Modeling progress

27.05.2021

Case-study

2 buildings on 24h

- Electricity
- Domestic hot water
- Heat space

2 inputs

- Electricity grid
- Fuel (natural gas)

2 converters

- Heat pump
- CHP

2 storages

- Battery
- Hot water storage

Pyomo operation

Model creation	*ConcreteModel() or AbstractModel().*
Sets definition	*Lists of “things” that define the problem.*
Parameters definition	*Inputs data.*
Variables definition	*Variables that will be optimized.*
Constraints definition	
Objective function creation	*Defined as a constraint. Can be maximized or minimized, no automatic multi-objective optimization. Activation/deactivation of the functions is possible.*
Model resolution	*Different solvers can be used.*

Pyomo operation

Model creation	*ConcreteModel() or AbstractModel().*
Sets definition	*Lists of “things” that define the problem.*
Parameters definition	*Inputs data.*
Variables definition	*Variables that will be optimized.*
Constraints definition	
Objective function creation	*Defined as a constraint. Can be maximized or minimized, no automatic multi-objective optimization. Activation/deactivation of the functions is possible.*
Model resolution	*Different solvers can be used.*

Abstract model

$$
\begin{array}{lll}
\min & \sum_{j=1}^{n} c_j x_j & \\
\text{s.t.} & \sum_{j=1}^{n} a_{ij} x_j \geq b_i & \forall i = 1 \ldots m \\
& x_j \geq 0 & \forall j = 1 \ldots n
\end{array}
$$

Definition of the model with AMPL format as:

```
# one way to input the data in AMPL format
# for indexed parameters, the indexes are given before the value

param m := 1 ;
param n := 2 ;

param a :=
1 1 3
1 2 4
;

param c:=
1 2
2 3
;

param b := 1 1 ;
```

Example for OPTIM-EASE

```
##
## Model hours, technologies, timesteps
##
param hours := 24 ;
param unique_conv := 2 ;
param conv := 6 ;
param unique_stor := 2 ;
param stor := 6 ;
param techs := 13 ;
param inputs := 2 ;
param demand := 3 ;

##
## Coupling matrix : In: 1:Grid / 2:CHP1 / 3:CHP2 / 4:CHP3 / 5:HP1 / 6:HP2 / 7:HP3
## Out: 1:Elec / 2:SpaceHeat / 3:DHW
##

set Out := elec sh dhw ;

set In := Electricity_grid Natural_gas ;

set InCategory := fuel elec ;

param ICategory :=
Electricity_grid elec
Natural_gas fuel
;

param IMin_capacity :=
Electricity_grid 1
Natural_gas 0
;

param IMax_capacity :=
Electricity_grid 100
Natural_gas 100000
;

param Iho :=
Electricity_grid 0
Natural_gas 14.5
;
```

Pyomo operation

Model creation	*ConcreteModel() or AbstractModel().*
Sets definition	*Lists of “things” that define the problem.*
Parameters definition	*Inputs data.*
Variables definition	*Variables that will be optimized.*
Constraints definition	
Objective function creation	*Defined as a constraint. Can be maximized or minimized, no automatic multi-objective optimization. Activation/deactivation of the functions is possible.*
Model resolution	*Different solvers can be used.*

Sets created

- **Time** : [1, 24]
- **SubTime** : [2, 24]
- **Buildings** : [1, 2]
- **UConv** : CHP, HP
- **Conv** : $CHP_{1\text{-}2\text{-}3}$, $HP_{1\text{-}2\text{-}3}$
- **UStor** : Battery, Hot water
- **Stor** : $Battery_{1\text{-}2\text{-}3}$, Hot $water_{1\text{-}2\text{-}3}$
- **Techs** : $CHP_{1\text{-}2\text{-}3}$, $HP_{1\text{-}2\text{-}3}$, $Battery_{1\text{-}2\text{-}3}$, Hot $water_{1\text{-}2\text{-}3}$
- **In** : Electricity grid, Natural gas
- **InCategory** : elec, fuel
- **Out** : elec, sh, dhw
- **OutCategory** : elec, heat
- **Heat** : sh, dhw
- **HP** : $HP_{1\text{-}2\text{-}3}$
- **CHP** : $CHP_{1\text{-}2\text{-}3}$
- **Battery, HW**
- **ElecIn** : $HP_{1\text{-}2\text{-}3}$, $Battery_{1\text{-}2\text{-}3}$
- **ElecOut** : $Battery_{1\text{-}2\text{-}3}$
- **FuelIn** : $CHP_{1\text{-}2\text{-}3}$
- **HeatIn** : Hot $water_{1\text{-}2\text{-}3}$
- **HeatOut** : $HP_{1\text{-}2\text{-}3}$, Hot $water_{1\text{-}2\text{-}3}$
- **Fuels** : Natural gas
- **Elec** : Electricity grid

