(f) 1 last
Year
Mean change in elevation [m]
Maintenance effort [%] and maintenance location


(a) 2 apart
(b) 2 close
(c) 2 med
(d) 1 first
(e) 1 middle
(f) 1 last
Year
Mean change in elevation [m]
Maintenance effort [%] and maintenance location


(a) 2 apart
(b) 2 close
(c) 2 med
(d) 1 first
(e) 1 middle
(f) 1 last
Years
Mean change in elevation [m]
Maintenance effort [%]


(a) 2 apart
(b) 2 close
(c) 2 med
(d) 1 first
(e) 1 middle
(f) 1 last
Year
Mean change in elevation [m]
Maintenance location



```
# MEAN ELEVATION DIFFERENCE: This script calculates the mean elevation difference
# of the total channel after a certain number of years. You can choose to calculate
# the total difference after a 100 years of simulation ('diff_yrs = [100]') or de-
# fine the years of difference you want to look at (e.g. always calculate the dif-
# ference after 20 years, calculate the difference after the flood events, or only
# after the big flood events). Depending if you chose the first option (difference
# after 100 years) # or the second option (continuous difference during the 100
# years), different figures will be plotted. The figures present either the mean
# channel change after the whole simulation time for the different scenarios or the
# evolution of the channel change within the 100 years of simulation. If 'diff_yrs # = [100]', all arrays exported to
ArcGIS files, which include geometric infor-
# mation. additionally, a functin for a unique cholor scheme is developed.

# -IMPORT LIBRARIES & VARIABLES HERE-
import numpy as np
import matplotlib.pyplot as plt
from tool2a_openDEMs import DEM

# -DEFINE FUNCTIONS HERE-
def doDEMdiff(scenarios, diffyear):
    '''reates the difference for each cell between predefined years for each scenario'''
    DEMdiff = []
    for scenario in range(scenarios):
        DEMdiff_list = DEM[scenario, diffyear, :, :] - DEM[scenario, 0, :, :]
        DEMdiff.append(DEMdiff_list)
    DEMdiff = np.array(DEMdiff)
    DEMdiffzero = DEMdiff
    DEMdiffnan = []  # tranfer zero values into nan
    for scenario in range(DEM.shape[0]):
        interlist = []
        for diff in range(len(diffyear)):
            DEMdiff0 = np.where(DEMdiff[scenario, diff, :, :] == 0, np.nan, DEMdiff[scenario, diff, :, :])
            interlist.append(DEMdiff0)
        DEMdiffnan.append(interlist)
    DEMdiffnan = abs(np.array(DEMdiffnan))

    print('\n"difference calculations finished"\n')
    return DEMdiffzero, DEMdiffnan
def doProfile (paths, scenarios, diffyear, DEMdiff):
    '''mask all generated arrays with the thalweg array, so only the values that belong to the thalweg are analyzed'''
    # load in thalweg file, created in ArcGIS with flow accumulation, which has the same extent as the DEM
    thalweg = np.genfromtxt(paths, skip_header=6, delimiter=' ')

    thal = []
    for x in range(scenarios):
        interlist = []
        for y in range(len(diffyear)):
            thal_0 = np.where(thalweg == True, DEMdiff0[x, y,:, :], 0)    # mask DEMdiff array to get global mean and
std
            interlist.append(thal_0)
        thal.append(interlist)
    thalzero = np.array(thal)

    thal = []
    for x in range(scenarios):
        interlist = []
        for y in range(len(diffyear)):
            thal_0 = np.where(thalweg == True, DEMdiff[x, y, :, :], np.nan)    # mask DEMdiff array to get global mean
and std
            interlist.append(thal_0)
        thal.append(interlist)
    thalnan = np.array(thal)
    print('thalweg read in and zeros changed to NaN"\n')
    return thalzero, thalnan
def doStatistics(scenarios, diffyear, elev_diff):
    """calculate mean, std of difference of selected years"""
    mean_DEMdiff = []
    for scenario in range(scenarios):  # loop through all scenarios
        interlist = []
        for diff in range(len(diffyear)):
            mean_DEMdiff_m = np.nanmean(
                elev_diff[scenario, diff, :, :])  # build mean of DEMdiff for each cell during 100yrs
            interlist.append(mean_DEMdiff_m)
        mean_DEMdiff.append(interlist)
    mean_DEMdiff = np.array(mean_DEMdiff)

    std_DEMdiff = []
    for scenario in range(scenarios):
        interlist = []
        for diff in range(len(diffyear)):
            std_DEMdiff_s = np.nanstd(
                elev_diff[scenario, diff, :, :])  # build std of DEMdiff for each cell during 100yrs
            interlist.append(std_DEMdiff_s)
        std_DEMdiff.append(interlist)
    std_DEMdiff = np.array(std_DEMdiff)

    def rmse(diff):
        return np.sqrt(np.nanmean((abs(diff))**2))
    rmse_DEMdiff = []
    for scenario in range(scenarios):
        interlist = []
        for diff in range(len(diffyear)):
            rmse_DEMdiff_s = rmse(
                elev_diff[scenario, diff, :, :])  # build std of DEMdiff for each cell during 100yrs
            interlist.append(rmse_DEMdiff_s)
        rmse_DEMdiff.append(interlist)
    rmse_DEMdiff = np.array(rmse_DEMdiff)
    print('statistic calculations finished''\n')
    return mean_DEMdiff, std_DEMdiff, rmse_DEMdiff
def doNewArray(input):
    '''create arrays to original elev array: specific maintenance scn ("perc"), location scn ("loc"), flood scn ("flood")'''
    perc_loc = np.repeat(np.arange(0, 7, 0.5), 6).reshape(input.shape[0], 1)
    new_array = np.append(np.vstack(input), perc_loc, axis=1)  # combine the new created perc_loc
    floods = np.array(14 * ['2apart', '2close', '2med', 'a_first', 'b_middle', 'c_last']).reshape(input.shape[0], 1)
    new_array = np.append(new_array, floods, axis=1)  # append third column to STD array
    return new_array
def doArray(elev_diff):
    '''create new array which is sorted in the right way for analyzing summary statistics of difference'''
    new_array = doNewArray(elev_diff)
    # sort array by following order: 1st by flood scenarios (3rd col), 2nd by maintenance/location scenarios (2nd col)
    a = []
    for x in range(elev_diff.shape[1]):
        b = new_array[:, x]
        a.append(b)
    a = np.array(a)
    b = new_array[:, -2]
    c = new_array[:, -1]

    sorts = []
    for x in range(elev_diff.shape[1]):
        ind = np.lexsort((a[x, :], b, c))                        # create array with the specified order
        sort = np.array([(a[x, :][i], b[i], c[i]) for i in ind])    # apply the "sorting array" to the original array
        sorts.append(sort)
    sorts = np.array(sorts)

    if elev_diff.shape[1]>1:                                    # loop through diff_yrs
        splits = []
        for x in range(elev_diff.shape[1]):
            split = np.array(np.split(sorts[x, :, :], 6))           # split array into different flood scn (6)
            splits.append(split)
        splits = np.array(splits)
    else:                                                   # no need to loop through diff_yrs
        splits = np.array(np.split(sorts[0, :, :], 6))
        splits = np.array(splits)
    # location & maintenance scenarios need to be split in order to plot them separately
    split1 = []
    split2 = []

    if elev_diff.shape[1]>1:                                    # loop through maint scn AND diff_yrs
        for x in range(splits.shape[0]):
            interlist1 = []
            interlist2 = []
            for y in range(splits.shape[1]):
                first = splits[x, y, :11, :]
                last = splits[x, y, -3:, :]
                interlist1.append(first)
                interlist2.append(last)
            split1.append(interlist1)
            split2.append(interlist2)
        split1 = np.array(split1)
        split2 = np.array(split2)
    else:                                                   # only loop through maint scn
```

```
        for x in range(splits.shape[0]):
            first = splits[x, :11, :]
            last = splits[x, -3:, :]
            split1.append(first)
            split2.append(last)
        split1 = np.array(split1)
        split2 = np.array(split2)
    print('array created''\n')
    return split1, split2
def doMinMax(elev_diff):
    '''calculates the min and the max value for each flood scenario. depending on how many diffyears are analyzed
    (if 1 or more), the calculation method is adapted'''
    new_array = doNewArray(elev_diff)

    if elev_diff.shape[1]>1:
        # sort array by following order: 1st by flood scenarios (3rd col), 2nd by maintenance/location scenarios (2nd
col)
        sort_list=[]
        for x in range(elev_diff.shape[1]):
            a = new_array[:, x]
            b = new_array[:, -1]
            c = new_array[:, -2]
            ind = np.lexsort((b, c))  # create array with the specified order
            sort = np.array([(a[i], b[i], c[i]) for i in ind])  # apply the "sorting array" to the original array
            sort_list.append(sort[:, 0])
        sort_list.append((sort[:, -1]))
        sort_list.append(sort[:, -2])
        sort_list = np.array(sort_list)

        splits = np.array(np.hsplit(sort_list[:, :], 14))
        splits = np.array(splits)

        min_f = []
        for x in range(splits.shape[0]):  # loop through all scenarios
            interlist=[]
            for y in range(splits.shape[1]-2):
                min = np.min((splits[x, y, :]).astype('float'))   # min of all flood scn
                interlist.append(min)
            min_f.append(interlist)
        min_flood = np.array(min_f)

        max_f = []
        for x in range(splits.shape[0]):  # loop through all scenarios
            interlist=[]
            for y in range(splits.shape[1]-2):
                min = np.max((splits[x, y, :]).astype('float'))   # min of all flood scn
                interlist.append(min)
            max_f.append(interlist)
        max_flood = np.array(max_f)

        perc_loc2 = np.vstack(np.append((0, .5, 1, 1.5, 2, 2.5, 3, 3.5, 4, 4.5, 5), np.array([5.5, 6, 6.5])))

        min = np.append(min_flood[:], perc_loc2, axis=1)
        min1 = np.array([(min[x, :]) for x in range(11)])
        min2 = np.array([(min[x, :]) for x in range(11, 14)])

        max = np.append(max_flood[:], perc_loc2, axis=1)
        max1 = np.array([(max[x, :]) for x in range(11)])
        max2 = np.array([(max[x, :]) for x in range(11, 14)])
    else:
        # sort array by following order: 1st by flood scenarios (3rd col), 2nd by maintenance/location scenarios (2nd
col)
        a = new_array[:, 0]
        b = new_array[:, -1]
        c = new_array[:, -2]
        ind = np.lexsort((a, b, c))  # create array with the specified order
        sort = np.array([(a[i], b[i], c[i]) for i in ind])  # apply the "sorting array" to the original array

        splits = np.array(np.split(sort[:, :], 14))
        splits = np.array(splits)

        min_f = []
        for x in range(splits.shape[0]):  # loop through all scenarios
            min = np.min((splits[x, :, 0]).astype('float'))   # min of all flood scn
            min_f.append(min)
        min_flood = np.array(min_f)

        max_f = []
        for x in range(splits.shape[0]):  # loop through all scenarios
            max = np.max((splits[x, :, 0]).astype('float'))   # max of all flood scn
            max_f.append(max)
        max_flood = np.array(max_f)

        perc_loc2 = np.append((0, .5, 1, 1.5, 2, 2.5, 3, 3.5, 4, 4.5, 5), np.array([5.5, 6, 6.5]))

        min = np.append(min_flood[:], perc_loc2).reshape(2,min_flood.shape[0])
        min1 = np.array([(min[:,x]) for x in range(11)])
        min2 = np.array([(min[:,x]) for x in range(11, 14)])

        max = np.append(max_flood[:], perc_loc2).reshape(2,max_flood.shape[0])
        max1 = np.array([(max[:,x]) for x in range(11)])
        max2 = np.array([(max[:,x]) for x in range(11, 14)])
    print('min and max calculated''\n')
    return min1, min2, max1, max2
def doFloodmean(elev_diff):
    '''calculates the flood mean of all flood scenarios. depending on how many diffyears are analyzed (if 1 or more),
    the calculation method is adapted'''
    new_array = doNewArray(elev_diff)

    if elev_diff.shape[1]>1:
        # sort array by following order: 1st by flood scn (3rd col), 2nd by maintenance/location scn (2nd col)
        sort_list=[]
        for x in range(elev_diff.shape[1]):
            a = new_array[:, x]
            b = new_array[:, -1]
            c = new_array[:, -2]
            ind = np.lexsort((b, c))  # create array with the specified order
            sort = np.array([(a[i], b[i], c[i]) for i in ind])  # apply the "sorting array" to the original array
            sort_list.append(sort[:, 0])
        sort_list.append((sort[:, -1]))
        sort_list.append(sort[:, -2])
        sort_list = np.array(sort_list)

        splits = np.array(np.hsplit(sort_list[:, :], 14))
        splits = np.array(splits)

        mean_flood = []
        for x in range(splits.shape[0]):  # loop through all scenarios
            interlist=[]
            for y in range(splits.shape[1]-2):
                mean_m = np.mean((splits[x, y, :]).astype('float'))  # build mean of DEMdiff for each cell during 100yrs
                interlist.append(mean_m)
            mean_flood.append(interlist)
        mean_flood = np.array(mean_flood)

        perc_loc2 = np.vstack(np.append((0, .5, 1, 1.5, 2, 2.5, 3, 3.5, 4, 4.5, 5), np.array([5.5, 6, 6.5])))
        mean_flood2 = np.append(mean_flood, perc_loc2, axis=1)

        split1 = np.array([(mean_flood2[x, :]) for x in range(11)])
        split2 = np.array([(mean_flood2[x, :]) for x in range(11, 14)])
    else:
        # sort array by following order: 1st by maintenance scenarios (3rd col), 2nd by flood scenarios (2nd col)
        a = new_array[:, 0]
        b = new_array[:, -1]
        c = new_array[:, -2]
        ind = np.lexsort((a, b, c))  # create array with the specified order
        sort = np.array([(a[i], b[i], c[i]) for i in ind])  # apply the "sorting array" to the original array

        splits = np.array(np.split(sort[:, :], 14))
        splits = np.array(splits)

        # calculate mean of flood scn for different maintenance scn
        mean_flood = []
        for x in range(splits.shape[0]):  # loop through all scenarios
            mean_m = np.mean((splits[x, :, 0]).astype('float'))  # build mean of DEMdiff for each cell during 100yrs
            mean_flood.append(mean_m)
        mean_flood = np.array(mean_flood)

        perc_loc2 = np.append((0, .5, 1, 1.5, 2, 2.5, 3, 3.5, 4, 4.5, 5), np.array([5.5, 6, 6.5]))
        mean_flood2 = np.append((mean_flood), perc_loc2).reshape(2,mean_flood.shape[0])

        split1 = np.array([(mean_flood2[:, x]) for x in range(11)])
        split2 = np.array([(mean_flood2[:, x]) for x in range(11, 14)])

    print('mean_flood calculated''\n')
    return split1, split2
def doColors(diffyear):
    '''create color palette to plot the different diff_yrs scenarios. depending on the number of diff_years,
    the palette will be extended'''
    if diffyear <= 8:
        # palette = 'gold','orangered','darkorchid','dodgerblue','limegreen','dimgrey'
        palette = np.array([['#ffe766', '#ffb499', '#e0c1ef', '#bbddff', '#adebad', '#c3c3c3'],
                            ['#ffdf32', '#ff8f66', '#cc98e5', '#8ec7ff', '#84e184', '#a5a5a5'],
                            ['#ffae19', '#ff6a32', '#b76fdb', '#61b1ff', '#5ad75a', '#878787'],
                            ['#e59400', '#ff4500', '#a346d1', '#349bff', '#32cd32', '#696969'],
                            ['#b27300', '#e53e00', '#9932cc', '#1e90ff', '#2db82d', '#5e5e5e'],
                            ['#895900', '#b23000', '#7a28a3', '#1873cc', '#238f23', '#494949'],
                            ['#6b4500', '#7f2200', '#5b1e7a', '#125699', '#196619', '#343434'],
                            ['#4c3100', '#4c1400', '#3d1451', '#0c3966', '#0f3d0f', '#1f1f1f']])
    elif 9 < diffyear <= 15:
        palette = np.array([['#ffe766', '#ffb499', '#e0c1ef', '#bbddff', '#adebad', '#c3c3c3'],
                            ['#ffe34c', '#ffa27f', '#d6adea', '#a5d2ff', '#98e698', '#b4b4b4'],
                            ['#ffdf32', '#ff8f66', '#cc98e5', '#8ec7ff', '#84e184', '#a5a5a5'],
                            ['#ffd700', '#ff7c4c', '#c184e0', '#78bcff', '#6fdc6f', '#969696'],
                            ['#ffae19', '#ff6a32', '#b76fdb', '#61b1ff', '#5ad75a', '#878787'],
                            ['#ffa500', '#ff5719', '#ad5ad6', '#4aa6ff', '#46d246', '#787878'],
                            ['#e59400', '#ff4500', '#a346d1', '#349bff', '#32cd32', '#696969'],
                            ['#b27300', '#e53e00', '#9932cc', '#1e90ff', '#2db82d', '#5e5e5e'],
                            ['#996300', '#cc3700', '#892db7', '#1b81e5', '#28a428', '#545454'],
                            ['#895900', '#b23000', '#7a28a3', '#1873cc', '#238f23', '#494949'],
                            ['#7a4f00', '#992900', '#6b238e', '#1564b2', '#1e7b1e', '#3f3f3f'],
                            ['#6b4500', '#7f2200', '#5b1e7a', '#125699', '#196619', '#343434'],
                            ['#5b3b00', '#661b00', '#4c1966', '#0f487f', '#145214', '#2a2a2a'],
                            ['#4c3100', '#4c1400', '#3d1451', '#0c3966', '#0f3d0f', '#1f1f1f'],
                            ['#3d2700', '#330d00', '#2d0f3d', '#092b4c', '#0a290a', '#0a0a0a']])
    else:
        print('the number of lines exceeds the number of colors. the color palette will be replicated as many
times as '
              'necessary.')
        if diffyear % 15 == 0:
            multi = int(diffyear / 15)
        else:
            multi = np.round(diffyear / 15, 0).astype(int) + 1
        palette = np.array(multi * [['#ffe766', '#ffb499', '#e0c1ef', '#bbddff', '#adebad', '#c3c3c3'],
                                    ['#ffe34c', '#ffa27f', '#d6adea', '#a5d2ff', '#98e698', '#b4b4b4'],
                                    ['#ffdf32', '#ff8f66', '#cc98e5', '#8ec7ff', '#84e184', '#a5a5a5'],
                                    ['#ffd700', '#ff7c4c', '#c184e0', '#78bcff', '#6fdc6f', '#969696'],
                                    ['#ffae19', '#ff6a32', '#b76fdb', '#61b1ff', '#5ad75a', '#878787'],
                                    ['#ffa500', '#ff5719', '#ad5ad6', '#4aa6ff', '#46d246', '#787878'],
                                    ['#e59400', '#ff4500', '#a346d1', '#349bff', '#32cd32', '#696969'],
                                    ['#b27300', '#e53e00', '#9932cc', '#1e90ff', '#2db82d', '#5e5e5e'],
                                    ['#996300', '#cc3700', '#892db7', '#1b81e5', '#28a428', '#545454'],
                                    ['#895900', '#b23000', '#7a28a3', '#1873cc', '#238f23', '#494949'],
                                    ['#7a4f00', '#992900', '#6b238e', '#1564b2', '#1e7b1e', '#3f3f3f'],
                                    ['#6b4500', '#7f2200', '#5b1e7a', '#125699', '#196619', '#343434'],
                                    ['#5b3b00', '#661b00', '#4c1966', '#0f487f', '#145214', '#2a2a2a'],
                                    ['#4c3100', '#4c1400', '#3d1451', '#0c3966', '#0f3d0f', '#1f1f1f'],
                                    ['#3d2700', '#330d00', '#2d0f3d', '#092b4c', '#0a290a', '#0a0a0a']])
    return palette
def doPlot(diff1, diff2, flood1, flood2, min1, min2, max1, max2, diffyear, diff):
    '''create two different plots, first one for all flood scn seperately, second one for the mean of all flood scn and
    its range. the plots includes the location scenarios'''
    floods = ['(a) 2 apart', '(b) 2 close', '(c) 2 med', '(d) 1 first', '(e) 1 middle', '(f) 1 last']
    palette3 = np.array([diffyear*['#f9f0a1']])
    palette2 = np.array(['#e3ce8d', '#db786c', '#8e729d', '#7ba6d0', '#7ba47b', '#8d8d8d'])
    palette = doColors(diffyear)
    if diffyear>1:
        # all flood scn
        fig = plt.figure(figsize=(19, 12))
        # fig.suptitle(title, fontsize=24, fontweight=1, color='black').set_position([.5, 0.94])
        ax = plt.axes([0, 0, 1, 1], frameon=False)
        ax.axes.get_xaxis().set_visible(False)
        ax.axes.get_yaxis().set_visible(False)
        fig.text(0.5, 0.06, xlabel_subplot, ha='center', fontsize=ax_size)
        fig.text(0.09, 0.5, ylabel, va='center', rotation='vertical', fontsize=ax_size)
        for x in range(len(floods)):
            ax = fig.add_subplot(3, 2, (x + 1), sharey=ax)
            plt.title(floods[x], fontsize=ax_size, color='black', loc='left')
            plt.subplots_adjust(wspace=0.1, hspace=0.32)
            for y in range(diffyear):
                ax.plot(diff1[y, x, :, 1], diff1[y, x, :, 0].astype(float), color=palette[y, x],
                        marker='.', markersize=2, linestyle='-', linewidth=.5, label=diff[y])
                ax.plot(diff2[y, x, :, 1], diff2[y, x, :, 0].astype(float), color=palette[y, x],
                        marker='.', markersize=2, linestyle='-', linewidth=.5, label='_nolegend_')
                legend = plt.legend(loc='upper center', ncol=np.round(diffyear/4, 0).astype(int),
                                    fontsize=10, title=legend_y)
                plt.setp(legend.get_title(), fontsize=(l_size-4))
                plt.xticks([r + 0.005 for r in range(0, 14)],
                           [0, 10, 20, 30, 40, 50, 60, 70, 80, 90, 100, 'High', 'Mid', 'Low'],
                           fontsize=l_size-2)
                plt.yticks(fontsize=l_size-2)
                plt.axvline(x=10.5, color='black', linestyle='--', linewidth=0.5, alpha=0.6)
        plt.savefig(save1, dpi=300, bbox_inches='tight')

    else:
        # all flood scn
        floods = ['2 apart', '2 close', '2 med', '1 first', '1 middle', '1 last']
        fig = plt.figure(figsize=(19, 12))
        # fig.suptitle(title, fontsize=24, fontweight=1, color='black').set_position([.5, 0.94])
        ax = fig.add_subplot(1, 1, 1)
        for x in range(len(floods)):
            ax.plot(diff1[x, :, 1], diff1[x, :, 0].astype(float), color=palette2[x],
                marker='.', linestyle='--', linewidth=1, label=floods[x])
            ax.plot(diff2[x, :, 1], diff2[x, :, 0].astype(float), color=palette2[x],
                marker='.', linestyle='--', linewidth=1, label='_nolegend_')
        plt.plot(flood1[:, 1], flood1[:, 0].astype(float), color='black',
                marker='.', linestyle='-', linewidth=1, label=label_mean)
        plt.plot(flood2[:, 1], flood2[:, 0].astype(float), color='black',
                marker='.', linestyle='-', linewidth=1, label='_nolegend_')
        legend = plt.legend(ncol=2, fontsize=l_size, title=legend_h, bbox_to_anchor=(0.042, 0.05), loc=3)
        plt.setp(legend.get_title(), fontsize=l_size)
        plt.xticks([r + 0.005 for r in range(0, 14)], [0, 10, 20, 30, 40, 50, 60, 70, 80, 90, 100, 'High', 'Mid', 'Low'],
                   fontsize=l_size)
        plt.yticks(fontsize=l_size)
        plt.ylim(lim)
        plt.xlabel(xlabel, fontsize=ax_size, labelpad=8)
        plt.ylabel(ylabel, fontsize=ax_size, labelpad=15)
        ax.yaxis.grid(linestyle='--', alpha=0.3)
        plt.axvline(x=10.5, color='black', linestyle='--', linewidth=0.4)
        plt.savefig(save1, dpi=300, bbox_inches='tight')

        # mean + range flood scn
        fig = plt.figure(figsize=(19, 12))
        # fig.suptitle(title, fontsize=24, fontweight=1, color='black').set_position([.5, 0.94])
        ax = fig.add_subplot(1, 1, 1)
        ax.plot(min1[:, 1], min1[:, 0].astype(float), color='#8e729d',
                linestyle='-', linewidth=0.7)
        ax.plot(min2[:, 1], min2[:, 0].astype(float), color='#8e729d',
                linestyle='-', linewidth=0.7, label='_nolegend_')
        ax.plot(max1[:, 1], max1[:, 0].astype(float), color='#8e729d',
                linestyle='-', linewidth=0.7, label='_nolegend_')
        ax.plot(max2[:, 1], max2[:, 0].astype(float), color='#8e729d',
                linestyle='-', linewidth=0.7, label='_nolegend_')
        ax.plot(flood1[:, 1], flood1[:, 0].astype(float), color='black',
                linestyle='-', marker='.', linewidth=1, label=label_mean)
        ax.plot(flood2[:, 1], flood2[:, 0].astype(float), color='black',
                linestyle='-', marker='.', linewidth=1, label='_nolegend_')
        plt.fill_between(min1[:, 1], max1[:, 0], min1[:, 0], color='#8e729d', alpha=0.4, label=label_range)
        plt.fill_between(min2[:, 1], max2[:, 0], min2[:, 0], color='#8e729d', alpha=0.4)
        legend = plt.legend(ncol=1, fontsize=l_size, title=legend_h, bbox_to_anchor=(0.042, 0.05), loc=3)
        plt.setp(legend.get_title(), fontsize=l_size)
        plt.xticks([r + 0.005 for r in np.arange(0, 7, 0.5)], [0, 10, 20, 30, 40, 50, 60, 70, 80, 90, 100,
                                                                'High', 'Mid', 'Low'], fontsize=l_size)
        plt.ylim(lim)
        plt.yticks(fontsize=l_size)
        plt.ylabel(ylabel, fontsize=ax_size, labelpad=15)
        ax.yaxis.grid(linestyle='--', alpha=0.3)
        plt.axvline(x=5.25, color='black', linestyle='--', linewidth=0.4)
        plt.xlabel(xlabel, fontsize=ax_size, labelpad=8)
        plt.savefig(save2, dpi=300, bbox_inches='tight')
        print('plots with maintenance and location scenario''\n')
    return
def doPlot_maint(diff1, flood1, min1, max1, diffyear, diff):
    '''create two different plots, first one for all flood scn seperately, second one for the mean of all flood scn and
    its range. the plots only show the maintenance effort scenarios'''
    palette2 = np.array(['#e3ce8d', '#db786c', '#8e729d', '#7ba6d0', '#7ba47b', '#8d8d8d'])
    palette = doColors(diffyear)
    floods = ['(a) 2 apart', '(b) 2 close', '(c) 2 med', '(d) 1 first', '(e) 1 middle', '(f) 1 last']

    if diffyear>1:
        # all flood scn
        # subplot
        fig = plt.figure(figsize=(19, 12))
        # fig.suptitle(title, fontsize=24, fontweight=1, color='black').set_position([.5, 0.94])
        palette = doColors(diffyear)
        ax = plt.axes([0, 0, 1, 1], frameon=False)
        ax.axes.get_xaxis().set_visible(False)
        ax.axes.get_yaxis().set_visible(False)
        fig.text(0.5, 0.06, xlabel_subplot, ha='center', fontsize=ax_size)
        fig.text(0.09, 0.5, ylabel, va='center', rotation='vertical', fontsize=ax_size)
        for x in range(len(floods)):
            ax = fig.add_subplot(3, 2, (x + 1), sharey=ax)
            ax.text(0.8, .9, floods[x], fontsize=l_size, color='black', transform=ax.transAxes)
            ax.text(0.02, .92, 'Year', fontsize=l_size-4, color=palette[diffyear-1, x], transform=ax.transAxes)
            plt.subplots_adjust(wspace=0.1, hspace=0.2)
            for y in range(diffyear):
                ax.plot(diff1[y, x, :, 1], diff1[y, x, :, 0].astype(float), color=palette[y, x],
                        marker='.', markersize=2, linestyle='-', linewidth=0.5)
                if y % 2 == 0:
                    ax.text(-.4, diff1[y, x, 0, 0].astype(float), diff_yrs[y],
                            alpha=1, color=palette[y, x], fontsize=l_size-4)
                else:
                    ax.text(-0.8, diff1[y, x, 0, 0].astype(float), diff_yrs[y],
                            alpha=1, color=palette[y, x], fontsize=l_size-4)
                plt.xticks([r + 0.005 for r in range(0, 11)], [0, 10, 20, 30, 40, 50, 60, 70, 80, 90, 100], fontsize=l_size)
                plt.yticks(fontsize=l_size)
                plt.xlim(-1, 10.2)
                plt.ylim(-0.2, 6.75)
        plt.savefig(save1, dpi=400, bbox_inches='tight')

        # individual plots
        for x in range(len(floods)):
            floods = ['(a) 2 apart', '(b) 2 close', '(c) 2 med', '(d) 1 first', '(e) 1 middle', '(f) 1 last']
            fig = plt.figure(figsize=(19, 12))
            palette = doColors(diffyear)

            ax = fig.add_subplot(1, 1, 1)
            ax.text(0.85, 0.95, floods[x], fontsize=ax_size, color='black', transform=ax.transAxes)

            for y in range(diffyear):
                ax.plot(diff1[y, x, :, 1], diff1[y, x, :, 0].astype(float), color=palette[y, x],
                        marker='.', markersize=4, linestyle='-', linewidth=0.9)
                ax.text(-.25, diff1[y, x, 0, 0].astype(float), diff_yrs[y], alpha=1, color=palette[y, x], fontsize=l_size)
            ax.yaxis.grid(linestyle='--', alpha=0.3)
            plt.ylabel(ylabel, fontsize=ax_size, labelpad=15)
            plt.xlabel(xlabel_subplot, fontsize=ax_size, labelpad=(l_size-2))
            plt.xticks([r + 0.005 for r in range(0, 11)], [0, 10, 20, 30, 40, 50, 60, 70, 80, 90, 100], fontsize=l_size)
            plt.yticks(fontsize=l_size)
            plt.ylim(-0.1, 6.1)
            plt.xlim(-.5, 10.2)
            floods = ['2 apart', '2 close', '2 med', '1 first', '1 middle', '1 last']
            plt.savefig('U:simulations/analysis/python/channel                              change/Multi-
    Indplot'+str(typ)+'_'+floods[x]+'_maint.png',
                        dpi=300, bbox_inches='tight')

    else:
        # all flood scn
        floods = ['2 apart', '2 close', '2 med', '1 first', '1 middle', '1 last']
        fig = plt.figure(figsize=(19, 12))
        # fig.suptitle(title, fontsize=24, fontweight=1, color='black').set_position([.5, 0.94])
        ax = fig.add_subplot(1, 1, 1)
        for x in range(len(floods)):
            ax.plot(diff1[x, :, 1], diff1[x, :, 0].astype(float), color=palette2[x],
                    marker='.', linestyle='--', linewidth=1, label=floods[x])
        plt.plot(flood1[:,1], flood1[:,0].astype(float), color='black',
                   marker='.', linestyle='-', linewidth=1, label=label_mean)
        legend = plt.legend(ncol=2, fontsize=l_size, title=legend_h, bbox_to_anchor=(0.042, 0.05), loc=3)
        plt.setp(legend.get_title(), fontsize=l_size)
```

```
        ax.yaxis.grid(linestyle='--', alpha=0.3)
        plt.xticks([r + 0.005 for r in range(0,11)], [0, 10, 20, 30, 40, 50, 60, 70, 80, 90, 100], fontsize=l_size)
        plt.yticks(fontsize=l_size)
        plt.ylim(lim)
        plt.xlabel(xlabel, fontsize=ax_size, labelpad=8)
        plt.ylabel(ylabel, fontsize=ax_size, labelpad=15)
        ax.yaxis.grid(linestyle='--', alpha=0.3)
        plt.savefig(save1, dpi=300, bbox_inches='tight')

        # mean + range flood scn
        fig = plt.figure(figsize=(19, 12))
        # fig.suptitle(title, fontsize=24, fontweight=1,
        #              color='black').set_position([.5, 0.94])
        ax = fig.add_subplot(1, 1, 1)
        ax.plot(min1[:, 1], min1[:, 0].astype(float), color='#8e729d',
                linestyle='-', linewidth=0.7)
        ax.plot(max1[:, 1], max1[:, 0].astype(float), color='#8e729d',
                linestyle='-', linewidth=0.7, label='_nolegend_')
        ax.plot(flood1[:, 1], flood1[:, 0].astype(float), color='black',
                linestyle='-', marker='.', linewidth=1, label=label_mean)
        plt.fill_between(min1[:, 1], max1[:, 0], min1[:, 0], color='#8e729d', alpha=0.4, label=label_range)
        legend = plt.legend(ncol=1, fontsize=l_size, title=legend_h, bbox_to_anchor=(0.042, 0.05), loc=3)
        plt.setp(legend.get_title(), fontsize=l_size)
        ax.yaxis.grid(linestyle='--', alpha=0.3)
        plt.xticks([r + 0.005 for r in np.arange(0, 5.5, 0.5)], [0, 10, 20, 30, 40, 50, 60, 70, 80, 90, 100], fontsize=l_size)
        plt.yticks(fontsize=l_size)
        plt.ylim(lim)
        plt.ylabel(ylabel, fontsize=ax_size, labelpad=15)
        plt.xlabel(xlabel, fontsize=ax_size, labelpad=8)
        ax.yaxis.grid(linestyle='--', alpha=0.3)
        plt.savefig(save2, dpi=300, bbox_inches='tight')
        print('plots with maintenance scenario''\n')
    return
def doPlot_loc(diff1, diff2, flood1, flood2, min2, max2, diffyear, diff):
    '''create two different plots, first one for all flood scn seperately, second one for the mean of all flood scn and
    its range. the plots only show the maintenance location scenarios'''
    palette3 = np.array([diffyear*['#f9f0a1']])
    palette2 = np.array(['#e3ce8d', '#db786c', '#8e729d', '#7ba6d0', '#7ba47b', '#8d8d8d'])
    palette = doColors(diffyear)

    if diffyear>1:
        # all flood scn
        # subplot
        floods = ['(a) 2 apart', '(b) 2 close', '(c) 2 med', '(d) 1 first', '(e) 1 middle', '(f) 1 last']
        fig = plt.figure(figsize=(19, 12))
        # fig.suptitle(title, fontsize=24, fontweight=1, color='black').set_position([.5, 0.95])
        ax = plt.axes([0, 0, 1, 1], frameon=False)
        ax.axes.get_xaxis().set_visible(False)
        ax.axes.get_yaxis().set_visible(False)
        fig.text(0.5, 0.06, xlabel_subplot, ha='center', fontsize=ax_size)
        fig.text(0.09, 0.5, ylabel, va='center', rotation='vertical', fontsize=ax_size)
        for x in range(len(floods)):
            ax = fig.add_subplot(3, 2, (x + 1), sharey=ax)
            ax.text(0.78, .9, floods[x], fontsize=l_size, color='black', transform=ax.transAxes)
            ax.text(0.02, .65, 'Year', fontsize=l_size-4, color=palette[diffyear-1, x], transform=ax.transAxes)
            plt.subplots_adjust(wspace=0.1, hspace=0.2)
            for y in range(diffyear):
                ax.plot(2.38, diff1[y, x, 3, 0].astype(float), color='grey', marker='x', markersize=5.5,
                        label=label_30 if y == 0 else '')
                ax.plot(diff2[y, x, :, 1], diff2[y, x, :, 0].astype(float), color=palette[y, x],
                        marker='.', markersize=3.5, linestyle='-', linewidth=0.9)
                # ax.text(2.19, 3.8, '30% mainte-''\n''nance effort' if y == 0 else '', fontsize=l_size-4, color='grey')
                if y % 2 == 0:
```

```
                ax.text(-.1, diff2[y, x, 0, 0].astype(float), diff_yrs[y],
                        alpha=1, color=palette[y, x], fontsize=l_size-4)
            else:
                ax.text(-.2, diff2[y, x, 0, 0].astype(float), diff_yrs[y],
                        alpha=1, color=palette[y, x], fontsize=l_size-4)
        plt.xticks(np.arange(0, 3, 1), ['High', 'Mid', 'Low'], fontsize=l_size)
        plt.yticks(fontsize=l_size)
        plt.xlim(-0.25, 2.1)
        plt.ylim(-0.2, 6.75)
    plt.savefig(save1, dpi=400, bbox_inches='tight')

    # individual plots
    for x in range(len(floods)):
        floods = ['(a) 2 apart', '(b) 2 close', '(c) 2 med', '(d) 1 first', '(e) 1 middle', '(f) 1 last']
        fig = plt.figure(figsize=(19, 12))
        palette = doColors(diffyear)
        ax = fig.add_subplot(1, 1, 1)
        ax.text(0.02, 0.95, floods[x], fontsize=ax_size, color='black', transform=ax.transAxes)
        for y in range(diffyear):
            ax.plot(2.4, diff1[y, x, 3, 0].astype(float), color='grey', marker='x', markersize=8,)
            ax.plot(diff2[y, x, :, 1], diff2[y, x, :, 0].astype(float), color=palette[y, x],
                    marker='.', markersize=4, linestyle='-', linewidth=0.9)
            if y % 2 == 0:
                ax.text(2.03, diff2[y, x, 2, 0].astype(float), diff_yrs[y],
                        alpha=1, color=palette[y, x], fontsize=(l_size-2))
            else:
                ax.text(2.08, diff2[y, x, 2, 0].astype(float), diff_yrs[y],
                        alpha=1, color=palette[y, x], fontsize=(l_size-2))
        ax.text(2.3, 3.8, '30% mainte-"\n"nance effort', fontsize=l_size, color='grey')
        # ax.text(1.7, 6, 'after ... years', fontsize=l_size, color=palette[y-2, x])

        ax.yaxis.grid(linestyle='--', alpha=0.3)
        plt.ylabel(ylabel, fontsize=ax_size, labelpad=15)
        plt.xlabel(xlabel_subplot, fontsize=ax_size, labelpad=l_size)
        plt.xticks(np.arange(0, 3, 1), ['High', 'Mid', 'Low'], fontsize=l_size)
        plt.yticks(fontsize=l_size)
        plt.xlim(-.1, 2.6)
        plt.ylim(-0.05, 6.2)
        floods = ['2 apart', '2 close', '2 med', '1 first', '1 middle', '1 last']
        plt.savefig('U:simulations/analysis/python/channel change/Multi-
Indplot'+str(typ)+'_'+floods[x]+'_loc.png',
                    dpi=300, bbox_inches='tight')

  else:
    # all flood scn
    floods = ['2 apart', '2 close', '2 med', '1 first', '1 middle', '1 last']
    fig = plt.figure(figsize=(19, 12))
    # fig.suptitle(title, fontsize=24, fontweight=1, color='black').set_position([.5, 0.94])
    ax = fig.add_subplot(1, 1, 1)

    for x in range(len(floods)):
        ax.plot(diff2[x, :, 1], diff2[x, :, 0].astype(float), color=palette2[x],
                marker='.', linestyle='--', linewidth=1, label=floods[x])
    plt.plot(flood2[:, 1], flood2[:, 0].astype(float), color='black',
            marker='.', linestyle='-', linewidth=1, label=label_mean)
    plt.plot(2, flood1[3, 0].astype(float), color='grey', marker='x', markersize=9, linestyle='-', linewidth=1)
    plt.plot(1, flood1[3, 0].astype(float), color='grey', marker='x', markersize=9, linestyle='-', linewidth=1)
    plt.plot(0, flood1[3, 0].astype(float), color='grey',
             marker='x', markersize=9, linestyle='-.', linewidth=1, label=label_30)
    legend = plt.legend(ncol=2, fontsize=l_size, title=legend_h, bbox_to_anchor=(0.042, 0.05), loc=3)
    plt.setp(legend.get_title(), fontsize=l_size)
    ax.yaxis.grid(linestyle='--', alpha=0.3)
    plt.xticks(np.arange(0, 3, 1), ['High', 'Mid', 'Low'], fontsize=l_size)
```

