WASAoutputSimilarityTest

Introduction

Some tests have been made on WASA-SED. To do these tests, we study 16 cases that were detailed earlier and described in the table ***test_cases_description***.

Now, the objective is to check the consistance of WASA-SED. How de we proceed? For one case we will get two output folders with two different code versions. We will compare the files by pairs with the same name and check if there is any difference between them. We impose an error tolerance, a threshold, under which we consider that there is no error. In order to do this we write a Python code named **WASAoutputSimilarityTest**.

Files organization

In the code, we consider the organization for the Output folders presented by figure 1.

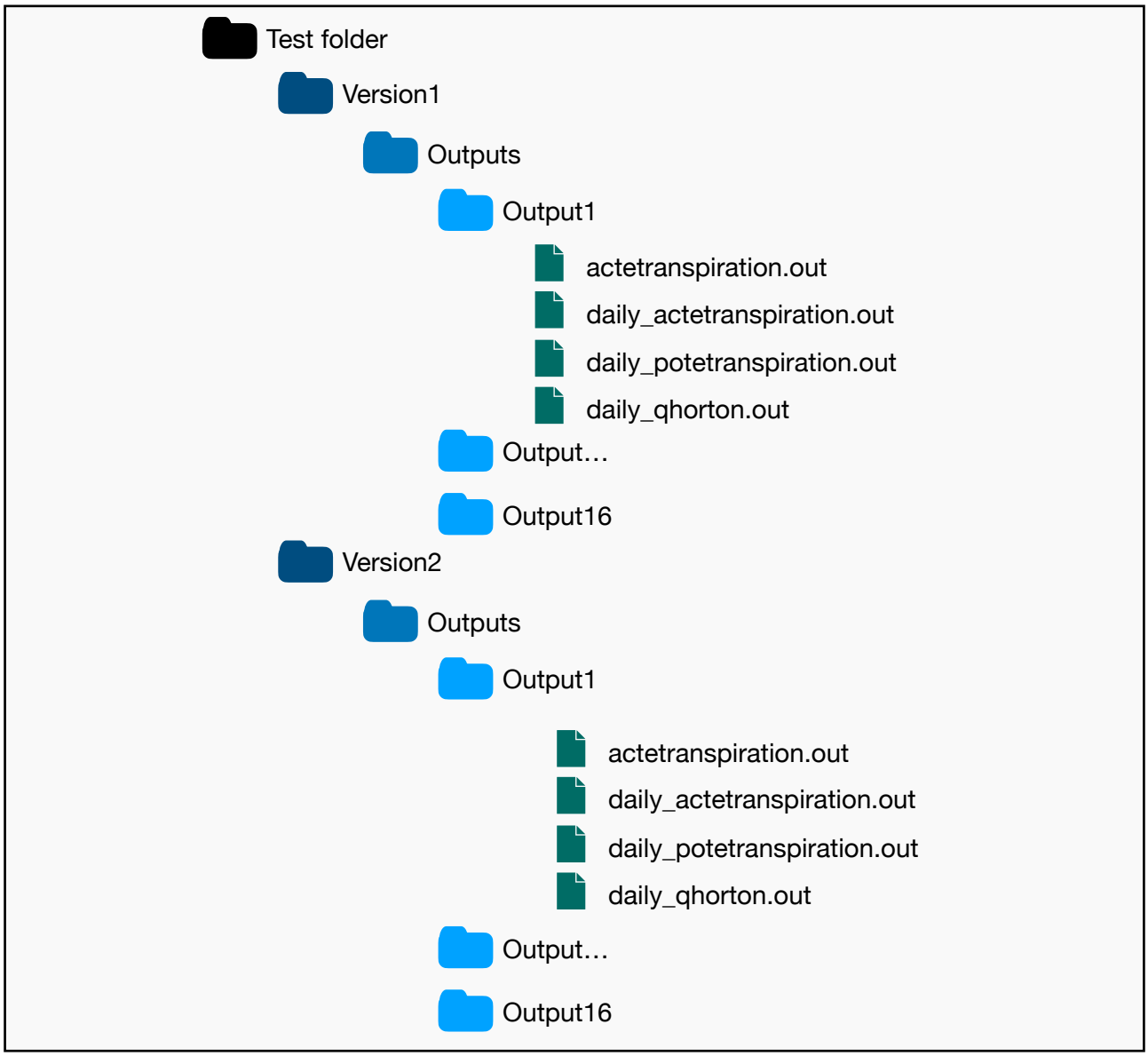

Figure 1: *Files and folders organization to use the code*

The folder organization is important into the code. If this one is not followed the actual code will not work. But, this is easy to change this part into the code. So, if in the future we decide to change the path of the files it won't be difficult to modify the code.

