import pandas as pd

import matplotlib.pyplot as plt

import numpy as np

from pyomo.environ import ConcreteModel, Var, Objective, Constraint, Reals, NonNegativeReals, Suffix, Set, \

ConstraintList, value

from pyomo.opt import SolverFactory

df = pd.read\_pickle("dataRTE.pkl")

parc = pd.read\_pickle("parcRTE.pkl")

sample = "T"

#df = df.resample(sample).mean()

nb\_pas = len(df.index)

pas = (24\*365)/nb\_pas # en heures

pas\_par\_mois = int(nb\_pas/12)

disp\_solaire = df["Solaire normalisé"].to\_list()

disp\_eolien = df["Eolien normalisé"].to\_list()

disp = [disp\_solaire, disp\_eolien]

max\_mensuel\_hydro = (pas\*df["Hydraulique"]).resample("M").sum().to\_list()

max\_mensuel\_nucleaire = (pas\*df["Nucléaire"]).resample("M").sum().to\_list()

# PARAMETRES

alpha = 0.25 # part (MINIMALE ! cf définition contrainte) électricité d'origine renouvelable

part\_nuc = 0 # part maximale de la capacité nucléaire actuelle encore installée

part\_fossile = 0 # part maximale de la capacité charbon+fioul actuelle encore installée

part\_gaz = 0 # part maximale de la capacité gaz actuelle encore installée

G = ["Parc solaire", "Parc eolien", "Parc hydraulique", "Parc nucleaire", "Parc gaz", "Parc charbon", "Parc fioul", "Parc stockage"]

cap = parc.loc[2019, G[:-1]].to\_list()

cout\_cap = [708e3, 1480e3, 0, 4010e3, 1010e6, 2850e3, 2361e3, 0] #€/MW sauf stockage en dernier (€/MWh)

cout\_marg = [0, 0, 0, 30, 70, 86, 162, 0] #€/MWh

CO2 = [55, 7, 6, 6, 418, 1060, 730, 0] # tC02/MWh\_el

#DV = [25, 25, 80, 60, 40, 40, 20]

DV = 30 #durée de vie du projet

rendements\_batterie = [0.9, 0.9] # [charge, décharge]

rendements\_H2 = [0.65, 0.45] # [charge, décharge]

N = [n for n in range(len(G))] #pour les capacités

Np = N[2:-1]

T = [t for t in range(nb\_pas)] # pour les pas de temps

S = [0, 1, 2, 3] # pour le stockage

d = df["Consommation"].to\_list()

# MISE EN PLACE DU MODELE

m = ConcreteModel()

m.N = Set(initialize=N) # capacité solaire, éolien, pilotables, stockage

m.Np = Set(initialize=Np) # technologies pilotables

m.T = Set(initialize=T) # pas temporels

m.S = Set(initialize=S) # stockage

# VARIABLES

m.g = Var(m.N, domain=NonNegativeReals) # capacité installée pour chaque techno

m.x = Var(m.Np, m.T, domain=NonNegativeReals) # pilotage

m.s = Var(m.S, m.T, domain=NonNegativeReals) # stockage : [état, charge, décharge]

# CONTRAINTES

m.demande = Constraint(m.T, rule=lambda m, t: m.g[0] \* disp\_solaire[t] + m.g[1] \* disp\_eolien[t]

+ sum(m.x[n, t] for n in N[2:-1]) - m.s[1, t] + m.s[2, t] == d[t])

# A) contraintes sur le stockage

m.etat\_stockage\_init = Constraint(expr=m.s[0, 0] == 0)

m.etat\_stockage = Constraint(T[:-1], rule=lambda m, t: m.s[0, t+1] == m.s[0, t]+m.s[1, t+1]\*pas-m.s[2, t+1]\*pas)

m.limite\_etat\_stockage = Constraint(m.T, rule=lambda m, t: m.s[0, t] >= 0)

m.limite\_stockage = Constraint(m.T, rule=lambda m, t: m.s[0, t] <= m.g[7])

#m.limite\_cap\_batterie = Constraint(expr=m.g[7] <= max\_stockage)

# B) Contrainte sur la part de l'électricité d'origine renouvelable

m.part\_RE = Constraint(rule=lambda m, t: sum(m.g[n]\*disp[n][t]\*pas for t in T for n in N[:2])

>= alpha\*(sum(m.g[n]\*disp[n][t]\*pas for t in T for n in N[:2]) +

(sum(m.x[n, t] for n in N[2:-1] for t in T))))

#contrainte sur la part de renouvelables (= éolien + PV + hydro) dans la production

# C) contrainte sur la production des sources pilotables

m.cap\_pilotable = Constraint(m.Np, m.T,

rule=lambda m, n, t: m.x[n, t] <= m.g[n]) # aucune source ne peut dépasser sa capacité installée

# D) contraintes sur les capacités pilotables installées

m.cap\_hydro = Constraint(expr=m.g[2] == cap[2]) # la capacité hydro installée ne peut pas être changée

#m.cap\_nuc = Constraint(expr=m.g[3] <= part\_nuc \* cap[3]) # contrainte sur la capacité nucléaire

m.cap\_fossile = Constraint(expr=m.g[5]+m.g[6] <= part\_fossile\*(cap[5]+cap[6])) # contrainte sur la capacité charbon+fioul