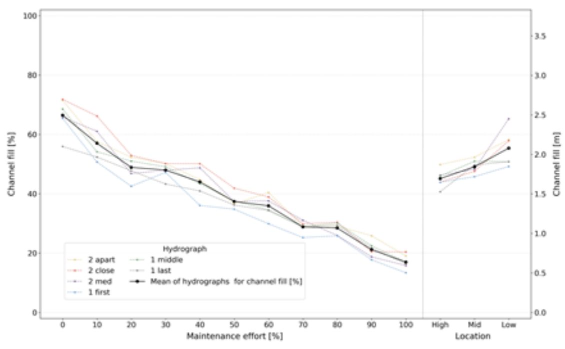
```
        plt.yticks(fontsize=l_size)
        plt.xlabel(xlabel, fontsize=ax_size, labelpad=8)
        plt.ylabel(ylabel, fontsize=ax_size, labelpad=15)
        ax.yaxis.grid(linestyle='--', alpha=0.3)
        plt.ylim(lim)
        plt.axhline(flood1[3, 0].astype(float), xmin=0.045, xmax=0.955, color='grey', linestyle='-.', linewidth=1, al-
pha=1)
        plt.savefig(save1, dpi=400, bbox_inches='tight')

        # mean + range flood scn
        fig = plt.figure(figsize=(19, 12))
        # fig.suptitle(title, fontsize=24, fontweight=1, color='black').set_position([.5, 0.94])
        ax = fig.add_subplot(1, 1, 1)

        ax.fill_between(min2[:, 1], max2[:, 0], min2[:, 0], color='#8e729d', alpha=0.4, label=label_range)
        ax.plot(min2[:, 1], min2[:, 0].astype(float), color='#8e729d', linestyle='-', linewidth=0.7)
        ax.plot(max2[:, 1], max2[:, 0].astype(float), color='#8e729d', linestyle='-', linewidth=0.7)
        ax.plot(flood2[:, 1], flood2[:, 0].astype(float), color='black', marker='.', linewidth=1, label=label_mean)
        ax.plot(6.5, flood1[3, 0].astype(float), color='grey', linestyle='-.', marker='x', markersize=9, label=label_30)
        ax.plot(6, flood1[3, 0].astype(float), color='grey', linestyle='-', marker='x', markersize=9)
        ax.plot(5.5, flood1[3, 0].astype(float), color='grey', linestyle='-', marker='x', markersize=9)
        handles, labels = plt.gca().get_legend_handles_labels()     # change order of labels in legend
        order = [0, 2, 1]
        legend = plt.legend([handles[idx] for idx in order], [labels[idx] for idx in order], ncol=1, fontsize=l_size,
                    title=legend_h, bbox_to_anchor=(0.042, 0.05), loc=3)
        plt.setp(legend.get_title(), fontsize=l_size)
        ax.yaxis.grid(linestyle='--', alpha=0.3)
        plt.xticks(np.arange(5.5, 7, 0.5), ['High', 'Mid', 'Low'], fontsize=l_size)
        plt.yticks(fontsize=l_size)
        plt.ylabel(ylabel, fontsize=ax_size, labelpad=15)
        ax.yaxis.grid(linestyle='--', alpha=0.3)
        plt.xlabel(xlabel, fontsize=ax_size, labelpad=8)
        plt.ylim(lim)
        plt.axhline(flood1[3, 0].astype(float), xmin=0.045, xmax=0.955, color='grey', linestyle='-.', linewidth=1, al-
pha=1)
        plt.savefig(save2, dpi=300, bbox_inches='tight')
        print('plots with location scenario''\n')
    return
def doExport(scenarios):
    '''export all arrays to ArcGIS files which include geometric information or to normal files'''
    ArcGIS = 'ncols 127' '\n' 'nrows 115' '\n' 'xllcorner 602510.99199495' '\n'\
         'yllcorner 175232.74791014' '\n' 'cellsize 15' '\n' 'NODATA_value 0.000000000000000'

    diff_name = 'U:simulations/analysis/python/channel change/DEMs/DEMdiff{x}.asc'
    scenario_list = 'U:simulations/analysis/python/list.txt'

    for scenario in range(scenarios):
        np.savetxt(diff_name.format(x=scenario), DEMdiff_thal0[scenario, 0, :, :], delimiter=' ', comments='',
header=ArcGIS)
    # np.savetxt(scenario_list, elev[:, :], delimiter=' ', comments='', fmt="%s")
    print('\n''saved them ALL''\n')

# -DEFINE GLOBAL VARIABLES HERE-
# years from which the difference should be calculated. (uncomment which one you want to calculate!)
#___________________________________________________________________________________________

# diff_yrs = [100]                                  # difference years reflect first & last year
# typ = ''
# after_during = "after"
diff_yrs = list(range(1, 100, 1))                     # a: difference years reflect random years
n = 100
typ = 'Random'+str(n)
after_during = "during"
```

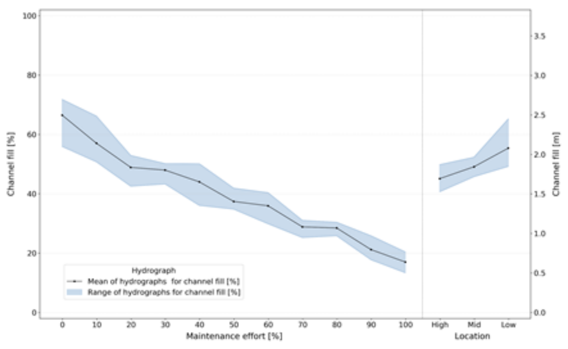
```
# diff_yrs = [1, 4, 21, 24, 50, 53, 95, 98]                    # b: difference years reflect big events
# typ = 'BigEvents'
# diff_yrs = [1, 8, 15, 23, 30, 38, 45, 52, 60, 67, 75, 82, 89, 97]       # c: difference years reflect all events
# typ = 'AllEvents'
# after_during = "during"

#________________________________________________________________________________________

# flood scn
floods = ['2 apart', '2 close', '2 med', '1 first', '1 middle', '1 last']

# mask for where to calculate the difference; thalweg = narrow/channel = wide. (uncomment which one you want
to calculate!)
profile = 'U:simulations/analysis/python/channel change/thalweg.txt'
# profile = 'U:simulations/analysis/python/channel change/channel.txt'

# -CALL FUNCTIONS HERE-
# calculate elevation difference of DEM between predefined years
DEMdiff0, DEMdiff = doDEMdiff(DEM.shape[0], diff_yrs)

# mask the DEMdiff file with the profile line (created in ArcGIS)
DEMdiff_thal0, DEMdiff_thal = doProfile(profile, DEMdiff.shape[0], diff_yrs, DEMdiff)

# calculate the mean, std and rmse of the total channel DEM
mean, std, rmse = doStatistics(DEMdiff.shape[0], diff_yrs, DEMdiff)

# thalweg only
mean_thal, std_thal, rmse_thal = doStatistics(DEMdiff_thal.shape[0], diff_yrs, DEMdiff_thal)

# calculate the mean, std and rmse of the thalweg only channel DEM
mean1_t, mean2_t = doArray(mean_thal)

# calculate the min and max of the elevation difference (thalweg only). if you want the whole channel, change input
to
# total DEM (DEMdiff). the min max can only be calculated if DEMdiff is calculated between year 100 and 0 and
not
# multiple with multiple diff_yrs.
min1, min2, max1, max2 = doMinMax(mean_thal)

# calculate the mean elevation diff for all flood scn for the different maintenance scn. also here only for diff_yrs=100.
floodm_t1, floodm_t2 = doFloodmean(mean_thal)

# plot maintenance and location scn
# define plot properties
xlabel = "                              Maintenance effort [%]" \
         "                                              Location"
xlabel_subplot = "Maintenance effort [%] and maintenance location"
ylabel = "Mean change in elevation [m]"
legend_h = "Hydrograph"
legend_y = "Year"
label_mean = "Mean of hydrographs"
label_range = "Range of hydrographs"
label_30 = "30 % maintenance"
lim = -0.35, 6.35
ax_size = 18
l_size = 16
title = "Change in elevation"+after_during+"100 years of simulation"

if mean_thal.shape[1] == 1:
    diff_yrs = 0                                   # set diff_yrs to zero, because its not defined
    save1 = 'U:simulations/analysis/python/channel change/SingleAll'+str(typ)+'_maint+loc.png'
    save2 = 'U:simulations/analysis/python/channel change/SingleRange'+str(typ)+'_maint+loc.png'
    save3 = 'U:simulations/analysis/python/channel change/SingleMean'+str(typ)+'_maint+loc.png'
    doPlot(mean1_t, mean2_t, floodm_t1, floodm_t2, min1, min2, max1, max2, mean_thal.shape[1], diff_yrs)
else:
    save1 = 'U:simulations/analysis/python/channel change/MultiAll'+str(typ)+'_maint+loc.png'
    save2 = 'U:simulations/analysis/python/channel change/MultiRange'+str(typ)+'_maint+loc.png'
    save3 = 'U:simulations/analysis/python/channel change/MultiMean'+str(typ)+'_maint+loc.png'
    doPlot(mean1_t, mean2_t, floodm_t1, floodm_t2, min1, min2, max1, max2, mean_thal.shape[1], diff_yrs)

# plot maintenance scn only
xlabel = "Maintenance effort [%]"
Channel filxlabel_subplot = "Maintenance effort [%]"

if mean_thal.shape[1] == 1:
    save1 = 'U:simulations/analysis/python/channel change/SingleAll_maint.png'
    save2 = 'U:simulations/analysis/python/channel change/SingleRange_maint.png'
    doPlot_maint(mean1_t, floodm_t1, min1, max1, mean_thal.shape[1], diff_yrs)
else:
    save1 = 'U:simulations/analysis/python/channel change/MultiSubplot'+str(typ)+'_maint.png'
    save2 = ''
    doPlot_maint(mean1_t, floodm_t1, min1, max1, mean_thal.shape[1], diff_yrs)

# plot location scn only
xlabel = "Maintenance location"
xlabel_subplot = "Maintenance location"

if mean_thal.shape[1] == 1:
    save1 = 'U:simulations/analysis/python/channel change/SingleAll'+str(typ)+'_loc.png'
    save2 = 'U:simulations/analysis/python/channel change/SingleRange'+str(typ)+'_loc.png'
    doPlot_loc(mean1_t, mean2_t, floodm_t1, floodm_t2, min2, max2, mean_thal.shape[1], diff_yrs)
else:
    save1 = 'U:simulations/analysis/python/channel change/MultiSubplot'+str(typ)+'_loc.png'
    save2 = ''
    doPlot_loc(mean1_t, mean2_t, floodm_t1, floodm_t2, min2, max2, mean_thal.shape[1], diff_yrs)

# export DEMdiff in ArcGIS readable format
if DEMdiff_thal.shape[1] == 1:
    doExport(DEMdiff.shape[0])
else:
    print('ERROR: exports only DEMs if diff_yrs = [100]. otherwise too many DEMs too store. if you want
them to be '
          'stored anyways, you have to add a second loop which loops over the diff_yrs"\n')
```

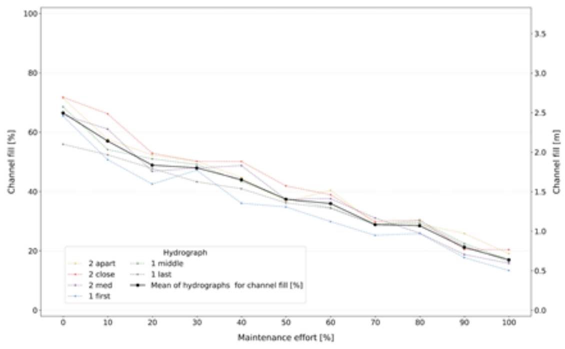
3.4 Tool 4: Sediment yield with potential channel fill over time

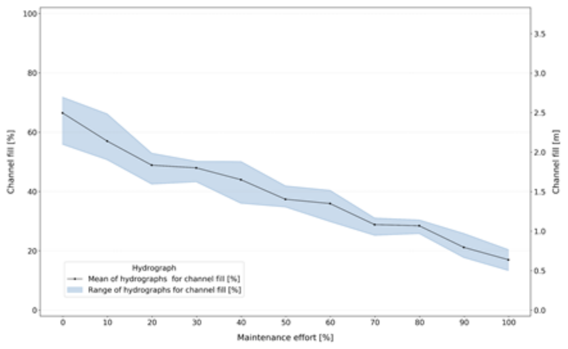
Sed yield + downstream fill: this script deals with the sediment yield generated at the output of the model. It calculates the total sum of sediment yield, cumulative sediment yield (evolution of sediment yield over simulation time), potential downstream channel fill in percentage and meters, and the cross section specific analysis of channel aggradation.