Hypothesis : max 3 of the same technology /building

Pyomo operation

Model creation	*ConcreteModel() or AbstractModel().*
Sets definition	*Lists of “things” that define the problem.*
Parameters definition	*Inputs data.*
Variables definition	*Variables that will be optimized.*
Constraints definition	
Objective function creation	*Defined as a constraint. Can be maximized or minimized, no automatic multi-objective optimization. Activation/deactivation of the functions is possible.*
Model resolution	*Different solvers can be used.*

Parameters

Inputs

		Electricity_grid	Natural_gas
-	ICategory	elec	fuel
kW	IMin_capacity	1	0
kW	IMax_capacity	100	1000000000
kWh/kg	Iho	0	14,5
kWh/kg	Ihu	0	13,1
CHF/kWh	Icostw	0,2244	0,094
CHF/kWh	IFeedIn	0,1	0
CHF/kg	Icostkg	0	0
kWh	IUBP	1	137
kWh oil-eq	IPrimaryEnergyR	1	0,004
kWh oil-eq	IPrimaryEnergyNR	1	1,06
kgCO2 eq	IGWP	0,2	0,228

Converters

		CHP	Heat_Pump
kW	CMin_capacity	5	5
kW	CMax_capacity	36	50
-	CSwitching_frequency	2	4
-	CNumber_max_per_building	3	3
-	CInput	fuel	elec
-	COutput	elec+heat	heat
% Investment	CMaintenanceCost	0,05	0,02
% Investment	CInstallationCost	0,15	0,15
% Investment	CPlanificationCost	0,05	0,05
CHF/kW	CInvestmentCost	830	980
CHF	CInvestmentCostBase	20700	6950
-	CUBP	70,5	149
kWh oil-eq	CPrimaryEnergyR	0,002	0,818
kWh oil-eq	CPrimaryEnergyNR	0,502	0,908
kg CO2 eq	CGWP	0,111	0,063

Storage

		Battery	Hot_water
kWh	SMin_capacity	1	10
kWh	SMax_capacity	200	50
-	SNumber_max_per_building	3	3
-	SMin_power_of_charging	0,7	0,8
-	SMax_power_of_charging	0,7	0,8
-	SMin_power_of_discharging	0,95	1
-	SMax_power_of_discharging	0,95	1
-	SEfficiency_of_charging	0,9	0,7
-	SEfficiency_of_discharging	0,86	0,7
-	SInput	elec	heat
-	SOutput	elec	heat
% Investment	SInstallationCost	0,15	0,15
% Investment	SPlanificationCost	0,05	0,05
CHF/kWh	SInvestmentCost	570	100
CHF	SInvestmentCostBase	230	0
-	SUBP	1	1
kWh oil-eq	SPrimaryEnergyR	1	1
kWh oil-eq	SPrimaryEnergyNR	1	1
kg CO2 eq	SGWP	1	1

.dat file used in the code is created with python script data_extraction.py from an excel file

Pyomo operation

Model creation	*ConcreteModel() or AbstractModel().*
Sets definition	*Lists of “things” that define the problem.*
Parameters definition	*Inputs data.*
Variables definition	*Variables that will be optimized.*
Constraints definition	
Objective function creation	*Defined as a constraint. Can be maximized or minimized, no automatic multi-objective optimization. Activation/deactivation of the functions is possible.*
Model resolution	*Different solvers can be used.*

Variables

- **Tinp (Time, In)** : quantity of input consumed per time
- **ElecTech (Time, Building, ElecIn)** : quantity of electricity from electricity grid that goes in every technologies per building that need it, per time
- **FuelTech (Time, Fuels, Building, FuelIn)** : quantity of each fuel that goes in technologies that need it, per time and per building
- **FeedIn (Time, Building, ElecOut+CHP)** : quantity of electricity that is sold to the grid, per time and per building