#m.cap\_gaz = Constraint(expr=m.g[4] <= part\_gaz\*cap[4]) # contrainte sur le gaz

#m.cap\_eolien = Constraint(expr=m.g[1] >= cap[1]) # la capacité en éolien ne peut que croître

#m.cap\_solaire = Constraint(expr=m.g[0] >= cap[0]) # idem solaire

# E) contraintes sur l'énergie mensuelle max (saisonnalité hydro et nucléaire du fait des maintenances)

m.max\_energie\_hydro = ConstraintList()

#m.max\_energie\_nucleaire = ConstraintList()

for j in range(12):

m.max\_energie\_hydro.add(expr=sum(m.x[2, t] for t in range(j\*pas\_par\_mois, (j+1)\*pas\_par\_mois))\*pas

<= max\_mensuel\_hydro[j])

#m.max\_energie\_nucleaire.add(expr=sum(m.x[3, t] for t in range(j\*pas\_par\_mois, (j+1) \* pas\_par\_mois))\*pas

#<= max\_mensuel\_nucleaire[j])

# F) Contraintes de ramp-up/start-up

ramp\_up = 0.2\*60\*pas #cf thèse C.Cany: toutes les variations sont inférieures à 0,2%P\_nom/min, soit 6%P\_nom/demie-heure

ru\_yn = True #ramp\_up\_yes\_no : inclure les contraintes de ramp-up ? (augmente le temps le temps de calcul)

if ru\_yn:

#m.ramp\_up\_hydro\_max = Constraint(T[:-1], rule=lambda m, t: m.x[2, t+1] <= (1+ramp\_up)\*m.x[2, t])

#m.ramp\_up\_hydro\_min = Constraint(T[:-1], rule=lambda m, t: m.x[2, t+1] >= (1-ramp\_up)\*m.x[2, t])

m.ramp\_up\_nucleaire\_max = Constraint(T[:-1], rule=lambda m, t: m.x[3, t+1]-m.x[3, t] >= -ramp\_up/100\*m.g[3])

m.ramp\_up\_nucleaire\_min = Constraint(T[:-1], rule=lambda m, t: m.x[3, t+1]-m.x[3, t] <= ramp\_up/100\*m.g[3])

# OBJECTIF

poids\_CO2 = 0

#poids\_cout = 1-poids\_CO2

m.objectif = Objective(expr=sum(m.x[n, t] \* cout\_marg[n] \* pas \* DV for t in T for n in Np) # OPEX\*Ny pour pilotable

+ sum(m.g[0] \* disp\_solaire[t] \* cout\_marg[0] \* pas \* DV for t in T) #OPEX\*Ny pour solaire

+ sum(m.g[1] \* disp\_eolien[t] \* pas \* cout\_marg[1] \* DV for t in T) #OPEX\*Ny pour eolien

+ sum(m.s[2, t] \* cout\_marg[-1] \* pas \* DV for t in T) #OPEX\*Ny pour stockage

+ sum(m.g[n] \* cout\_cap[n] for t in T for n in N)) # coût total sur la durée de vie projet

print("init probleme ok")

opt = SolverFactory("cplex")

solver = opt.solve(m)

print("optimisation ok")

print(solver)

result = pd.DataFrame(index=df.index)

result["Solaire simulé"] = m.g[0].value\*df["Solaire normalisé"]

result["Eolien simulé"] = m.g[1].value \* df["Eolien normalisé"]

result["Production simulée"] = result["Solaire simulé"] + result["Eolien simulé"]

result["Consommation"] = df["Consommation"]

gens = ["Solaire simulé", "Eolien simulé", "Hydraulique simulé", "Nucleaire simulé", "Gaz simulé",

"Charbon simulé", "Fioul simulé"]

for n, gen in enumerate(gens[2:]):

result["{}".format(gen)] = [m.x[n + 2, t].value for t in range(nb\_pas)]

result["Production simulée"] += result["{}".format(gen)]

result["Etat stockage"] = [m.s[0,t].value for t in range(nb\_pas)]

result["Charge"] = [-m.s[1,t].value for t in range(nb\_pas)]

result["Decharge"] = [m.s[2,t].value for t in range(nb\_pas)]

# Stockage et affichage des résultats

result.to\_pickle("resultats\_stockage.pkl")

result\_stockage = open(r"resultats\_stockage.txt", "w")

for i, gen in enumerate(G[:-1]):

print("{} = {}GW".format(gen, m.g[i].value / 1000))

result\_stockage.write("{} = {}GW".format(gen, m.g[i].value / 1000))

print("{} = {}TWh".format(G[-1], m.g[7].value / 1e6))

result\_stockage.write("{} = {}TWh".format(G[-1], m.g[7].value / 1e6))

print("Coût total (capacité) = {}M€".format(value(m.objectif)/1e6))

result\_stockage.write("Coût total (capacité) = {}M€".format(value(m.objectif)/1e6))

ax = (result.loc["2019", ["Production simulée", "Consommation", "Charge", "Decharge"]+gens]/1000).plot()

ax.set\_ylabel("GW")

ax1 = ax.twinx()

(result.loc["2019", ["Etat stockage"]]/1e6).plot(ax=ax1, color="pink")

ax1.set\_ylabel("TWh")

plt.show()