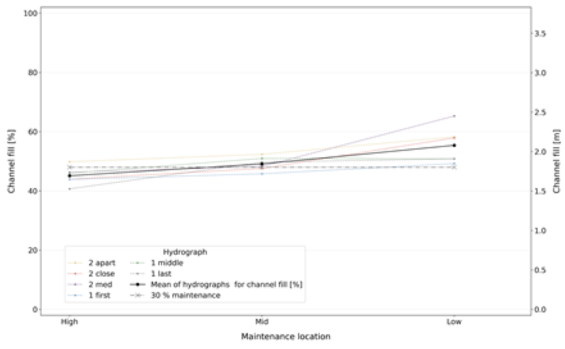
The following script produces these figures:

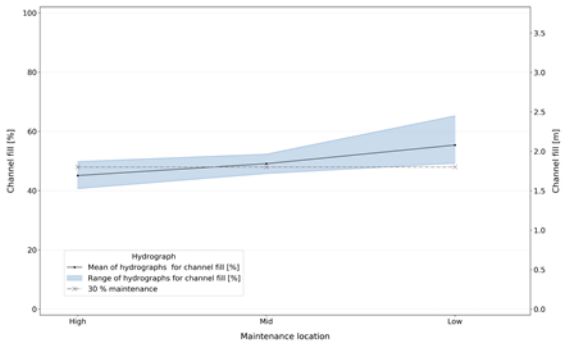

Channel fill [%]
Channel fill [m]
Maintenance effort [%]
Location
High
Mid
Low
Hydrograph
2 apart
2 close
2 med
1 first
1 middle
1 last
Mean of hydrographs for channel fill [%]

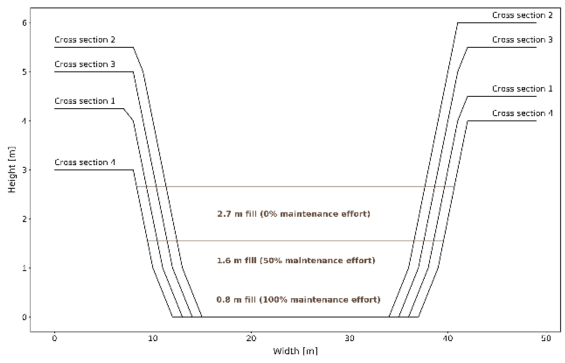

Channel fill [%]
Channel fill [m]
Maintenance effort [%]
Location
High
Mid
Low
Hydrograph
Mean of hydrographs for channel fill [%]
Range of hydrographs for channel fill [%]

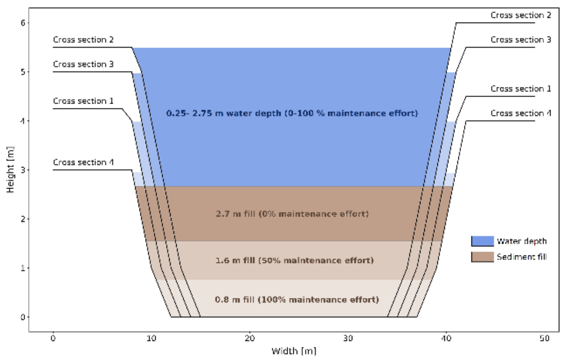

Channel fill [%]
Channel fill [m]
Maintenance effort [%]
Hydrograph
2 apart
2 close
2 med
1 first
1 middle
1 last
Mean of hydrographs for channel fill [%]

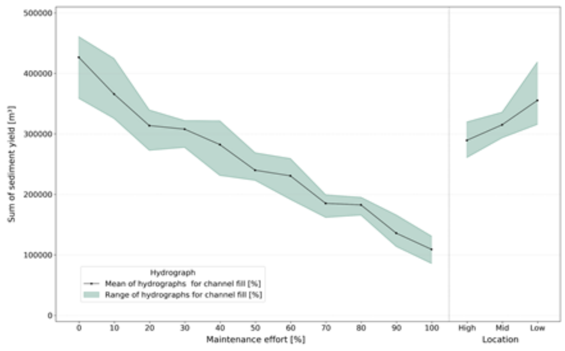

Channel fill [%]
Channel fill [m]
Maintenance effort [%]
Hydrograph
Mean of hydrographs for channel fill [%]
Range of hydrographs for channel fill [%]

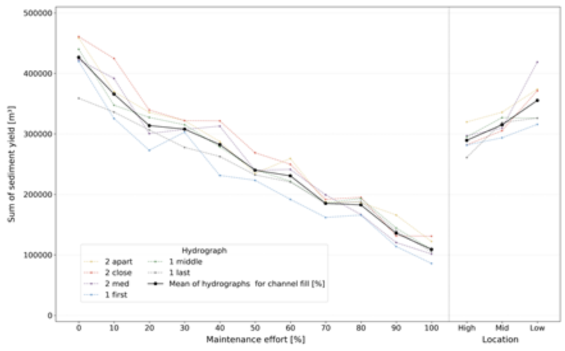

Channel fill [%]
Channel fill [m]
Maintenance location
High
Mid
Low
Hydrograph
2 apart
2 close
2 med
1 first
1 middle
1 last
Mean of hydrographs for channel fill [%]
30 % maintenance

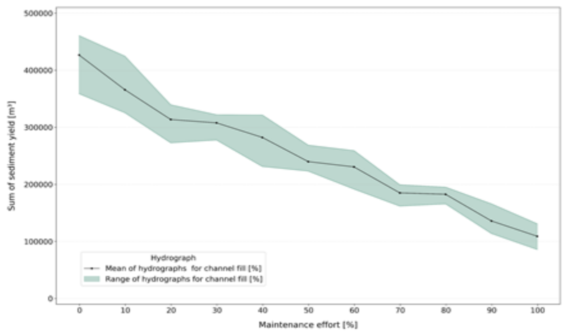

Channel fill [%]
Channel fill [m]
Maintenance location
High
Mid
Low
Hydrograph
Mean of hydrographs for channel fill [%]
Range of hydrographs for channel fill [%]
30 % maintenance


Height [m]
Width [m]
Cross section 2
Cross section 3
Cross section 1
Cross section 4
2.7 m fill (0% maintenance effort)
1.6 m fill (50% maintenance effort)
0.8 m fill (100% maintenance effort)

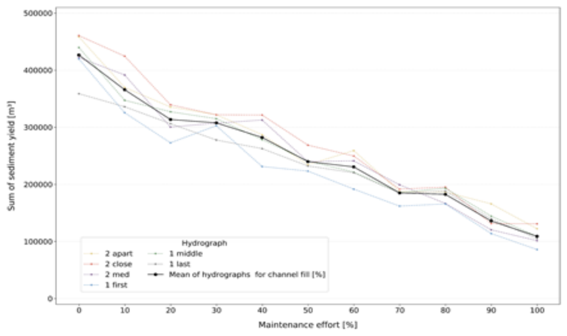

Height [m]
Width [m]
Cross section 2
Cross section 3
Cross section 1
Cross section 4
0.25- 2.75 m water depth (0-100 % maintenance effort)
2.7 m fill (0% maintenance effort)
1.6 m fill (50% maintenance effort)
0.8 m fill (100% maintenance effort)
Water depth
Sediment fill


Sum of sediment yield [m³]
Maintenance effort [%]
Location
High
Mid
Low
Hydrograph
Mean of hydrographs for channel fill [%]
Range of hydrographs for channel fill [%]

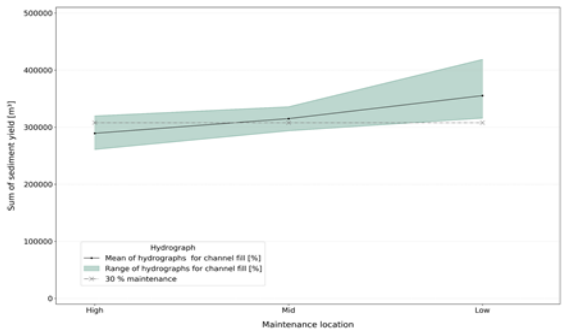

Sum of sediment yield [m³]
Maintenance effort [%]
Location
High
Mid
Low
Hydrograph
2 apart
2 close
2 med
1 first
1 middle
1 last
Mean of hydrographs for channel fill [%]



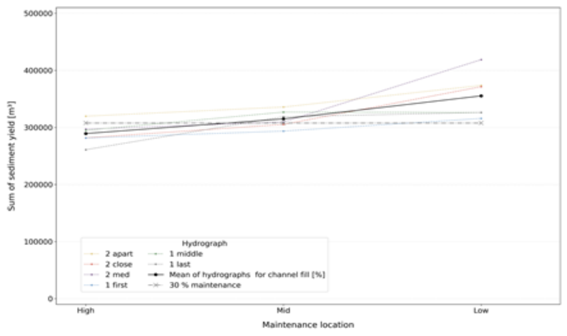
Sum of sediment yield [m³]
Maintenance effort [%]
Hydrograph
Mean of hydrographs for channel fill [%]
Range of hydrographs for channel fill [%]



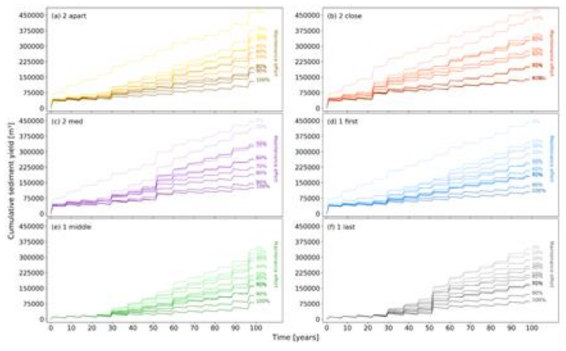
Sum of sediment yield [m³]
Maintenance effort [%]
Hydrograph
2 apart
2 close
2 med
1 first
1 middle
1 last
Mean of hydrographs for channel fill [%]



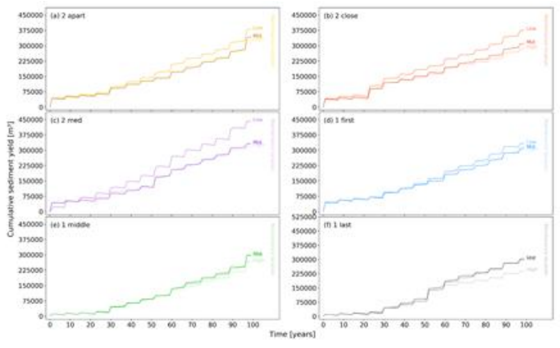
Sum of sediment yield [m³]
Maintenance location
High
Mid
Low
Hydrograph
Mean of hydrographs for channel fill [%]
Range of hydrographs for channel fill [%]
30 % maintenance



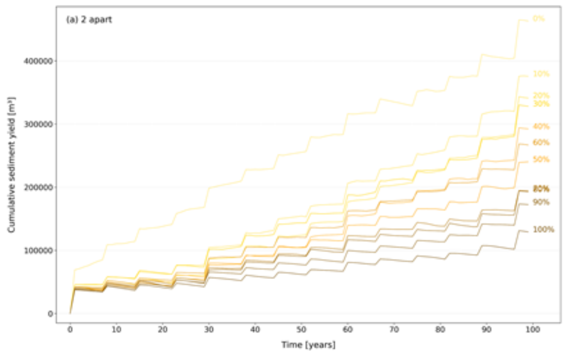
Sum of sediment yield [m³]
Maintenance location
High
Mid
Low
Hydrograph
2 apart
2 close
2 med
1 first
1 middle
1 last
Mean of hydrographs for channel fill [%]
30 % maintenance



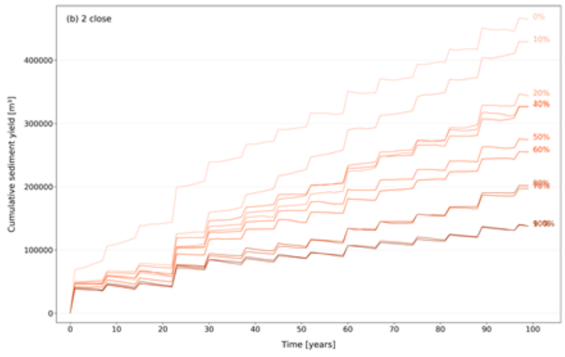
(a) 2 apart
(b) 2 close
(c) 2 med
(d) 1 first
(e) 1 middle
(f) 1 last
Cumulative sediment yield [m³]
Time [years]



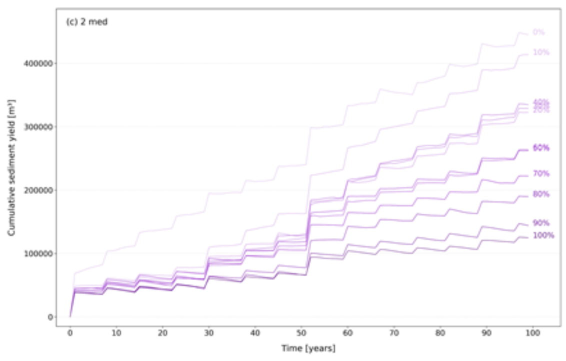
(a) 2 apart
(b) 2 close
(c) 2 med
(d) 1 first
(e) 1 middle
(f) 1 last
Cumulative sediment yield [m³]
Time [years]



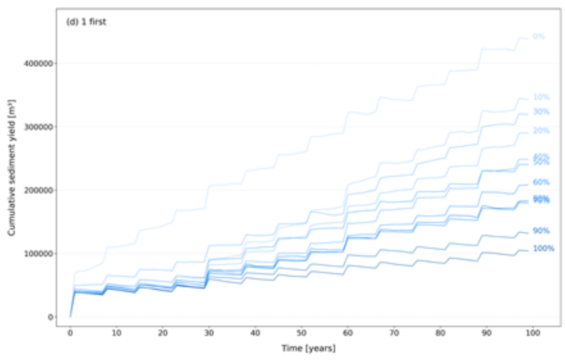
(a) 2 apart
Cumulative sediment yield [m³]
Time [years]
0%
10%
20%
30%
40%
60%
50%
80%
90%
100%



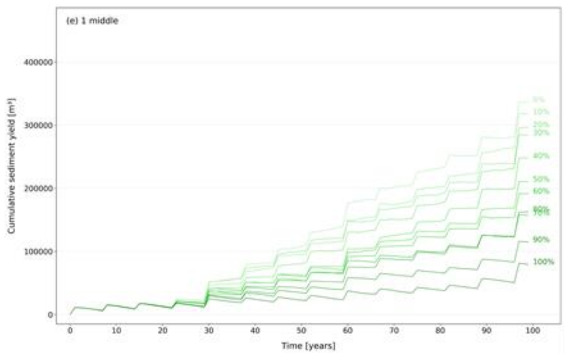
(b) 2 close
Cumulative sediment yield [m³]
Time [years]
0%
10%
20%
40%
50%
60%
100%

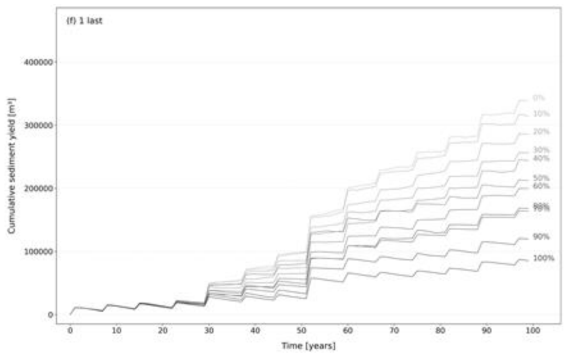


(c) 2 med
Cumulative sediment yield [m³]
Time [years]
0%
10%
60%
70%
80%
90%
100%