In this situation the two versions of the code we want to test are called « version 1 » and « version 2 » and are contained into a general folder names **test_folder**. For each version we have a folder named **Outputs** which contains the output folders for each case tested. So there is 16 output folders in which there are the files we want to compare. We assume that there is exactly the same files for one case into the **version1** and **version2** folders. For instance *\Version1\Outputs\Output1* contains the same files (same number and same names) as *\Version2\Outputs\Output1*. Afterwards if we notice that this is not the case we can add a function into the code to test that.

Code description

1. General

The aim of this code is to return if there is an error or not. We can use it by three ways that we will describe :

- **Option 1:** it returns for each case *OK* if all the files into the case are identical. If they are not, it returns *ERROR* and specifies which files are different.
- **Option 2:** it returns for each case *OK* if all the files into the case are identical. If they are not, it returns *ERROR* and specifies which files are different with the maximum error found into the file between two data.
- **Option 3:** we add a threshold for the error. If the calculated error is under the threshold we consider the files identical. The returns are the same as in option 2 but the maximum error printed will alway be above the threshold.

The most interesting way is the option 3. This is the most complete option.

The code is organized into a first part with the functions definition and an other part with the calculations.

Before continuing, we precise the python modules we use :

```
import os
import pandas as pd
```

2. Functions description

In this part we will list and then describe the role of the functions.

The functions are :

- test_path
- create_df_to_compare
- simple_comparison
- comparisonTF
- error_calculation
- comparison_threshold

➡ test_path :

This function creates two paths to have access to one case of study for the two versions we compare.

- Entry : **folder_version1** and **folder_version2**, strings with the names of the folders corresponding to the versions we want to test (e.g. version1 and version2) and **case_nb**, an integer which is the number of the case that we test (from 1 until 16).
- Return : **path1**, a string corresponding to the path which permits to go into the output folder of the case we are interested in for the first version and **path2**, a string corresponding to the path which permits to go into the output folder of the case we are interested in for the second version.

➡ create_df_to_compare :

From a path to a case output folder, the function first creates a list of all the file names contained into the folder. Then, it sorts out the files to keep only the ones which end with *.out* (we don't keep the *.stat* and *.stat_start* files). After this first step, we read the files with the *.out* extension and we create a data-frame for each. Then we get a list of data-frame and a list of names corresponding to the data-frame.

- Entry : **path**, a string corresponding to the path of a case output folder (e.g. '/Users/alicelegendre/Desktop/WASA/wasa_tests/WASA_pyProject/test_folder/version1/Outputs/Output1' for me if I want to go into the first case of the first version).
- Return : **file_names**, a list of the file names we test for the case considered and **df_list**, the list of the data-frame corresponding to the files.

Note: if **L** is the list of data-frames and **L_names** the list of the file names, **L_names[k]** is the name of the data_frame **L[k]**.

➡ simple_comparison :

From two data-frames it compares if they are identical or not.

- Entry : **df1** and **df2**, two data-frames we want to compare.
- Return : **comp**, a boolean. **True** means that the two data-frames are identical and **False** means that they are different.

➡ comparisonTF :

From two list of data-frames it uses the function **simple_comparison** to compare the data-frames by pairs. If the two data-frames are identical, so **simple_comparison** gives **True**, we write 'OK' and if it gives **False**, we keep the name of the corresponding file. At the end we will get a list with just one element 'OK' or with one or several names of file.

- Entry : **namelist1** and **dflist1**, two lists one with the names and the other with the corresponding data-frames. **namelist2** and **dflist2**, two lists one with the names and the other with the corresponding data-frames but from an other version.
- Return : **res**, a list which contains the result of the simple comparison ('OK' or some file names).

Note: we use this function only for the option 1 of the code.

➡ error_calculation :

From two values it calculates the error between them. We write the error with 3 decimal places.

- Entry : **v1** and **v2**, float, two values we want to compare.
- Return : **error**, a float with 3 decimal places.

Note: we decided to calculate the error through one method but we are open to use others.