- **TechCapacity (Building, Techs)** : capacity of the technology
- **TechUse (Building, Techs)** : use binary on the whole horizon
- **TechUset (Building, Time, Techs)** : use binary per time
- ***Switch (Building, Time, CHP+HP)*** *: Integer (-1, 0, 1) wich describe the transition between on and off state for CHP and HP per time* (not used yet)
- **TechIn (Building, Time, Techs)** : quantity of power received by every technology (except heat-in) per time
- **TechOut (Building, Time, Techs)** : quantity of power out of every technology (except CHP and heat-out) per time
- **TechHeatInG (Building, Time, HeatIn, Heat)** : quantity of heat received by corresponding technologies per time
- **TechHeatOutG (Building, Time, HeatOut, Heat)** : quantity of heat out of corresponding technologies per time
- **TechCHPOut (Building, Time, CHP, Out)** : quantity of power out of CHP technologies per time

<u>Hypothesis</u> : every technology can interact with others (except itself)

Variables

- **SOC (Building, Time, Battery)** : state of charge of the battery per time
- **SOCdhw (Building, Time, HW)** : state of charge of the HW dhw storage per time
- **SOCsh (Building, Time, HW)** : state of charge of the HW sh storage per time

- **TechFlow (Building, Time, Tech, Techs, Out)** : quantity of electricity/heat moving from a technology A to a technology B at a specific time and in a specific building

- **DemElec (Building, Time, Elec+ElecOut+CHP)** : quantity of electricity satisfying the demand per time and its origins, per building
- **DemSH (Building, Time, HeatOut+CHP)** : quantity of heat satisfying the demand of SH per time and its origins, per building
- **DemDHW (Building, Time, HeatOut+CHP)** : quantity of heat satisfying the demand of DHW per time and its origins, per building

<u>Hypothesis</u> : every technology can interact with others (except itself)

Pyomo operation

Model creation	*ConcreteModel() or AbstractModel().*
Sets definition	*Lists of “things” that define the problem.*
Parameters definition	*Inputs data.*
Variables definition	*Variables that will be optimized.*
Constraints definition	
Objective function creation	*Defined as a constraint. Can be maximized or minimized, no automatic multi-objective optimization. Activation/deactivation of the functions is possible.*
Model resolution	*Different solvers can be used.*

Constraints

- Demand satisfaction
- Link between the flows (heat-out, heat-in, elec-out, elec-in, …)
- Constraining a technology not to provide itself
- Bounding the capacity + introduction of the use binary (whole horizon)
- Bounding the output of the technologies with the capacity chosen
  → For HW : SOC for SH storage and SOC for DHW storage <= capacity chosen /2
- HP model *(see next page)*
  → SH relation (priority to DHW)
- CHP model (similar to HP)
- SOC definition for batteries, SH and DHW storage for HW *(see next page)*
- SOCs initial and final state to 0
- 1st definition of the use binary per time : zero if whole use binary at 0

HP model

« 3 » : random number

```python
def hp_rule(model, b, t, hp):
    efficaciteqdhw = (3 + 3 * model.Tamb[t] + 3 * 55 + 3 * model.Tamb[t] * 55 + 3 * (model.Tamb[t] ^ 2) + 3 * (55 ^ 2))
    efficacitewdhw = (1 + 1 * model.Tamb[t] + 1 * 55 + 1 * model.Tamb[t] * 55 + 1 * (model.Tamb[t] ^ 2) + 1 * (55 ^ 2))
    COPdhw = efficaciteqdhw / efficacitewdhw
    Tcond = 25 * (model.Tamb[t] > 15) + (32.5 - 1 / 2 * model.Tamb[t]) * (model.Tamb[t] <= 15)
    efficaciteqsh = (3 + 3 * model.Tamb[t] + 3 * Tcond + 3 * model.Tamb[t] * Tcond + 3 * (model.Tamb[t] ^ 2) + 3 * (Tcond ** 2))
    efficacitewsh = (1 + 1 * model.Tamb[t] + 1 * Tcond + 1 * model.Tamb[t] * Tcond + 1 * (model.Tamb[t] ^ 2) + 1 * (Tcond ** 2))
    COPsh = efficaciteqsh / efficacitewsh
    return model.TechIn[b, t, hp] == model.TechHeatOutG[b, t, hp, "dhw"] * COPdhw + model.TechHeatOutG[b, t, hp, "sh"] * COPsh

# Priority of Domestic Hot Water over Space Heat
def hp_sh_rule(model, b, t, hp):
    return model.TechHeatOutG[b, t, hp, "sh"] <= model.TechCapacity[b, hp] - model.TechHeatOutG[b, t, hp, "dhw"]
```