(d) 1 first
Cumulative sediment yield [m³]
Time [years]
0%
10%
30%
20%
60%
90%
100%





```
# SED YIELD + DOWNSTREAM FILL: this script deals with the sediment yield generated
# at the output of the model. it calculates the total sum of sediment yield, cumu-
# lative sediment yield (evolution of sediment yield over simulation time), poten-
# tial downstream channel fill in percentage and meters, and the cross section spe-
# cific analysis of channel aggradation.

# -IMPORT LIBRARIES & VARIABLES HERE-
import numpy as np
import glob
import matplotlib.pyplot as plt
from tool2c_channel_change import doColors

# -DEFINE FUNCTIONS HERE-
def doRead(files):
    '''read in the .dat file which includes the hourly sediment yield values'''
    print('\n"IMPORTAMT MESSAGE:"\n"\n"sorry to tell you that this is gonna take A LOOOOOONG time (5-
10min). maybe'
          ' you wanna go wash some dishes or start cooking dinner, while this is loading in the files."\n')
    dat_data = []
    for x in range(files.shape[0]):
        read_files = np.genfromtxt(files[x], delimiter=' ', usecols=(1, 4), skip_header=3)            # only read in Qw &
Qs
        dat_data.append(read_files)
        if (x % 6 == 0):
            print('file ' + str(x) + ' is created. Only ' + str(files.shape[0]-x) + ' to go!')
    dat_data = np.array(dat_data)
    print('\n"well, that took a while, thank\'s for being so patient. hope dinner or dishes or even both are done
by now"\n')
    return dat_data
def doSumSed(file):
    '''calculate sum of sediment yield for all scenarios'''
    sumsed = []
    for x in range(file.shape[0]):
        sum_Qs = np.sum(file[x, :, 1])
        sumsed.append(sum_Qs)
    sumsed = np.array(sumsed)
    return sumsed
def doCumsum(file):
    '''calculate cumulative sum of sediment yield for all scenarios'''
    cumsum = []
    for x in range(0, file.shape[0]):
        csum_Qs = np.cumsum(file[x, :, 1])
        cumsum.append(csum_Qs)
        if x % 14 == 0:
            print('cumsum ' + str(x) +' done. only ' + str(file.shape[0]-x) + ' to go!')
    cumsum = np.array(cumsum)

    # get cumsum of each year (otherwise array is too big to calculate)
    cumsum = cumsum[:, np.arange(0, cumsum.shape[1], (24*365))]
    print('ALL done! cumsum is calculated."\n')
    return cumsum
def doNewArray(input):
    '''create arrays to original elev array: specific maintenance scn ("perc"), location scn ("loc"), flood scn ("flood")'''
    perc_loc = np.repeat(np.arange(0, 7, 0.5), 6).reshape(input.shape[0], 1)
    new_array = np.append(np.vstack(input), perc_loc, axis=1)  # combine the new created perc_loc
    floods = np.array(14 * ['2apart', '2close', '2med', 'a_first', 'b_middle', 'c_last']).reshape(input.shape[0], 1)
    new_array = np.append(new_array, floods, axis=1)  # append third column to STD array
    return new_array
def doArray(sed_a, sed_m):
    '''create new array which is sorted in the right way for analyzing summary statistics of difference'''
    new_array = doNewArray(sed_a)

    # sort array by following order: 1st by flood scenarios (3rd col), 2nd by maintenance/location scenarios (2nd col)
    a1 = new_array[:, 0]
    a2 = new_array[:, 1]
    b = new_array[:, -2]
    c = new_array[:, -1]

    ind = np.lexsort((a1, a2, b, c))  # create array with the specified order
    sort = np.array([(a1[i], a2[i], b[i], c[i]) for i in ind])  # apply the "sorting array" to the original array

    splits = np.array(np.split(sort[:, :], 6))
    splits = np.array(splits)

    # location & maintenance scenarios need to be split in order to plot them
    split1 = []
    split2 = []
    for x in range(splits.shape[0]):
        first = splits[x, :11, :]
        last = splits[x, -3:, :]
        split1.append(first)
        split2.append(last)
    split1 = np.array(split1)
    split2 = np.array(split2)

    perc_loc2 = np.append((5, 4.5, 4, 3.5, 3, 2.5, 2, 1.5, 1, 0.5, 0), np.array([5.5, 6, 6.5]))
    m_sed2 = np.append(np.append(sed_m[:,0], sed_m[:, 1]), perc_loc2).reshape(3, sed_m.shape[0])

    msplit1 = np.array([(m_sed2[:, x]) for x in range(11)])
    msplit2 = np.array([(m_sed2[:, x]) for x in range(11, 14)])

    print('array created''\n')
    return split1, split2, msplit1, msplit2
def doMinMax(sed):
    '''calculates the min and the max value for each flood scenario. depending on if only 1 or 2 sediment yield values
    are analyzed, the calculation method is adapted'''
    new_array = doNewArray(sed)

    if len(sed.shape) == 2:
        a1 = new_array[:, 0]
        a2 = new_array[:, 1]
        b = new_array[:, -1]
        c = new_array[:, -2]

        # sort array by following order: 1st by flood scenarios (3rd col), 2nd by maintenance/location scenarios (2nd
col)
        ind = np.lexsort((a1, a2, b, c))  # create array with the specified order
        sort = np.array([(a1[i], a2[i], b[i], c[i]) for i in ind])  # apply the "sorting array" to the original array

        splits = np.array(np.split(sort[:, :], 14))
        splits = np.array(splits)

        min_p_f = []
        min_m_f = []
        for x in range(splits.shape[0]):  # loop through all scenarios
            min_p = np.min((splits[x, :, 0]).astype('float'))   # mean of fill [%]
            min_m = np.min((splits[x, :, 1]).astype('float'))   # mean of fill [m]
            min_p_f.append(min_p)
            min_m_f.append(min_m)
        min_flood = np.append(min_p_f, min_m_f).reshape(2,splits.shape[0])

        max_p_f = []
        max_m_f = []
        for x in range(splits.shape[0]):  # loop through all scenarios
            max_p = np.max((splits[x, :, 0]).astype('float'))
            max_m = np.max((splits[x, :, 1]).astype('float'))
            max_p_f.append(max_p)
            max_m_f.append(max_m)
        max_flood = np.append(max_p_f, max_m_f).reshape(2,splits.shape[0])

        perc_loc2 = np.append((0, .5, 1, 1.5, 2, 2.5, 3, 3.5, 4, 4.5, 5), np.array([5.5, 6, 6.5]))

        min = np.append(min_flood[:], perc_loc2).reshape(3,min_flood.shape[1])
        min1 = np.array([(min[:, x]) for x in range(11)])
        min2 = np.array([(min[:, x]) for x in range(11, 14)])

        max = np.append(max_flood[:], perc_loc2).reshape(3,max_flood.shape[1])
        max1 = np.array([(max[:, x]) for x in range(11)])
        max2 = np.array([(max[:, x]) for x in range(11, 14)])

    else:
        a = new_array[:, 0]
        b = new_array[:, -1]
        c = new_array[:, -2]

        ind = np.lexsort((a, b, c))  # create array with the specified order
        sort = np.array([(a[i], b[i], c[i]) for i in ind])  # apply the "sorting array" to the original array

        splits = np.array(np.split(sort[:, :], 14))
        splits = np.array(splits)

        min_p_f = []
        for x in range(splits.shape[0]):  # loop through all scenarios
            min_p = np.min((splits[x, :, 0]).astype('float'))   # mean of fill [%]
            min_p_f.append(min_p)
        min_flood = np.array(min_p_f)

        max_p_f = []
        for x in range(splits.shape[0]):  # loop through all scenarios
            max_p = np.max((splits[x, :, 0]).astype('float'))
            max_p_f.append(max_p)
        max_flood = np.array(max_p_f)

        perc_loc2 = np.append((0, .5, 1, 1.5, 2, 2.5, 3, 3.5, 4, 4.5, 5), np.array([5.5, 6, 6.5]))

        min = np.append(min_flood[:], perc_loc2).reshape(2, min_flood.shape[0])
        min1 = np.array([(min[:, x]) for x in range(11)])
        min2 = np.array([(min[:, x]) for x in range(11, 14)])

        max = np.append(max_flood[:], perc_loc2).reshape(2, max_flood.shape[0])
        max1 = np.array([(max[:, x]) for x in range(11)])
        max2 = np.array([(max[:, x]) for x in range(11, 14)])
    print('min and max calculated''\n')
    return min1, min2, max1, max2
def doFloodmean(elev_diff):
    '''calculates the flood mean of all flood scenarios'''
    new_array = doNewArray(elev_diff)

    # sort array by following order: 1st by maintenance scenarios (3rd col), 2nd by flood scenarios (2nd col)
    a = new_array[:, 0]
    b = new_array[:, -1]
    c = new_array[:, -2]

    ind = np.lexsort((a, b, c))  # create array with the specified order
    sort = np.array([(a[i], b[i], c[i]) for i in ind])  # apply the "sorting array" to the original array

    splits = np.array(np.split(sort[:, :], 14))
    splits = np.array(splits)

    # calculate mean of flood scn for different maintenance scn
    mean_flood = []
    for x in range(splits.shape[0]):  # loop through all scenarios
        mean_m = np.mean((splits[x, :, 0]).astype('float'))  # build mean of DEMdiff for each cell during 100yrs
        mean_flood.append(mean_m)
    mean_flood = np.array(mean_flood)

    perc_loc2 = np.append((0, .5, 1, 1.5, 2, 2.5, 3, 3.5, 4, 4.5, 5), np.array([5.5, 6, 6.5]))
    mean_flood2 = np.append(mean_flood, perc_loc2).reshape(2, mean_flood.shape[0])

    split1 = np.array([(mean_flood2[:, x]) for x in range(11)])
    split2 = np.array([(mean_flood2[:, x]) for x in range(11, 14)])

    print('mean_flood calculated''\n')
    return split1, split2
def doSplit(cumsum):
    '''split the new created array into the maitnenance effort and location scenarios'''
    perc_loc = np.repeat(np.array([0, 5, 0.5, 1, 1.5, 2, 2.5, 3, 3.5, 4, 4.5, 5.5, 6, 6.5]), 6).reshape(cumsum.shape[0],
1)
    new_array = np.append(np.vstack(cumsum), perc_loc, axis=1)  # combine the new created perc_loc
    floods = np.array(int(cumsum.shape[0]/6) * ['2apart', '2close', '2med', 'a_first',
                                               'b_middle', 'c_last']).reshape(cumsum.shape[0], 1)
    new_array = np.append(new_array, floods, axis=1)  # append third column to STD array

    # sort array by maintenance scn
    a = new_array[:, range(0, cumsum.shape[1])]
    b = new_array[:, -1]    # flood scn
    c = new_array[:, -2]    # maint scn

    sorts = []
    for x in range(a.shape[1]):
        ind = np.lexsort((a[:, x], b, c))  # create array with the specified order
        sort = np.array([(a[:, x][i], b[i], c[i]) for i in ind])  # apply the "sorting array" to the original array
        sorts.append(sort)
    sorts = np.array(sorts)
    comb = np.concatenate(sorts[:, :, 0]).reshape(a.shape[1], a.shape[0])
    split = np.array(np.split(comb[:, :], (cumsum.shape[0]/6), axis=1)).astype('float32')  # split array into different
                                                                                          # flood scenarios (6)

    # substract offset (174400m^3) from arrays
    offset = np.transpose((6*[(np.arange(0, 174400, 1744))]))
    split_off = []
    for x in range(split.shape[0]):
        interlist = []
        for y in range(split.shape[2]):
            off = split[x, :, y]-offset[:, y]
            interlist.append(off)
        split_off.append(interlist)
    split_off = np.array(split_off)

    # calculate mean of flood scn for different maintenance scn
    offset = np.arange(0, 174400, 1744)
    mean = []
    for x in range(split.shape[0]):  # loop through all scenarios
        interlist = []
        for y in range(split.shape[1]):
            mean_m = np.mean((split[x, y, :]).astype('float'))  # build mean of DEMdiff for each cell during 100yrs
            interlist.append(mean_m)
        mean.append(interlist)
    mean = np.array(mean)
    mean_off = mean-offset

    # calculate min + max of flood scn for different maintenance scn
    min = []
    for x in range(split.shape[0]):  # loop through all scenarios
        interlist = []
        for y in range(split.shape[1]):
            mean_m = np.min((split[x, y, :]).astype('float'))  # build mean of DEMdiff for each cell during 100yrs
            interlist.append(mean_m)
        min.append(interlist)
    min = np.array(min)
    min_off = min-offset

    max = []
    for x in range(split.shape[0]):  # loop through all scenarios
        interlist = []
        for y in range(split.shape[1]):
            mean_m = np.max((split[x, y, :]).astype('float'))  # build mean of DEMdiff for each cell during 100yrs
            interlist.append(mean_m)
        max.append(interlist)
    max = np.array(max)
    max_off = max-offset
    print('profile split into maintenance scenarios. mean of different flood scenarios calculated.''\n')
    return split_off
def doPlot(scenario, diff1, diff2, flood1, flood2, min1, min2, max1, max2, color, xlabel, ylabel1, ylabel2,
           title, save1, save2):
    '''create two different plots, first one for all flood scn seperately, second one for the mean of all flood scn and
    its range. the plots includes the location scenarios'''
    print('let\'s plot now''\n')
    # 1st plot: all flood scn
    floods = ['2 apart', '2 close', '2 med', '1 first', '1 middle', '1 last']
    palette = np.array(['#e3ce8d', '#db786c', '#8e729d', '#7ba6d0', '#7ba47b', '#8d8d8d'])

    fig = plt.figure(figsize=(19, 12))
    # fig.suptitle(title, fontsize=24, fontweight=1, color='black').set_position([.5, 0.94])
    ax1 = fig.add_subplot(1, 1, 1)
    for x in range(scenario):
        ax1.plot(diff1[x, :, 2], diff1[x, :, 0].astype(float), color=palette[x],
                 marker='.', linestyle='--', linewidth=1, label=floods[x])
        ax1.plot(diff2[x, :, 2], diff2[x, :, 0].astype(float), color=palette[x],
                 marker='.', linestyle='--', linewidth=1)
    ax1.plot(flood1[:, -1], flood1[:, 0].astype(float), color='black',
             marker='o', linestyle='-', linewidth=1.25, label=lab_mean)
    ax1.plot(flood2[:, -1], flood2[:, 0].astype(float), color='black',
             marker='o', linestyle='-', linewidth=1.25)
    legend = plt.legend(ncol=2, fontsize=l_size, title=legend_h, bbox_to_anchor=(0.042, 0.05), loc=3)
    plt.setp(legend.get_title(), fontsize=l_size)
    plt.ylabel(ylabel1, fontsize=ax_size, labelpad=15)
    plt.xlabel(xlabel, fontsize=ax_size, labelpad=8)
    ax1.yaxis.grid(linestyle='--', alpha=0.3)
    plt.xticks(fontsize=l_size)
    plt.yticks(fontsize=l_size)
    plt.ylim(lim_range)

    if max1.shape[1] > 2:
        ax2 = ax1.twinx()
        ax2.plot(flood1[:, 2], flood1[:, 1].astype(float), color='black', linestyle='-', marker='.', linewidth=0.01, alpha=0)
        plt.yticks(np.arange(0, 4.5, 0.5), np.arange(0, 4.5, 0.5), fontsize=l_size)
        plt.ylabel(ylabel2, fontsize=ax_size, labelpad=15)
        plt.ylim(-0.075, 3.83)
        plt.yticks(fontsize=l_size)
    plt.xticks([r + 0.005 for r in range(0, 14)], [0, 10, 20, 30, 40, 50, 60, 70, 80, 90, 100, 'High', 'Mid', 'Low'],
               fontsize=l_size)
    plt.axvline(x=10.5, color='black', linestyle='--', linewidth=0.4)
    # plt.axvspan(0, 2, color='grey', alpha=0.05, lw=0)
    # plt.axvspan(8, 10, color='grey', alpha=0.05, lw=0)
    plt.savefig(save1, dpi=300, bbox_inches='tight')

    # 2nd plot: range of flood scn
    fig = plt.figure(figsize=(19, 12))
    # fig.suptitle(title, fontsize=24, fontweight=1, color='black').set_position([.5, 0.94])
    ax1 = fig.add_subplot(1, 1, 1)
    ax1.plot(min1[:, -1], min1[:, 0].astype(float), color=color, linestyle='-', linewidth=0.7)
    ax1.plot(min2[:, -1], min2[:, 0].astype(float), color=color, linestyle='-', linewidth=0.7)
    ax1.plot(max1[:, -1], max1[:, 0].astype(float), color=color, linestyle='-', linewidth=0.7)
    ax1.plot(max2[:, -1], max2[:, 0].astype(float), color=color, linestyle='-', linewidth=0.7)
    ax1.plot(flood1[:, -1], flood1[:, 0].astype(float), color='black',
            linestyle='-', marker='.', linewidth=1, label=lab_mean)
    ax1.plot(flood2[:, -1], flood2[:, 0].astype(float), color='black',
            linestyle='-', marker='.', linewidth=1)
    plt.fill_between(min1[:, -1], max1[:, 0], min1[:, 0], color=color, alpha=0.4)
    plt.fill_between(min2[:, -1], max2[:, 0], min2[:, 0], color=color, alpha=0.4, label=lab_range)
    legend = plt.legend(ncol=1, fontsize=l_size, title=legend_h, bbox_to_anchor=(0.042, 0.05), loc=3)
    plt.setp(legend.get_title(), fontsize=l_size)
    ax1.yaxis.grid(linestyle='--', alpha=0.3)
    plt.ylabel(ylabel1, fontsize=ax_size, labelpad=15)
    plt.xlabel(xlabel, fontsize=ax_size, labelpad=8)
    plt.xticks(fontsize=l_size)
    plt.yticks(fontsize=l_size)
    plt.ylim(lim_range)

    if max1.shape[1] > 2:
        ax2 = ax1.twinx()
        ax2.plot(flood1[:, -1], flood1[:, 1].astype(float), color='black',
                linestyle='-', linewidth=1, label=lab_mean, alpha=0)
        plt.yticks(np.arange(0, 4.5, 0.5), np.arange(0, 4.5, 0.5), fontsize=l_size)
        plt.ylabel(ylabel2, fontsize=ax_size, labelpad=15)
        plt.ylim(-0.075, 3.83)
        plt.yticks(fontsize=l_size)
    plt.xticks([r + 0.005 for r in np.arange(0, 7, 0.5)], [0, 10, 20, 30, 40, 50, 60, 70, 80, 90, 100,
                                                'High', 'Mid', 'Low'], fontsize=l_size)
    plt.axvline(x=5.25, color='black', linestyle='--', linewidth=0.4)
    # plt.axvspan(0, 1, color='grey', alpha=0.05, lw=0)
    # plt.axvspan(4, 5, color='grey', alpha=0.05, lw=0)
    plt.savefig(save2, dpi=300, bbox_inches='tight')
def doPlot_maint(scenario, diff1, flood1, min1, max1, color, xlabel, ylabel1, ylabel2, title, save1, save2):
    '''create two different plots, first one for all flood scn seperately, second one for the mean of all flood scn and
    its range. the plots only show the maintenance effort scenarios'''
    # 1st plot: all flood scn
    floods = ['2 apart', '2 close', '2 med', '1 first', '1 middle', '1 last']
    fig = plt.figure(figsize=(19, 12))
    # fig.suptitle(title, fontsize=24, fontweight=1, color='black').set_position([.5, 0.94])
    palette = np.array(['#e3ce8d', '#db786c', '#8e729d', '#7ba6d0', '#7ba47b', '#8d8d8d'])

    ax1 = fig.add_subplot(1, 1, 1)
    for x in range(scenario):
        ax1.plot(diff1[x, :, 2], diff1[x, :, 0].astype(float), color=palette[x],
                marker='.', linestyle='--', linewidth=1, label=floods[x])
    ax1.plot(flood1[:, -1], flood1[:, 0].astype(float), color='black',
            marker='o', linestyle='-', linewidth=1.25, label=lab_mean)

    legend = plt.legend(ncol=2, fontsize=l_size, title=legend_h, bbox_to_anchor=(0.042, 0.03), loc=3)
    plt.setp(legend.get_title(), fontsize=l_size)
    ax1.yaxis.grid(linestyle='--', alpha=0.3)
    plt.ylabel(ylabel1, fontsize=ax_size, labelpad=l_size)
    plt.xlabel(xlabel, fontsize=ax_size, labelpad=l_size)
    plt.yticks(fontsize=l_size)
    plt.xticks(fontsize=l_size)
    plt.ylim(lim_range)

    if max1.shape[1] > 2:
        ax2 = ax1.twinx()
        ax2.plot(max1[:, 2], flood1[:, 1].astype(float), color='maroon', alpha=0, linestyle='-.', marker='x')
        plt.ylabel(ylabel2, fontsize=ax_size, labelpad=15)
        plt.yticks(np.arange(0, 4.5, 0.5), np.arange(0, 4.5, 0.5), fontsize=l_size)
        plt.ylim(-0.075, 3.83)
    plt.xticks([r + 0.005 for r in range(0, 11)], [0, 10, 20, 30, 40, 50, 60, 70, 80, 90, 100], fontsize=l_size)
    plt.yticks(fontsize=l_size, color='maroon')

    plt.savefig(save1, dpi=475, bbox_inches='tight')

    # 2nd plot: range of flood scn
    fig = plt.figure(figsize=(19, 12))
    # fig.suptitle(title, fontsize=24, fontweight=1, color='black').set_position([.5, 0.94])
    ax1 = fig.add_subplot(1, 1, 1)
    ax1.plot(min1[:, -1], min1[:, 0].astype(float), color=color,
             linestyle='-', linewidth=0.7)
    ax1.plot(max1[:, -1], max1[:, 0].astype(float), color=color,
