TCC monthly report

Introduction

This pdf is created to explain script named GR-RO_monthly.py in path I:\Konstantinos Sidiropoulos\Documentation\TCC.

Script loads a folder that corresponds to a specific month of the year (**monthly folder**). Folder contains subfolders that correspond to specific days of this particular month (**daily folders**).

Each daily folder contains four subfolders that correspond to the type of Nodes (Greek/Romanian) and the capacity direction (Import/Export). Each subfolder contains 24 CGMs.

Script automate a proccess where for each CGM a row will be saved in a dataframe and will contain columns that provide date, time, capacity direction and TCC information.

A final dataframe is created and saved in .xlsx form and contains for a period of a month all the TCCs of each hourly specified CGM.

The code has been tested and works for CGMs in UCTE format. It was used in computer 10.91.100.15 with the following installations:

- Python version 3.12.0
- Pypowsybl library: User can simply install released versions of pypowsybl from PyPI using pip:

  ```
  pip install pypowsybl
  ```

- Visual Studio Code: VSCodeUserSetup-x64-1.91.0.exe

Note: File_type was always 2D = Two Days Ahead (D-2)
All UCTE's had name structure:
ucte_filename = {Date}_{hour}_{File_type}{number}_{country_code}{number}.{format}

1. Libraries and def functions

The script:

- Imports the following libraries:

```
import pandas as pd
import os
import pypowsybl.network as pp
import pypowsybl.report as nf
import pypowsybl.loadflow as lf
import logging
```

Figure 1: Libraries and paths

import pandas : Creates dataframe with specified columns and combines all dataframes in a final .xlsx file.

Import os : Combines the output directory with the output filename.

import pypowsybl.network : Imports UCTE files.

import pypowsybl.report: Generates and manages detailed reports on the execution of load flow analysis.

import pypowsybl.loadflow : Runs AC load flows on network.

Import logging: Provides flexible framework for emmiting log messages, helping with debugging and monitoring.

- **def get_user_inputs()**: Creates a function that prompts the user for several inputs, including the base path, save folder, year and month, specific dates, and types, splitting the types into a list. If specific dates are provided, it splits them by commas; otherwise, it sets specific_dates to None, and finally returns all inputs for further processing.

```
def get_user_inputs():

    Path = input("Enter base path for CGM TCC: ex. TCC Folder ")
    Save_folder = input("Enter save folder path: ")
    Year_Month = input("Enter Year_Month (e.g., '202402'): ")
    specific_dates_input = input("Enter specific dates separated by commas ex. 20240212,20240213... (leave blank for all dates): ")
    types = input("Enter types separated by commas (e.g., 'NGR Export,NGR Import,SRO Export,SRO Import'): ")

    #Check if user has entered specific types
    if not types.strip():
        types = ['NGR Export', 'NGR Import', 'SRO Export', 'SRO Import']
    else:
        types = types.split(',')

    # Checks if user has entered specific dates
    if specific_dates_input:
        specific_dates = specific_dates_input.split(',') # keeps the list of dates separated by comma
    else:
        specific_dates = None

    base_folder = rf'{Path}\\{Year_Month}'
    return types, Year_Month, Save_folder, base_folder, specific_dates

```

Figure 2: User Inputs

- **def get_dates_from_folders(base_folder, specific_dates=None)**: This function checks if specific_dates is provided; if so, it returns those dates directly. If not, it scans the base_folder for subfolders that match the YYYYMMDD format (8 digits) and returns a list of those valid date folders.

```
def get_dates_from_folders(base_folder, specific_dates = None): # User provides specific dates otherwise function reads every subfolder
    # Return specific dates if provided
    if specific_dates:
        return specific_dates

    # If not provided, list all valid subfolders that match a date pattern
    all_dates = []
    for folder_name in os.listdir(base_folder):
        if folder_name.isdigit() and len(folder_name) == 8:  # Check for YYYYMMDD format
            all_dates.append(folder_name)

    return all_dates
```

Figure 3: Get_dates_from_folders()

2. AC loadflow

def process_ucte_file(ucte_file_path, Date, current_timestamp, Type):

Script checks if CGM exists using **os.path.isfile(ucte_file_path)**. If CGM exists script loads the network using **pp.load(ucte_file_path)**. If it does not exist, script prints an error message.

nf.Reporter() and print(str(reporter)): Creates a comprehensive analysis of the LoadFlow execution available in the terminal.

lf.Parameters(): Function that specifies the network parameters for AC loadflow execution.