$$\dot{q}_c = bq_1 + bq_2 * \bar{T}_{ext} + bq_3 * \overline{35} + bq_4 * \bar{T}_{ext} * \overline{35} + bq_5 * \bar{T}_{ext}^2 + bq_6 * \overline{35}$$

$$\dot{\omega}_c = bp_1 + bp_2 * \bar{T}_{ext} + bp_3 * \overline{35} + bp_4 * \bar{T}_{ext} * \overline{35} + bp_5 * \bar{T}_{ext}^2 + bp_6 * \overline{35}$$

$$SH\,(out) \leq Capacity\; - DHW\,(out)$$

Storage model

```
def SOC_battery_rule(model, b, t, batt):
    return model.SOC[b, t, batt] == model.SOC[b, t-1, batt] + model.TechIn[b, t-1, batt] * model.SEfficiency_of_charging[batt] - \
            model.TechOut[b, t-1, batt] / model.SEfficiency_of_discharging[batt]

def init_state_battery_rule(model, b, batt):
    return model.SOC[b, 1, batt] == model.InitState

def final_state_battery_rule(model, b, batt):
    return model.SOC[b, model.hours, batt] == model.InitState
```

```
def SOC_HW_dhw_rule(model, b, t, hw):
    return model.SOCdhw[b, t, hw] == model.SOCdhw[b, t-1, hw] + model.TechHeatInG[b, t-1, hw, "dhw"] * model.SEfficiency_of_charging[hw] - \
            model.TechHeatOutG[b, t-1, hw, "dhw"] / model.SEfficiency_of_discharging[hw]

def SOC_HW_sh_rule(model, b, t, hw):
    return model.SOCsh[b, t, hw] == model.SOCsh[b, t-1, hw] + model.TechHeatInG[b, t-1, hw, "sh"] * model.SEfficiency_of_charging[hw] - \
            model.TechHeatOutG[b, t-1, hw, "sh"] / model.SEfficiency_of_discharging[hw]
```

$$SOC_{Bt+1} = SOC_{Bt} + \eta_{Bc} * P_{Bct} - (1/\eta_{Bd}) * P_{Bdt}$$ SPF model

$$SOC_{Bt} = SOC_{Bt-1} + \eta_{Bc} * P_{Bct} - (1/\eta_{Bd}) * P_{Bdt}$$ (urbs model)

Pyomo operation

Model creation	*ConcreteModel() or AbstractModel().*
Sets definition	*Lists of “things” that define the problem.*
Parameters definition	*Inputs data.*
Variables definition	*Variables that will be optimized.*
Constraints definition	
Objective function creation	*Defined as a constraint. Can be maximized or minimized, no automatic multi-objective optimization. Activation/deactivation of the functions is possible.*
Model resolution	*Different solvers can be used.*