             linestyle='-', linewidth=0.7)
    ax1.plot(flood1[:, -1], flood1[:, 0].astype(float), color='black',
             linestyle='-', marker='.', linewidth=1, label=lab_mean)
    plt.fill_between(min1[:, -1], max1[:, 0], min1[:, 0], color=color, alpha=0.4, label=lab_range)
    ax1.yaxis.grid(linestyle='--', alpha=0.3)
    plt.ylabel(ylabel1, fontsize=ax_size, labelpad=l_size)
    plt.xlabel(xlabel, fontsize=ax_size, labelpad=l_size)
    plt.xticks(fontsize=l_size)
    plt.yticks(fontsize=l_size)
    plt.ylim(lim_range)
    legend = plt.legend(ncol=1, fontsize=l_size, title=legend_h, bbox_to_anchor=(0.042, 0.05), loc=3)
    plt.setp(legend.get_title(), fontsize=l_size)
    # get second y-axis
    if max1.shape[1] > 2:
        ax2 = ax1.twinx()
        ax2.plot(flood1[:, -1], flood1[:, 1].astype(float), color='maroon', linestyle='-.', linewidth=0, alpha=0)
        plt.ylabel(ylabel2, fontsize=ax_size, labelpad=15)
        plt.yticks(np.arange(0, 4.5, 0.5), np.arange(0, 4.5, 0.5), fontsize=l_size)
        plt.ylim(-0.075, 3.83)
        # plt.legend(ncol=1, fontsize=l_size, bbox_to_anchor=(0.042, 0.03), loc=3)
    plt.xticks([r + 0.005 for r in np.arange(0, 5.5, 0.5)], [0, 10, 20, 30, 40, 50, 60, 70, 80, 90, 100], fontsize=l_size)
    plt.savefig(save2, dpi=400, bbox_inches='tight')
def doPlot_loc(scenario, diff1, diff2, flood1, flood2, min1, min2, max1, max2, color, xlabel, ylabel1, ylabel2,
               title, save1, save2):
    '''create two different plots, first one for all flood scn seperately, second one for the mean of all flood scn and
    its range. the plots only includes the location scenarios'''
    # 1st plot: all flood scn
    floods = ['2 apart', '2 close', '2 med', '1 first', '1 middle', '1 last']
    fig = plt.figure(figsize=(19, 12))
    # fig.suptitle(title, fontsize=24, fontweight=1, color='black').set_position([.5, 0.94])
    palette = np.array(['#e3ce8d', '#db786c', '#8e729d', '#7ba6d0', '#7ba47b', '#8d8d8d'])

    ax1 = fig.add_subplot(1, 1, 1)
    for x in range(scenario):
        ax1.plot(diff2[x, :, 2], diff2[x, :, 0].astype(float), color=palette[x],
                 marker='.', linestyle='--', linewidth=1, label=floods[x])
    ax1.plot(flood2[:, -1], flood2[:, 0].astype(float), color='black',
             marker='o', linestyle='-', linewidth=1.3, label=lab_mean)
    ax1.plot(2, flood1[z, 0].astype(float), color='grey', marker='x', linestyle='-.', markersize=9, label=label_30)
    ax1.plot(1, flood1[z, 0].astype(float), color='grey', marker='x', linestyle='-', markersize=9)
    ax1.plot(0, flood1[z, 0].astype(float), color='grey', marker='x', linestyle='-', markersize=9)
    legend = plt.legend(ncol=2, fontsize=l_size, title=legend_h, bbox_to_anchor=(0.042, 0.03), loc=3)
    plt.setp(legend.get_title(), fontsize=l_size)
    ax1.yaxis.grid(linestyle='--', alpha=0.3)
    plt.ylabel(ylabel1, fontsize=ax_size, labelpad=l_size)
    plt.xlabel(xlabel, fontsize=ax_size, labelpad=l_size)
    plt.yticks(fontsize=l_size)
    plt.xticks(fontsize=l_size)
    plt.ylim(lim_range)
    plt.axhline(flood1[z, 0].astype(float), xmin=0.06, xmax=0.844, color='grey', linestyle='-.', linewidth=1.6, alpha=1)

    if max1.shape[1] > 2:
        ax2 = ax1.twinx()
        ax2.plot(flood1[:, -1], flood1[:, 1].astype(float), color='maroon', linestyle='-.', linewidth=0, alpha=0)
        plt.ylabel(ylabel2, fontsize=ax_size, labelpad=15)
        plt.yticks(np.arange(0, 4.5, 0.5), np.arange(0, 4.5, 0.5), fontsize=l_size)
        plt.ylim(-0.075, 3.83)
        plt.yticks(fontsize=l_size)

    plt.xticks(np.arange(0, 3, 1), ['High', 'Mid', 'Low'], fontsize=l_size)
    # plt.yticks(fontsize=l_size)
    plt.xlim(-.15, 2.4)
    plt.savefig(save1, dpi=300, bbox_inches='tight')

    # 2nd plot: range of flood scn
    fig = plt.figure(figsize=(19, 12))
    # fig.suptitle(title, fontsize=24, fontweight=1, color='black').set_position([.5, 0.94])
    ax1 = fig.add_subplot(1, 1, 1)
    ax1.fill_between(min2[:, -1], max2[:, 0], min2[:, 0],color=color, alpha=0.4, label=lab_range)
    ax1.plot(min2[:, -1], min2[:, 0].astype(float), color=color, linestyle='-', linewidth=0.7)
    ax1.plot(max2[:, -1], max2[:, 0].astype(float), color=color, linestyle='-', linewidth=0.7)
    ax1.plot(flood2[:, -1], flood2[:, 0].astype(float), color='black',
             linestyle='-', marker='.', linewidth=1, label=lab_mean)
    ax1.plot(6.5, flood1[z, 0].astype(float), color='grey', linestyle='-.', marker='x', markersize=9, label=label_30)
    ax1.plot(6, flood1[z, 0].astype(float), color='grey', linestyle='-', marker='x', markersize=9)
    ax1.plot(5.5, flood1[z, 0].astype(float), color='grey', linestyle='-', marker='x', markersize=9)
    handles, labels = plt.gca().get_legend_handles_labels()     # change order of labels in legend
    order = [0, 2, 1]

    ax1.yaxis.grid(linestyle='--', alpha=0.3)
    plt.ylabel(ylabel1, fontsize=ax_size, labelpad=l_size)
    plt.xlabel(xlabel, fontsize=ax_size, labelpad=l_size)
    plt.xticks(np.arange(5.5, 7, 0.5), ['High', 'Mid', 'Low'], fontsize=l_size)
    plt.yticks(fontsize=l_size)
    plt.ylim(lim_range)
    plt.axhline(flood1[z, 0].astype(float), xmin=0.077, xmax=0.844, color='grey', linestyle='-.', linewidth=1.2, al-
pha=1)
    legend = plt.legend([handles[idx] for idx in order], [labels[idx] for idx in order], ncol=1, fontsize=l_size,
                        title=legend_h, bbox_to_anchor=(0.042, 0.05), loc=3)
    plt.setp(legend.get_title(), fontsize=l_size)

    # get second y-axis
    if max1.shape[1] > 2:
        ax2 = ax1.twinx()
        ax2.plot(flood1[:, -1], flood1[:, 1].astype(float), color='maroon', linestyle='-.', marker='x',
                 linewidth=0, label=lab_mean2, alpha=0)
        plt.ylabel(ylabel2, fontsize=ax_size, labelpad=15)
        plt.yticks(np.arange(0, 4.5, 0.5), np.arange(0, 4.5, 0.5), fontsize=l_size)
        plt.ylim(-0.075, 3.83)
        # plt.legend([handles[idx] for idx in order], [labels[idx] for idx in order], ncol=1, fontsize=l_size,
        #                 title=legend_h, bbox_to_anchor=(0.042, 0.03), loc=3)
    plt.xlim(5.4, 6.7)
    plt.savefig(save2, dpi=300, bbox_inches='tight')
def doPlot_cross(crosssection, xlabel, ylabel, l_size, ax_size, title, save):
    '''sediment aggregation in four different downstream cross sections for analyze the remaining potential water
depth'''
    palette = np.array(['#e3ce8d', '#ffa584', '#db786c', '#e796d8', '#8e729d', '#7ba6d0', '#198c8c',
                        '#7ba47b', '#bc856c', '#8d8d8d', '#383838'])
    fig = plt.figure(figsize=(19, 12))
    # fig.suptitle(title, fontsize=24, fontweight=1, color='black').set_position([.5, 0.94])
    ax = fig.add_subplot(1, 1, 1)
    plt.plot(width, height1, color='black')
    plt.plot(width, height2, color='black')
    plt.plot(width, height3, color='black')
    plt.plot(width, height4, color='black')

    plt.text(44.5, height1[-1]+0.1, crosssection[0], color='black', fontsize=l_size)
    plt.text(44.5, height2[-1]+0.1, crosssection[1], color='black', fontsize=l_size)
    plt.text(44.5, height3[-1]+0.1, crosssection[2], color='black', fontsize=l_size)
    plt.text(44.5, height4[-1]+0.1, crosssection[3], color='black', fontsize=l_size)
    plt.text(0, height1[0]+0.1, crosssection[0], color='black', fontsize=l_size)
    plt.text(0, height2[0]+0.1, crosssection[1], color='black', fontsize=l_size)
    plt.text(0, height3[0]+0.1, crosssection[2], color='black', fontsize=l_size)
    plt.text(0, height4[0]+0.1, crosssection[3], color='black', fontsize=l_size)

    plt.axhline(fill_max1[0,1], xmin=0.2, xmax=0.8,color='#5d4535', linewidth=0.7, alpha=0.6)
    plt.axhline(fill_max1[5,1], xmin=0.22, xmax=0.78, color='#5d4535', linewidth=0.7, alpha=0.6)
    plt.axhline(fill_max1[10,1], xmin=0.24, xmax=0.76,color='#5d4535', linewidth=0.7, alpha=0.6)
    # plt.axhline(height1[0]-0.05, xmin=0.18, xmax=0.814,color='grey', linewidth=0.7, alpha=0.6)
    # plt.axhline(height2[0]-0.05, xmin=0.194, xmax=0.795,color='grey', linewidth=0.7, alpha=0.6)
    # plt.axhline(height3[0]-0.05, xmin=0.195, xmax=0.805,color='grey', linewidth=0.7, alpha=0.6)
    # plt.axhline(height4[0]-0.05, xmin=0.195, xmax=0.805,color='grey', linewidth=0.7, alpha=0.6)
    plt.text(16.55, fill_max1[0,1]-0.61, '2.7 m fill (0% maintenance effort)', alpha=1, color='#5d4535',
             fontsize=16, fontweight='bold')
    plt.text(16.5, fill_max1[5,1]-0.455, '1.6 m fill (50% maintenance effort)', alpha=1, color='#5d4535',
             fontsize=16, fontweight='bold')
    plt.text(16.455, fill_max1[10,1]-0.4555, '0.8 m fill (100% maintenance effort)', alpha=1, color='#5d4535',
             fontsize=16, fontweight='bold')

    plt.xlabel(xlabel, fontsize=ax_size, labelpad=10)
    plt.ylabel(ylabel, fontsize=ax_size, labelpad=10)
    # ax.yaxis.grid(linestyle='--', alpha=0.3)
    plt.yticks(fontsize=l_size)
    plt.xticks(fontsize=l_size)

    plt.savefig(save, dpi=300, bbox_inches='tight')
    print('cross sections plotted''\n')
def doPlot_cum_maint(scenario, cums, ylabel, title1, title2, save):
    '''create plot with 6 subplots for flood scenarios, each plot presenting the cumulative sediment yield over time'''
    # subplot for all flood scn
    maint = ['0%', '10%', '20%', '30%', '40%', '50%', '60%', '70%', '80%', '90%', '100%']
    floods = ['(a) 2 apart', '(b) 2 close', '(c) 2 med', '(d) 1 first', '(e) 1 middle', '(f) 1 last']
    palette = np.array(['#e3ce8d', '#ffa584', '#db786c', '#e796d8', '#8e729d', '#7ba6d0', '#198c8c',
                        '#7ba47b', '#bc856c', '#8d8d8d', '#383838'])
    palette2 = np.array(['#e3ce8d', '#db786c', '#8e729d', '#7ba6d0', '#7ba47b', '#8d8d8d'])
    palette3 = doColors(scenario[0]-3)

    fig = plt.figure(figsize=(19, 12))
    # fig.suptitle(title1, fontsize=24, fontweight=1, color='black').set_position([.5, 0.965])
    # fig.text(0.5, 0.91, title2, ha='center', fontsize=ax_size, style='italic')
    ax = plt.axes([0, 0, 1, 1], frameon=False)
    ax.axes.get_xaxis().set_visible(False)
    ax.axes.get_yaxis().set_visible(False)
    fig.text(0.52, 0.05, xlabel, ha='center', fontsize=ax_size)
    fig.text(0.006, 0.5, ylabel, va='center', rotation='vertical', fontsize=ax_size)

    for x in range(scenario[1]):
        ax = fig.add_subplot(3, 2, (x+1), sharey=ax)
        ax.text(.02, .9, floods[x], fontsize=l_size, color='black', transform=ax.transAxes)
        ax.text(.96, .75, 'Maintenance effort', fontsize=s_size, color=palette3[scenario[0]-6, x],
                transform=ax.transAxes, rotation=270)
        plt.subplots_adjust(left=0.075, bottom=0.1, right=0.99, top=0.99, wspace=0.17, hspace=0.02)
        for y in range(scenario[0]-3):
            ax.plot(cums[y, x, :], color=palette3[y, x], linestyle='-', linewidth=1)
            if y % 2 == 0:
                ax.text(100, cums[y, x, 99], maint[y], alpha=1, color=palette3[y, x], fontsize=s_size)
            else:
                ax.text(100, cums[y, x, 99], maint[y], alpha=1, color=palette3[y, x], fontsize=s_size)
            plt.xticks(np.arange(0, 110, 10), np.arange(0, 110, 10), fontsize=l_size)
            plt.yticks(np.arange(0, 600000, 75000), np.arange(0, 600000, 75000), fontsize=l_size)
            plt.xlim(-2, 113)
            plt.ylim(lim_range)
    plt.savefig(save, dpi=400, bbox_inches='tight', pad_inches=0)

    # individual plot for each flood scn, different colors for maintenance (flood colors)
    for x in range(scenario[1]):
        floods = ['(a) 2 apart', '(b) 2 close', '(c) 2 med', '(d) 1 first', '(e) 1 middle', '(f) 1 last']
        fig = plt.figure(figsize=(19, 12))
        ax = fig.add_subplot(1, 1, 1)
        ax.text(0.02, 0.95, floods[x], fontsize=ax_size, color='black', transform=ax.transAxes)
        for y in range(scenario[0]-3):
            ax.plot(cums[y, x, :], color=palette3[y, x], linestyle='-', linewidth=1)
            if y==scenario[0]-4:
                z=4
                ax.text(100, cums[scenario[0]-z, x, 99], maint[scenario[0]-z], alpha=1, color=palette3[10, x], font-
size=l_size)
                z=z+1
                ax.text(100, cums[scenario[0]-z, x, 99], maint[scenario[0]-z], alpha=1, color=palette3[9, x], font-
size=l_size)
                z=z+1
                ax.text(100, cums[scenario[0]-z, x, 99], maint[scenario[0]-z], alpha=1, color=palette3[8, x], font-
size=l_size)
                z=z+1
                ax.text(100, cums[scenario[0]-z, x, 99], maint[scenario[0]-z], alpha=1, color=palette3[7, x], font-
size=l_size)
                z=z+1
                ax.text(100, cums[scenario[0]-z, x, 99], maint[scenario[0]-z], alpha=1, color=palette3[6, x], font-
size=l_size)
                z=z+1
                ax.text(100, cums[scenario[0]-z, x, 99], maint[scenario[0]-z], alpha=1, color=palette3[5, x], font-
size=l_size)
                z=z+1
                ax.text(100, cums[scenario[0]-z, x, 99], maint[scenario[0]-z], alpha=1, color=palette3[4, x], font-
size=l_size)
                z=z+1
                ax.text(100, cums[scenario[0]-z, x, 99], maint[scenario[0]-z], alpha=1, color=palette3[3, x], font-
size=l_size)
                z=z+1
                ax.text(100, cums[scenario[0]-z, x, 99], maint[scenario[0]-z], alpha=1, color=palette3[2, x], font-
size=l_size)
                z=z+1
                ax.text(100, cums[scenario[0]-z, x, 99], maint[scenario[0]-z], alpha=1, color=palette3[1, x], font-
size=l_size)
                z=z+1
                ax.text(100, cums[scenario[0]-z, x, 99], maint[scenario[0]-z], alpha=1, color=palette3[0, x], font-
size=l_size)
        ax.yaxis.grid(linestyle='--', alpha=0.3)
        plt.ylabel(ylabel, fontsize=ax_size, labelpad=l_size)
        plt.xlabel(xlabel, fontsize=ax_size, labelpad=l_size)
        plt.yticks(fontsize=l_size)
        plt.xticks(np.arange(0, 110, 10), np.arange(0, 110, 10), fontsize=l_size)
        plt.ylim(lim_range)
        plt.xlim(-3, 106)
        floods = ['2 apart', '2 close', '2 med', '1 first', '1 middle', '1 last']
        plt.savefig('U:simulations/analysis/python/sed  yield/CumSumIndplot'+floods[x]+'_diffcol2_maint.png',
dpi=300,
                    bbox_inches='tight')
def doPlot_cum_loc(scenario, cums, ylabel, title1, title2, save):
    '''create plot with 6 subplots for flood scenarios, each plot presenting the cumulative sediment yield over time'''
    # subplot for all flood scn
    maint = ['High', 'Low', 'Mid']
    floods = ['(a) 2 apart', '(b) 2 close', '(c) 2 med', '(d) 1 first', '(e) 1 middle', '(f) 1 last']
    palette = np.array(['#e3ce8d', '#ffa584', '#db786c', '#e796d8', '#8e729d', '#7ba6d0', '#198c8c',
                      '#7ba47b', '#bc856c', '#8d8d8d', '#383838'])
    palette2 = np.array(['#e3ce8d', '#db786c', '#8e729d', '#7ba6d0', '#7ba47b', '#8d8d8d'])
    palette3 = doColors(scenario[0]-11)

    fig = plt.figure(figsize=(19, 12))
    # fig.suptitle(title1, fontsize=24, fontweight=1, color='black').set_position([.5, 0.965])
    ax = plt.axes([0, 0, 1, 1], frameon=False)
    ax.axes.get_xaxis().set_visible(False)
    ax.axes.get_yaxis().set_visible(False)
    fig.text(0.52, 0.05, xlabel, ha='center', fontsize=ax_size)
    fig.text(0.006, 0.5, ylabel, va='center', rotation='vertical', fontsize=ax_size)
    # fig.text(0.5, 0.91, title2, ha='center', fontsize=ax_size, style='italic')

    for x in range(scenario[1]):
        ax = fig.add_subplot(3, 2, (x+1), sharey=ax)
        ax.text(0.02, .9, floods[x], fontsize=l_size, color='black', transform=ax.transAxes)
        ax.text(.96, .9, 'Maintenance location', fontsize=s_size, color=palette3[scenario[0]-11, x],
                transform=ax.transAxes, rotation=270)
        plt.subplots_adjust(left=0.075, bottom=0.1, right=0.99, top=0.99, wspace=0.17, hspace=0.02)
        for y in range(scenario[0]):
            ax.plot(cums[y, x, :], color=palette3[y*2, x], linestyle='-', linewidth=1)
            ax.text(100, cums[y, x, 99], maint[y], alpha=1, color=palette3[y*2, x], fontsize=s_size)
            plt.xlim(-2, 113)
            plt.ylim(lim_range)
            plt.xticks(np.arange(0, 110, 10), np.arange(0, 110, 10), fontsize=l_size)
        plt.yticks(np.arange(0, 600000, 75000), np.arange(0, 600000, 75000), fontsize=l_size)
    plt.savefig(save, dpi=400, bbox_inches='tight')

# -DEFINE GLOBAL VARIABLES HERE-
# font sizes for plot
ax_size = 18        # axis label
l_size = 16         # tick size, legend size
s_size = 12         # small in plot labels

# -READ IN FILES HERE-
# create list with paths from where to read in the files
loc = np.array(glob.glob('U:simulations/dem_reach/****/***/**/*.dat'))
main = np.array(glob.glob('U:simulations/dem_reach/***/**/*.dat'))
files = np.append(main, loc)

fill_mean = ('U:simulations/analysis/python/sed yield/mean_sedyield.csv')
fill_all = ('U:simulations/analysis/python/sed yield/all_sedyield.csv')
sum_sed = ('U:simulations/analysis/python/sed yield/sum.csv')

fill_mean = np.genfromtxt(fill_mean, delimiter=',', skip_header=1)  # mean of flood scenarios for downstream chan-
nel fill
fill_all = np.genfromtxt(fill_all, delimiter=',', skip_header=1)    # seperate flood scenarios for downstream channel
fill
sum_sed = np.genfromtxt(sum_sed, delimiter=',', skip_header=1)      # sum of total sed yield (minus offset from
spinoff part)

# -CALL FUNCTIONS HERE-
# ---------------------- 1 calculate sum of total sediment ----------------------
# read in water and sediment outputs
dat_data = doRead(files)

# calculate sum of sediment yield (2nd column) over 100 years (all rows) for all scenarios
sumsed = doSumSed(dat_data)

# WORK IN EXCEL: export 'sumsed' and subtract the sediment yield offset (model spinoff part (calculated in
ArcGIS) from
# it. also calculate the potential fill in percentage and in meters. this is all done in excel. in a next step read in
# this newly calculated sum of the total sediment yield with its potential filling value ('sum_sed').

## ---------------------- 2.1 channel fill ----------------------
# create array for sorting and splitting of the data
fill_all1, fill_all2, fill_mean1, fill_mean2 = doArray(fill_all[:, (-2,-1)], fill_mean[:, (-2,-1)])

# calculate the min and the max of the flood scenarios
fill_min1, fill_min2, fill_max1, fill_max2 = doMinMax(fill_all[:,(-2,-1)])

# plot maint and loc scenarios combined
lim_range = -2, 102
titel_fill = "Downstream channel fill after 100 years of simulation"
xlabel = "                                   Maintenance effort [%]" \
      "                                                  Location"
ylabel_fill1 = "Channel fill [%]"
ylabel_fill2 = "Channel fill [m]"
lab_mean = "Mean of hydrographs  for channel fill [%]"
lab_mean2 = "Mean of hydrographs for channel fill [m]"
lab_range = "Range of hydrographs for channel fill [%]"
lab_range2 = "Range of hydrographs for channel fill [m]"
label_30 = "30 % maintenance"
legend_h = "Hydrograph"
save_fill1 = 'U:simulations/analysis/python/sed yield/FillAll_maint+loc.png'
save_fill2 = 'U:simulations/analysis/python/sed yield/FillRange_maint+loc.png'

doPlot(fill_all1.shape[0], fill_all1, fill_all2, fill_mean1, fill_mean2, fill_min1, fill_min2,
       fill_max1, fill_max2, '#7ba6d0', xlabel, ylabel_fill1, ylabel_fill2,
       titel_fill, save_fill1, save_fill2)

# plot maint and loc scenarios separately
# maint scn
```