lf.run_ac(network, parameters=p, reporter=reporter): Function that performs AC loadflow including the parameters of the network and creates an execution report.

```
def process_ucte_file(ucte_file_path, Date, current_timestamp, Type):
    try:
        if not os.path.isfile(ucte_file_path):
            print(f"File {ucte_file_path} does not exist. Skipping.")
            return None

        #Loads specified UCTE file
        network = pp.load(ucte_file_path)
        reporter = nf.Reporter()

        #Parameters specification for AC LoadFlow
        p = lf.Parameters(
            distributed_slack=False,
            transformer_voltage_control_on=False ,
            phase_shifter_regulation_on=True ,
            shunt_compensator_voltage_control_on=True ,
            voltage_init_mode=None,
            use_reactive_limits=None,
            twt_split_shunt_admittance=None,
            read_slack_bus=None,
            write_slack_bus=None,
            balance_type=None,
            dc_use_transformer_ratio=None,
            countries_to_balance=None,
            connected_component_mode=None,
            dc_power_factor=None,
            provider_parameters={
                'maxOuterLoopIterations': str(30),
                'lowImpedanceBranchMode': 'REPLACE_BY_MIN_IMPEDANCE_LINE',}
        )
        lf.run_ac(network, parameters=p, reporter=reporter) # You can use also report_node = 
reporter
        print(str(reporter))

```

Figure 4: AC loadflow execution

3. X-Nodes

Script creates a **X-nodes** variable inside **def process_ucte_file()** that contains all the boundary lines of the network (**network.get_dangling_lines(attributes=['bus_id', 'boundary_p'])**). The project aims to keep only the active power of specific X-nodes, so script takes only as attributes the 'bus_id' and the 'boundary_p'.

Bus_id contains the id of the real boundary buses and is used for identification of specific Romanian/Greek buses that script wants to isolate and calculate total capacity of their respectively X-nodes.

Note: Boundary line in the script is the Line between a real boundary bus and a X-node. Script chooses from the X-node side the active power and from the real boundary bus side the id of the bus.

X_nodes['bus_id'] = X_nodes['bus_id'].replace({}): Replaces specific buses names accordingly to the names they have in an other software of the company.

X_nodes['bus_id'] = X_nodes['bus_id'].astype(str): ensures string data structure

X_nodes = X_nodes.dropna(subset=['bus_id']): removes rows with empty strings from bus_id column.

if Type.startswith(' '):

filtered = X_nodes[X_nodes['bus_id'].str.startswith(' ')].copy():

Script creates an if function that checks if the **Type in types** variable starts with 'NGR' or 'SRO' and then saves in a variable named **filtered** the boundary lines whose bus_id starts with 'G' or 'R' respectively. If types variable does not match with 'NGR'/'SRO' script prints an error message.

filtered = filtered[~filtered['bus_id'].isin([])]: Removes unnecessary buses.

columns_drop = ['id', 'bus_id']: Drops unnecessary columns for the final excel structure.

tcc_sum = abs(filtered['boundary_p'].sum()): Variable that calculates the absolute active power sum of specific X-nodes.

return pd.DataFrame({}): Creates dataframe of specified structure. Date, Timestamp, Border & Direction and TCC columns are created.

NOTE: For each CGM a single row is created with the predefined columns.

If filename does not exist, an error message occurs in the terminal.

```
# Extract X-nodes and rename certain X-Nodes
X_nodes = network.get_dangling_lines(attributes=['bus_id', 'boundary_p'])
X_nodes['bus_id'] = X_nodes['bus_id'].replace({
'RIS1A41_0': 'RISAC41', 'RMED141_0': 'RMEDG41_0',
'RPDF241_0': 'RPDFE41', 'RROS241_0': 'RROSI41', 'RTINTA1_0': 'RTINTB1'
})
X_nodes['bus_id'] = X_nodes['bus_id'].astype(str)
X_nodes = X_nodes.dropna(subset=['bus_id'])

# Filter based on the Type
if Type.startswith('NGR'):
  filtered = X_nodes[X_nodes['bus_id'].str.startswith('G')]
elif Type.startswith('SRO'):
  filtered = X_nodes[X_nodes['bus_id'].str.startswith('R')]
else:
  print(f"Warning: Type {Type} does not match expected values for filtering.")
  return None

#Delete unnecessary X-Nodes
filtered = filtered[~filtered['bus_id'].isin([
  'GARACH1_0', 'RISAC41', 'RARA4D1_0', 'RNADA_1_0', 'RROSI41'
])]
columns_drop = ['id', 'bus_id'] # only p_boundary for TCC
filtered = filtered.drop(columns=columns_drop, errors='ignore')

##Capacity calculation
tcc_sum = abs(filtered['boundary_p'].sum())

# Return certain dataframe structure
return pd.DataFrame({
    'Date': [Date],
    'Timestamp': [current_timestamp],
    'Border & Direction': [Type],
    'TCC': [tcc_sum]
 })

except Exception as e:
  print(f"Error processing file {ucte_file_path}: {e}")
  return None

```

Figure 5: X-nodes & TCC calculation

4. Loop creation and data merging

def process_all_data(base_folder, Year_Month, types, Save_folder, specific_dates=None):

Function that creates the loop to iterate through each UCTE and calls process_ucte_file() to create the dataframe of specific UCTE file. Final, appends all dataframes to one and saves it in .xlsx form.