Objective function: costs

- Inputs consumed *(l 517)*
- Maintenance costs *(l 518)*
- Installation costs *(l 519, 522)*
- Planification costs *(l 520, 523)*
- Investment costs *(l 521, 524)*
- Feed-in costs *(l 525)*

```
def objective_costs(model):
    return model.fcosts ==
```

```
517 sum(model.Icostw[inp]*model.Tinp[time, inp] for inp in model.In for time in model.Time) + \
518  sum(model.CMaintenanceCost[tech] * (model.TechCapacity[b, tech]*model.CInvestmentCost[tech] + model.CInvestmentCostBase[tech]) for b in model.Building for tech in model.Conv) + \
519  sum(model.CInstallationCost[tech] * (model.TechCapacity[b, tech] * model.CInvestmentCost[tech] + model.CInvestmentCostBase[tech]) for b in model.Building for tech in model.Conv) + \
520  sum(model.CPlanificationCost[tech] * (model.TechCapacity[b, tech] * model.CInvestmentCost[tech] + model.CInvestmentCostBase[tech]) for b in model.Building for tech in model.Conv) + \
521  sum(model.TechCapacity[b, tech] * model.CInvestmentCost[tech] + model.CInvestmentCostBase[tech] for b in model.Building for tech in model.Conv) + \
522  sum(model.SInstallationCost[tech] * (model.TechCapacity[b, tech] * model.SInvestmentCost[tech] + model.SInvestmentCostBase[tech]) for b in model.Building for tech in model.Stor) + \
523  sum(model.SPlanificationCost[tech] * (model.TechCapacity[b, tech] * model.SInvestmentCost[tech] + model.SInvestmentCostBase[tech]) for b in model.Building for tech in model.Stor) + \
524  sum(model.TechCapacity[b, tech] * model.SInvestmentCost[tech] + model.SInvestmentCostBase[tech] for b in model.Building for tech in model.Stor) + \
525  -model.IFeedIn["Electricity_grid"] * sum(model.FeedIn[t, b, tech] for t in model.Time for b in model.Building for tech in model.ElecOut | model.CHP)
```

Objective function: emissions

- Global warming potential applied to outputs *(l 529-534)*

```
def objective_emissions(model):
    return model.femissions ==
```

```
529 sum(model.Tinp[t, inp] * model.IGWP[inp] for t in model.Time for inp in model.In) + \
530  sum(model.TechOut[b, t, tech] * model.CGWP[tech] for b in model.Building for t in model.Time for tech in model.Conv - model.CHP - model.HeatOut) + \
531  sum(model.TechOut[b, t, tech] * model.SGWP[tech] for b in model.Building for t in model.Time for tech in model.Stor - model.CHP - model.HeatOut) + \
532  sum(model.TechHeatOutG[b, t, tech, heat] * model.CGWP[tech] for b in model.Building for t in model.Time for tech in model.HeatOut - model.Stor for heat in model.Heat) + \
533  sum(model.TechHeatOutG[b, t, tech, heat] * model.SGWP[tech] for b in model.Building for t in model.Time for tech in model.HeatOut - model.Conv for heat in model.Heat) + \
534  sum(model.TechCHPOut[b, t, tech, out] * model.CGWP[tech] for b in model.Building for t in model.Time for tech in model.CHP for out in model.Out)
```