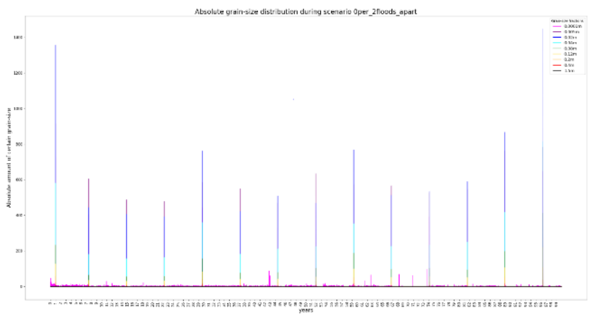
```
xlabel = "Maintenance effort [%]"
save_fill1 = 'U:simulations/analysis/python/sed yield/FillAll_maint.png'
save_fill2 = 'U:simulations/analysis/python/sed yield/FillRange_maint.png'
doPlot_maint(fill_all1.shape[0], fill_all1, fill_mean1, fill_min1, fill_max1, '#7ba6d0', xlabel, ylabel_fill1,
        ylabel_fill2, titel_fill, save_fill1, save_fill2)
# loc scn
z = -4        # corresponds to the 30 % maintenance value, which is unfortunately not always at the same position...
xlabel = "Maintenance location"
save_fill3 = 'U:simulations/analysis/python/sed yield/FillAll_loc.png'
save_fill4 = 'U:simulations/analysis/python/sed yield/FillRange_loc.png'
doPlot_loc(fill_all1.shape[0], fill_all1, fill_all2, fill_mean1, fill_mean2, fill_min1, fill_min2, fill_max1, fill_max2,
       '#7ba6d0', xlabel, ylabel_fill1, ylabel_fill2, titel_fill, save_fill3, save_fill4)

## ---------------------- 2.2 cross section ----------------------
# generate cross sections with spatial dimension values derived from geo.map.admin
width = np.arange(50)
height1                                        =                                        np.ar-
ray([4.25,4.25,4.25,4.25,4.25,4.25,4.25,4.25,4,3,2,1,0.5,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
            0.5,1,2,3,4,4.5,4.5,4.5,4.5,4.5,4.5,4.5,4.5])
height2                                        =                                        np.ar-
ray([5.5,5.5,5.5,5.5,5.5,5.5,5.5,5.5,5.5,5,4,3,2,1,0.5,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0.5,1,2,3,
            4,5,6,6,6,6,6,6,6,6])
height3                                        =                                        np.ar-
ray([5,5,5,5,5,5,5,5,5,4,3,2,1,0.5,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0.5,1,2,3,4,5,5.5,5.5,5.5,
            5.5,5.5,5.5,5.5,5.5])
height4                                        =                                        np.ar-
ray([3,3,3,3,3,3,3,3,3,2,1,0.5,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0.5,1,2,3,4,4,4,4,4,4,
            4,4])

# plot cross section
# define plot properties
title = 'Cross sections in downstream channel'
crosssection = ['Cross section 1', 'Cross section 2', 'Cross section 3', 'Cross section 4']
xlabel = 'Width [m]'
ylabel = 'Height [m]'
save = 'U:simulations/analysis/python/sed yield/Crossection.png'
doPlot_cross(crosssection, xlabel, ylabel, l_size, ax_size, title, save)

## ---------------------- 3 sed yield ----------------------
# create array for sorting and splitting of the data
sum_sed1, sum_sed2, fill_mean1, fill_mean2 = doArray(sum_sed[:, 1], fill_mean[:, (-2, -1)])

# calculate the min and the max of the flood scenarios
sum_min1, sum_min2, sum_max1, sum_max2 = doMinMax(sum_sed[:, 1])

# calculate the mean of the flood scenarios
sum_mean1, sum_mean2 = doFloodmean(sum_sed[:, 1])

# plot maint and loc scenarios combined
# define plot properties
lim_range = -10000, 510000
titel_sedsum = "Total sediment yield after 100 years of simulation"
xlabel = "                              Maintenance effort [%]" \
     "                              Location"
ylabel_sedsum = "Sum of sediment yield [m\u00b3]"
save_sedsum1 = 'U:simulations/analysis/python/sed yield/SedSum_maint+loc.png'
save_sedsum2 = 'U:simulations/analysis/python/sed yield/SedSumRange_maint+loc.png'

doPlot(sum_sed1.shape[0], sum_sed1, sum_sed2, sum_mean1, sum_mean2, sum_min1, sum_min2, sum_max1,
sum_max2, '#5C9C88',
    xlabel, ylabel_sedsum, ylabel_fill2, titel_sedsum,save_sedsum1, save_sedsum2)

# plot maint and loc scenarios separately
# maint scn
xlabel = "Maintenance effort [%]"
save_sedsum1 = 'U:simulations/analysis/python/sed yield/SedSum_maint.png'
save_sedsum2 = 'U:simulations/analysis/python/sed yield/SedSumRange_maint.png'
```