➡ **comparisonBIG :**

From two lists of data-frames and the associated file names, it calculates the maximum error there is into a file (go through a version1/case1 file and compare line by line the data with a version2/case1 file, calculate the error for each line and then take the maximum of all these errors). The first step consists in an initial sorting. We use **simple_comparison** to check, at the first place, if the files are identical or not. If they are we do not need to compare them more precisely. So now we just have the files which are different and that we need to compare. Then we compare the data-frames which are different two by two. For one comparison we get a new data-frame with the data that differs from a data-frame to another. After that, we convert this resulting data-frame into a list to have access to the data. Then we calculate the error with **error_calculation** for each line into the resulting data-frame (the one containing only the lines of the file which are different between the two data-frames we compare). We take the maximum of all the errors of one file. Finally, we return the name of the file with the maximum error associated.

- Entry : **namelist1** and **dflist1**, lists, one with the names and the other with the corresponding data-frames. **namelist2** and **dflist2**, lists, one with the names and the other with the corresponding data-frames but from an other version.
- Return : **final_result**, a list which contains several lists with two elements : a string (file name) and a float (error).

Exemple: final_result = [['daily_actetranspiration.out', 0.026], ['daily_potetranspiration.out', 0.021], ['daily_subsurface_runoff.out', 1.64]]

3. Calculation part

This section will present the calculation part for the option 3 of the code **WASAoutputSimilarityTest**. This is shown by figure 2. It takes place in three steps that we will describe.

- **Data definition :** Defining the data is the first step of the code. We use 'new' for version1 and 'old' for version 2. We can note that the counter **c** will be useful in the second loop to write neatly the results into the console or into a text file. Indeed, at the end the results will be written into a text file named ***test_results2.txt***.
- **Loop 1 :** The aim of this loop is to get the result of the comparison between the versions of the code WASA-SED. The index **k** takes the cases number. In figure 2 we can see that we study the cases 11, 12 and 13 (because the number of cases we have specified in the data definition part is 3). For one case **k**, we recover the path of the output folders, we create a list of data-frames (one data-frame per file) and we use **comparisonBIG** to get a list of 2-elements-lists containing file names and the associated error named **result**. With the element **q**, we go through the list **result** to compare calculated errors with the threshold thanks to the function **comparison_threshold**. If the calculated error is under the threshold we consider the two files identical and so we delete the list containing the name of the file and the error from the list **result**. Then, we create a new list, **general_results**, which will contain the result for each case studied. If the list **result** is empty, that means, for the case considered, that all the output files are identical so there is no problem and we write 'OK' into **general_results**. Otherwise, if **result** is not empty, we append it into **general_results**. Finally, at the end of the loop we get a list, **general_results**, which inform us on the state of each case of study.

Exemple: general_results = **[[** ['daily_actetranspiration.out', 0.026], ['daily_subsurface_runoff.out', 1.64] **]**, **[** ['daily_actetranspiration.out', 0.026], ['daily_subsurface_runoff.out', 1.64] **]**, 'OK'**]**

- **Loop 2 :** The goal of this loop is to write neatly the results. The element **l** takes values into **general_results**. That means **l** is a list of lists containing the file names and the error associated for one case. This loop write the result into the form: 'Case i : res' with i the number of the case and res is 'OK' if there is no error, else res is 'ERROR' followed by a list containing the name of the file in which the error is with the associated error (this error is necessarily higher than the threshold). We write the results into the console and into the text file at the same time.

  Exemple:
  Case 1 : ERROR => [['daily_actetranspiration.out', 0.026], ['daily_subsurface_runoff.out', 1.64]]
  Case 2 : ERROR => [['daily_actetranspiration.out', 0.026], ['daily_subsurface_runoff.out', 1.64]]
  Case 3 : OK

```
new_version_folder = 'version1'
old_version_folder = 'version2'
nb_cases = 3
c=0 #counter
threshold = 0.02
general_results = []

result_file = open("test_results2", mode = 'w+')
result_file.truncate()


for k in range(11,11+nb_cases):
    p_new, p_old = test_path(new_version_folder, old_version_folder, k)
    name_new, df_new = create_df_to_compare(p_new)
    name_old, df_old = create_df_to_compare(p_old)
    result = comparisonBIG(name_new, df_new, name_old, df_old)
    for q in result: #q[0] file name and q[1] error associated
        if comparison_threshold(q[1],threshold)==True:
            result.remove(q)
    if result==[]:
        general_results.append('OK') #=> no significant error or no error at all
    else:
        general_results.append(result)


for l in general_results:
    c+=1
    if l != 'OK':
        print('Case ',c, ' : ERROR => ', l, '\n \n')
        result_file.writelines(['\n Case ',str(c), ' : ERROR =>', str(l), '\n'])
    else :
        print('Case ',c, ' : OK \n')
        result_file.writelines(['\n Case ', str(c), ' : OK \n'])
```

Figure 2: *Calculation part for option 3.*