Mutli-objective optimization: sigma-constraint method

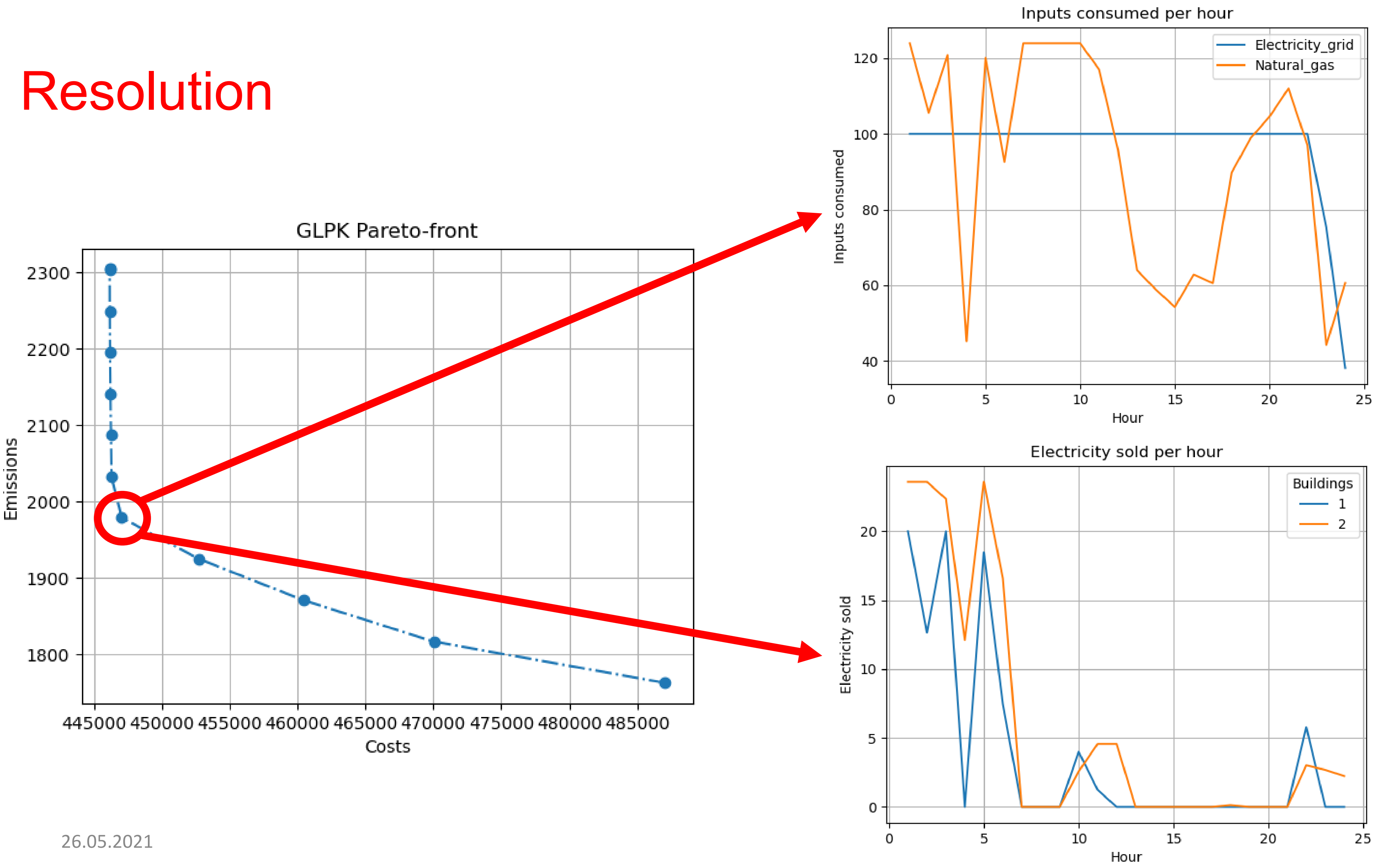
- Definition of functions A and B
- Deactivation of function A
- **Optimization**: max of function A registered
- Activation of function A, deactivation of function B
- **Optimization**: min of function A registered
- Deactivation of function A, activation of function B
- **Optimization** with constraint of function A between min and max
- Plot of the Pareto front

```python
model.O_fcosts.activate()
model.O_femissions.deactivate()

model.e = pyo.Param(initialize=0, mutable=True)
model.C_epsilon = pyo.Constraint(expr=model.femissions <= model.e+1)

instance = model.create_instance('extraction.dat')
opt = SolverFactory("glpk")  # slover can be chosen here
opt.options["mipgap"] = 0.001  # define optimality gap
opt.Tee = True
solver_manager = SolverManagerFactory("serial")  # solve instances in series
solver_manager.solve(instance, tee=True, opt=opt, timelimit=None)

n = 10
step = int((femissions_max - femissions_min) / n)
steps = list(range(int(femissions_min), int(femissions_max), step)) + [femissions_max]

f1_l = []
f2_l = []
for i in steps:
    print("etape")
    print(i)
    instance.e = i
    instance.e.pprint()
    solver_manager.solve(instance, tee=True, opt=opt, timelimit=None)
    f1_l.append(pyo.value(instance.fcosts))
    f2_l.append(pyo.value(instance.femissions))
plt.plot(f1_l, f2_l, 'o-.')
```

Pyomo operation

Model creation	*ConcreteModel() or AbstractModel().*
Sets definition	*Lists of “things” that define the problem.*
Parameters definition	*Inputs data.*
Variables definition	*Variables that will be optimized.*
Constraints definition	
Objective function creation	*Defined as a constraint. Can be maximized or minimized, no automatic multi-objective optimization. Activation/deactivation of the functions is possible.*
Model resolution	*Different solvers can be used.*

Resolution


GLPK Pareto-front
Emissions
Costs
2300
2200
2100
2000
1900
1800
445000 450000 455000 460000 465000 470000 475000 480000 485000
Inputs consumed per hour
Inputs consumed
Hour
Electricity_grid
Natural_gas
120
100
80
60
40
0
5
10
15
20
25
Electricity sold per hour
Electricity sold
Hour
Buildings
1
2
20
15
10
5
0
0
5
10
15
20
25