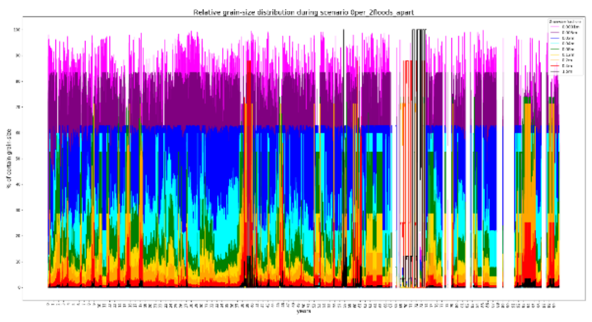
```
doPlot_maint(sum_sed1.shape[0], sum_sed1, sum_mean1, sum_min1, sum_max1, '#5C9C88', xlabel, yla-
bel_sedsum, ylabel_fill2,
            titel_sedsum, save_sedsum1, save_sedsum2)

# loc scn
z = 3
xlabel = "Maintenance location"
save_sedsum3 = 'U:simulations/analysis/python/sed yield/SedSum_loc.png'
save_sedsum4 = 'U:simulations/analysis/python/sed yield/SedSumRange_loc.png'
doPlot_loc(sum_sed1.shape[0], sum_sed1, sum_sed2, sum_mean1, sum_mean2, sum_min1, sum_min2,
sum_max1, sum_max2,
          '#5C9C88', xlabel, ylabel_sedsum, ylabel_fill2, titel_sedsum, save_sedsum3, save_sedsum4)

## ---------------------- 4 cum sum ----------------------
# calculate cumsum of four different maintenance scenarios (0%, 30%, 70%, 100%)
cumsum = doCumsum(dat_data)

# calculate the min and the max of the flood scenarios
cums_off = doSplit(cumsum)

# plot cumsum
# define plot properties
title_cum = "Cumulative sediment yield during 100 years of simulation "
title_maint = "Maintenance effort"
xlabel = "Time [years]"
ylabel_cum = "Cumulative sediment yield [m\u00b3]"
lim_range = -15000, 486000
save_cum = 'U:simulations/analysis/python/sed yield/CumSumSubplot_maint.png'
doPlot_cum_maint(cums_off.shape, cums_off, ylabel_cum, title_cum, title_maint, save_cum)


cums_off = cums_off[range(11, 14)]

title_loc = "Maintenance location"
save_cum = 'U:simulations/analysis/python/sed yield/CumSumSubplot_loc.png'
doPlot_cum_loc(cums_off.shape, cums_off, ylabel_cum, title_cum, title_loc, save_cum)
```

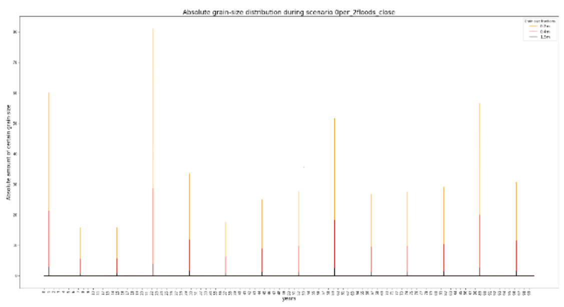
3.5 Tool 5: Grain-size distribution

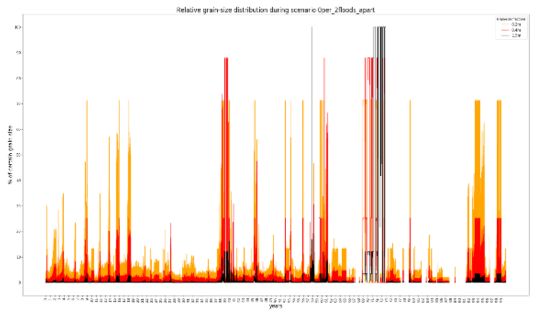
Grain-size distribution: this script deals with the different grain-size distributions generated at the output of the model. In total there are 9 different grain-size fractions. This script analyzes the absolute amount of every grain-size fractions and the relative distribution. Additionally, it calculates for every grain-size column the highest value (for the absolute and relative value) per simulation.