Dates variable contains a list of dates from **def get_dates_from_folders()**. An empty dataframe is created (**data = []**) that will store the dataframe created by **def process_ucte_file()** for each UCTE.

Code iterates through each date of the **dates** list.

For each date, iterates through each type of the **types** list.

Inside the loops, **destination_folder** variable contains a dynamic folder path to the specified type of CGMs (**{base_folder}\\{Date}\\CGM\\{Type}** structure).

for D in range(11): Takes values from 0 to 10 corresponding to different versions of 2D file types in UCTE filename structure.

for U in range(11): Takes values from 0 to 10 corresponding to different versions of country code (**UX**) number in UCTE filename structure.

for i in range(23): Takes values from 0 to 23.

time_value variable takes values after an algebraic addition (**= i * 100 + 30**).

base_ucte_filename = f'{Date}_{{:04d}}_2D{D}_UX{U}.uct': Structure of UCTE filenames. {{:04d}} ensures 4 digit timestamp with leading zeros if necessary.

current_ucte_filename = base_ucte_filename.format(time_value): Formats the time_value to {{:04d}}.

current_timestamp = f'{time_value // 100:02d}:30': Creates timestamp values in specific form (ex. 00:30, … ,23:30) that will be used in the final TCC excel (**'Timestamp'** column).

ucte_file_path = os.path.join(destination_folder, current_ucte_filename): Creates the path to a specific CGM in UCTE form.

result = process_ucte_file(ucte_file_path, Date, current_timestamp, Type): Saves structured dataframes for each timestamp, type, date and appends them in **data** list.

After the finalize of the loops, **final** variable concatenates all dataframes in the **data** list in order a single dataframe to be created that contains all TCC. Dataframe is stored in **output_file** in excel form.

```python
def process_all_data(base_folder, Year_Month, types, Save_folder, specific_dates=None):
    #Takes dates of specified monthly folder
    dates = get_dates_from_folders(base_folder, specific_dates)

    #Creates an empty list for the final dataframe
    data = []

    #Iterates through each 'date' folder and through each type folder inside the predefined date folder
    for Date in dates:
        for Type in types:
            destination_folder = f'{base_folder}\\{Date}\\CGM\\{Type}'

            #Iterates through each UCTE FILE
            for D in range(11):
                for U in range(11):
                    for i in range(24):
                        time_value = i * 100 + 30
                        base_ucte_filename = f'{Date}_{{:04d}}_2D{D}_UX{U}.uct'
                        current_ucte_filename = base_ucte_filename.format(time_value)
                        current_timestamp = f'{time_value // 100:02d}:30'
                        ucte_file_path = os.path.join(destination_folder, current_ucte_filename)

                        #Saves the structured dataframe from TCC of UCTE file
                        result = process_ucte_file(ucte_file_path, Date, current_timestamp, Type)

                        if result is not None:
                            data.append(result)

    # Concatenate all dataframes and save to Excel
    final = pd.concat(data, ignore_index=True) if data else pd.DataFrame()
    output_file = os.path.join(Save_folder, f'{Year_Month}_TCCS.xlsx')
    final.to_excel(output_file, index=False)
    print(f"Data saved to {output_file}")

```

Figure 6: Loops and UCTE filename structure

5. Main execution

This code block checks if the script is being run directly (not imported as a module) and, if so, it collects user inputs (e.g., types, year/month, save folder) using the get_user_inputs() function. It then processes the data based on the user's input by calling the process_all_data() function to handle the TCC data processing.

```
if __name__ == "__main__":
    # User inforamtion
    types, Year_Month, Save_folder, base_folder, specific_dates = get_user_inputs()
    #Processing User's info for TCC
    process_all_data(base_folder, Year_Month, types, Save_folder, specific_dates)

```

Figure 7: Main()

6. Merge.py

Separate script that merges .xlsx files in save_folder path. **This script is useful only if user wants to have for different months TCC.**

Merge.py is in the in path I:\Konstantinos Sidiropoulos\Documentation\TCC.

Script creates a **folder_path** variable that contains the path to the saved .xlsx files (same path with save_folder variable).

Creates a **df_list** empty dataframe and a loop that iterates through folder_path.

Loop reads all monthly TCC excels and appends files to the df_list dataframe.

Combine variable concatenates a list of individual dataframes stored in **df_list** variable and saves them in **output_file** in .xlsx form.

```
import pandas as pd
import os

#Script for user to combine monthly TCC calculations
folder_path = r'C:\Path\to\monthly\TCC\excels'
df_list = []
for filename in os.listdir(folder_path):
    if filename.endswith('.xlsx') or filename.endswith('.xls'):
        file_path = os.path.join(folder_path, filename)
        # Read each Excel file and append to the list
        df = pd.read_excel(file_path)
        df_list.append(df)

combine = pd.concat(df_list , ignore_index=True)
output_file = os.path.join(folder_path, 'combine_monthly_TCCs.xlsx')
combine.to_excel(output_file, index=False)
```

Figure 8: Monthly TCC combined