The script produces following figures:









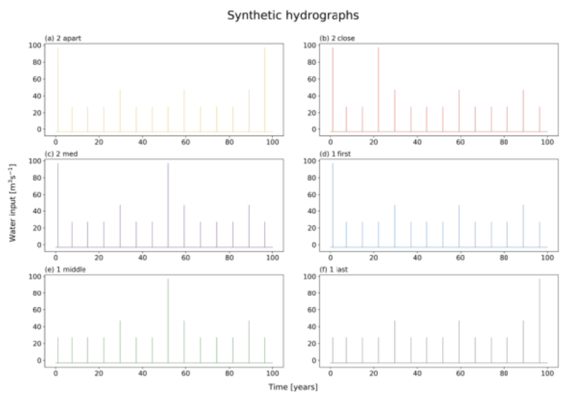
#GRAIN –SIZE DISTRIBUTION: this script dealls with the #sediment distribution (9 classes of grain-size) generated at the output of the model. The absolute and relative #values are displayed over a period of 100 years. Due to the complexity single grain sizes categories can be #considered. Additionally, the script calculates the highest values of each grain-size category of each scenario

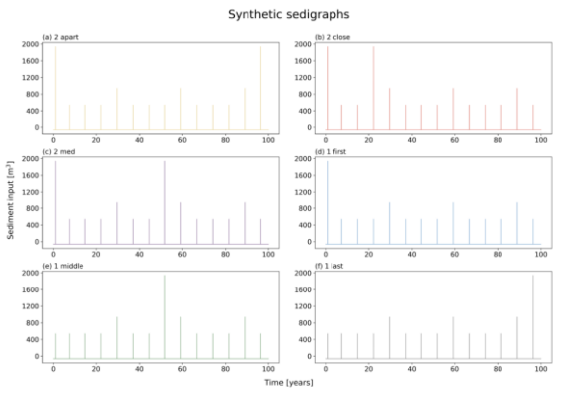
```
import numpy as np
import glob, os
import numpy as np
import matplotlib.pyplot as plt

load_max_files = 66
year = 8760
max_rows = year*100  # 8760 -> 1y; 17520 -> 2y;  35040 -> 4y
load_cols = range(4, 14)
skiprows = 2 # need to skip rows otherwise computer crashes (memory error)
use_cols = range(0, 9)
max_rows = max_rows / skiprows
# declare custom objects
# creating a template to store the grainsize data sets. The template has a parameter for name of the set and its data.
class GrainsizeSet(object):
    name = ""
    data = []
# create template to store information about grainsize name, scenario and the max value
class Grainsize(object):
    name = ""
    scenario = ""
    max_grainsize = 0
labels = []
class Label(object):
    name = ""
    color = ""
# declare global variables
selected_files = []
index = 0
# declare a list to store all gainsize related max values
grainsizes = []
# chart style
defaultFontSize = 16
headerFontSize = 20
numberFontSize = 11
legendFontSize = 12
# define labels and colors for each column
label = Label()
label.name = "0.0001m"
label.color = "fuchsia"
labels.append(label)
label = Label()
label.name = "0.005m"
label.color = "purple"
labels.append(label)
label = Label()
label.name = "0.02m"
label.color = "blue"
labels.append(label)
label = Label()
label.name = "0.04m"
label.color = "cyan"
labels.append(label)
label = Label()
label.name = "0.08m"
label.color = "green"
labels.append(label)
label = Label()
label.name = "0.12m"
label.color = "gold"
labels.append(label)
label = Label()
label.name = "0.2m"
label.color = "orange"
labels.append(label)
label = Label()
label.name = "0.4m"
label.color = "red"
labels.append(label)
label = Label()
label.name = "1.5m"
label.color = "black"
labels.append(label)
# Look for dat files
for root, dirs, files in os.walk("C:\LocalDrive\Seminar_DMA2\input"):
    # Get data for each file
    for file in files:
        if file.endswith(".dat") and (index < load_max_files):
            # increment index in order to compare with load max files setttings
            index += 1
            # get filename
            filename = os.path.join(root, file)
            print("Loading: " + filename)
            # load data
            data = np.genfromtxt(filename, delimiter=' ', max_rows=max_rows*skiprows, usecols=load_cols)
            # filter data set
            data = data[0::skiprows]
            # create new grainsize set
            grainsize_set = GrainsizeSet()
            grainsize_set.name = os.path.basename(file)
            print ("Processing: " + grainsize_set.name)
            # load one column
            column = np.array(data)
            first_row_data = data[:,0]
            # calculate percentage for all grainsize columns compared to the total column
            calculated = column / first_row_data[:,None] * 100
            # declare figure for plot charts
            fig, ax = plt.subplots()
            fig.set_size_inches(30.5, 15.5)
            # chart specific settings
            title = 'Absolute grain-size distribution during scenario ' + grainsize_set.name[:-4]
            # create new plot with absolute data (not calculated data=relative data)
            x = np.arange(max_rows)
            plt.ylabel('Absolute amount of certain grain-size', fontsize=defaultFontSize)
            plt.yticks(fontsize=numberFontSize)
            plt.xlabel('years', fontsize=defaultFontSize)
            plt.title(title, fontsize=headerFontSize)
            plt.xticks(np.arange(0, max_rows, step=year/skiprows), np.arange(0, max_rows / (year/skiprows)), font-
size= numberFontSize, rotation ='vertical')
            #plt.xticks(fontsize = labelFontsize, position = 'vertical')
            # add data
            for count in use_cols:
                # skip the first column with total values
                y = data[:, count+1]
                ax.plot(x, y, color=labels[count].color, label=labels[count].name)
            # add legend
            ax.legend(loc=1, fontsize = legendFontSize, title= "Grain-size fractions")
            # save as image
            plt.savefig(os.path.join("C:\LocalDrive\Seminar_DMA2\output", "absolute-" + grainsize_set.name +
    ".png"))
            # declare figure for plot charts
            fig, ax = plt.subplots()
            fig.set_size_inches(30.5, 15.5)
            # chart specific settings
            title = 'Relative grain-size distribution during scenario ' + grainsize_set.name[:-4]
            # create new plot with relative data (calculated data)
            x = np.arange(max_rows)
            plt.ylabel('% of certain grain-size', fontsize=defaultFontSize)
            plt.yticks(np.arange(0, 101, 10), fontsize = numberFontSize)
            plt.xlabel('years', fontsize=defaultFontSize)
            plt.title(title, fontsize=headerFontSize)
            plt.xticks(np.arange(0, max_rows, step=year / skiprows), np.arange(0, max_rows / (year / skiprows)), font-
    size= numberFontSize, rotation ='vertical')
            # add data
            for count in use_cols:
                # skip the first column with total values
                y = calculated[:, count+1]
                ax.plot(x, y, color=labels[count].color, label=labels[count].name)
            # add legend
            ax.legend(loc=1, fontsize= legendFontSize, title= "Grain-size fractions")
            # save as image
            plt.savefig(os.path.join("C:\LocalDrive\Seminar_DMA2\output", "relative-" + grainsize_set.name + ".png"))
            plt.close('all')
            # get max value for each grainsize column
            for count in use_cols:
                grainsize = Grainsize()
                grainsize.name = labels[count-1].name
                grainsize.scenario = grainsize_set.name
                grainsize.max_grainsize = np.max(data[:, count])
                grainsizes.append(grainsize)
    print("finished")
```

3.6 Tool 6: Comparison and synthetic input

Output comparison + synthetic input: This script includes two additional tools. The first one is comparing two different simulation runs, to check whether the model can reproduce twice similar answers. The second part is producing bar plots for the different synthtetic water and sediment input.

The following script produces these figures:





```
# OUTPUT COMPARISON + SYNTHETIC INPUT: this script includes two additional tools.
# The first one is comparing two different simulation runs, to check whether the
# model can reproduce twice similar answers. the second part is producing the syn
# thetic water input data for the simulation.

# -IMPORT LIBRARIES & VARIABLES HERE-
import numpy as np
import glob
import matplotlib.pyplot as plt

# -DEFINE FUNCTIONS HERE-
def doRead():
    '''read in output results model runs'''
    dat_data = []
    for x in range(files.shape[0]):
        for y in range(files.shape[1]):
            read_files = np.genfromtxt(files[x][y], delimiter=' ', usecols=(1, 4), skip_header=3) # only read in Qw & Qs
            dat_data.append(read_files)
        print('read in '+str(x)+' of ' +str(files.shape[0]))
    dat_data = np.array(dat_data)
    print('read\'em all in''\n')

    return dat_data
def doSum():
    '''calculate sum of sediment yield for all scenarios'''
    sum = []
    for x in range(dat_data.shape[0]):
        sum_Qs = np.sum(dat_data[x, :, 1])
        sum.append(sum_Qs)
    sum = np.array(sum).reshape(6, 2)       # reshape it two array with 6 rows and 2 cols
    return sum
def doDiff():
    '''calculate the difference of sed yield between the tow simulation rusn'''
    diff_perc = []
    for x in range(6):
        diff = (sum[x, 0]-sum[x, 1])/sum[x, 0]*100
        diff_perc.append(diff)
    diff_perc = np.array(diff_perc).reshape(6, 1)
    return diff_perc
def doHydro(time,n, z):
    '''fill the created hydrograph with any number (n) and repeat the number for a certain number of times (z)'''
    hydro = []
    for x in range(len(flood)):
        discharge = np.repeat(n, z)
        np.put(discharge, time[x][:], flood[x][:])
        hydro.append(discharge)
    hydro = np.array(hydro)
    return hydro
def doPlot(xlabel, ylabel, ytick1, ytick2, ax_size, l_size, title, save):
    '''plot the hydro- or sedigraph'''
    floods = ['(a) 2 apart', '(b) 2 close', '(c) 2 med', '(d) 1 first', '(e) 1 middle', '(f) 1 last']
    palette2 = np.array(['#e3ce8d', '#db786c', '#8e729d', '#7ba6d0', '#7ba47b', '#8d8d8d'])
    fig = plt.figure(figsize=(19, 12))
    fig.suptitle(title, fontsize=24, fontweight=1, color='black').set_position([.5, 0.96])
    ax = plt.axes([0, 0, 1, 1], frameon=False)
    ax.axes.get_xaxis().set_visible(False)
    ax.axes.get_yaxis().set_visible(False)
    fig.text(0.5, 0.055, xlabel, ha='center', fontsize=ax_size)
    fig.text(0.07, 0.5, ylabel, va='center', rotation='vertical', fontsize=ax_size)

    for x in range(len(floods)):
        ax = fig.add_subplot(3, 2, (x+1), sharey=ax)
        plt.title(floods[x], fontsize=l_size, loc='left')
        plt.subplots_adjust(wspace=0.15, hspace=0.25)
        ax.plot(years, hydro[x, :], color=palette2[x], linestyle='-', linewidth=1, label='_nolegend_')
        plt.xticks(range(0, 876100, (87600*2)), range(0, 110, 20), fontsize=l_size)
        plt.yticks(ytick1, ytick2, fontsize=l_size)
    plt.savefig(save, dpi=450, bbox_inches='tight')

# -READ IN FILES HERE-
files1 = np.array(glob.glob('U:simulations/2nd try/**/*.dat'))
```

```
files2 = np.array(glob.glob('U:simulations/3rd try/**/*.dat'))
files = np.append(np.vstack(files1), np.vstack(files2), axis=1)

# -CALL FUNCTIONS HERE-
# ---------------------- 1 output comparison ----------------------
# this script is comparing 6 scenarios from two different simulation runs
# read in water and sediment output
dat_data = doRead()

# calculate sum of sediment yield (2nd column) over 100 years (all rows) for all scenarios
sum = doSum()

# calculate difference of sed yield between the two runs
diff_perc = doDiff()

# combine two cols of sum plus the diff_perc col in one array, round everything to one decimal
comparison = np.round(np.append(sum, diff_perc, axis=1), 1)

# export file
np.savetxt('U:simulations/analysis/python/run_comparison.txt', comparison[:], delimiter=' ', comments='')

# ---------------------- 2 synthetic input ----------------------
# create synthetic hydrograph and sedigraph
# flood magnitude
apart = np.array([100, 30, 30, 30, 50, 30, 30, 30, 50, 30, 30, 30, 50, 100])
close = np.array([100, 30, 30, 100, 50, 30, 30, 30, 50, 30, 30, 30, 50, 30])
med = np.array([100, 30, 30, 30, 50, 30, 30, 100, 50, 30, 30, 30, 50, 30])
first = np.array([100, 30, 30, 30, 50, 30, 30, 30, 50, 30, 30, 30, 50, 30])
middle = np.array([30, 30, 30, 30, 50, 30, 30, 100, 50, 30, 30, 30, 50, 30])
last = np.array([30, 30, 30, 30, 50, 30, 30, 30, 50, 30, 30, 30, 50, 100])
flood = np.ndarray.tolist(np.concatenate([[apart], [close], [med], [first], [middle], [last]]))

# flood times for each flood scenario
at = np.array([8746, 64985, 129935, 194885, 259835, 324785, 389735, 454661, 519611, 584561,
              649511, 714461, 779411, 844361])
ct = np.array([8746, 64985, 129935, 194909, 259859, 324809, 389759, 454685, 519635, 584585,
              649535, 714485, 779435, 844385])
met = np.array([8746, 64985, 129935, 194885, 259835, 324785, 389735, 454661, 519635, 584585,
               649535, 714485, 779435, 844385])
ft = np.array([8746, 64985, 129935, 194885, 259835, 324785, 389735, 454661, 519611, 584561,
              649511, 714461, 779411, 844361])
mit = np.array([8745, 64961, 129911, 194861, 259811, 324761, 389711, 454637, 519587, 584537,
               649487, 714437, 779387, 844337])
l = np.array([8745, 64961, 129911, 194861, 259811, 324761, 389711, 454661, 519611, 584561,
             649511, 714461, 779411, 844361])
time = np.ndarray.tolist(np.concatenate([[at], [ct], [met], [ft], [mit], [l]]))

# fill hydrographs between the flood events with zeros
hydro = doHydro(time, 0, 876000)

# plot hydrographs
years = np.arange(0, 876000, 1)            # x value for plotting
years2 = np.arange(0, 100, 1)
# define plot properties
xlabel = "Time [years]"
ylabel = "Water input [$\mathregular{m^3 s^{-1}}$]"
ytick1 = range(3, 120, 20)
ytick2 = range(0, 120, 20)
ax_size = 16
l_size = 14
title = "Synthetic hydrographs"
save = 'U:simulations/analysis/python/bonus/hydrograph.png'
doPlot(xlabel, ylabel, ytick1, ytick2, ax_size, l_size, title, save)

# plot sedigraphs
# define plot properties
ylabel = "Sediment input [$\mathregular{m^3}$]"
ytick1 = range(3, 120, 20)
```

```
ytick2 = range(0, 2400, 400)
title = "Synthetic sedigraphs"
save = 'U:simulations/analysis/python/bonus/sedigraph.png'
doPlot(xlabel, ylabel, ytick1, ytick2, ax_size, l_size, title, save)
```