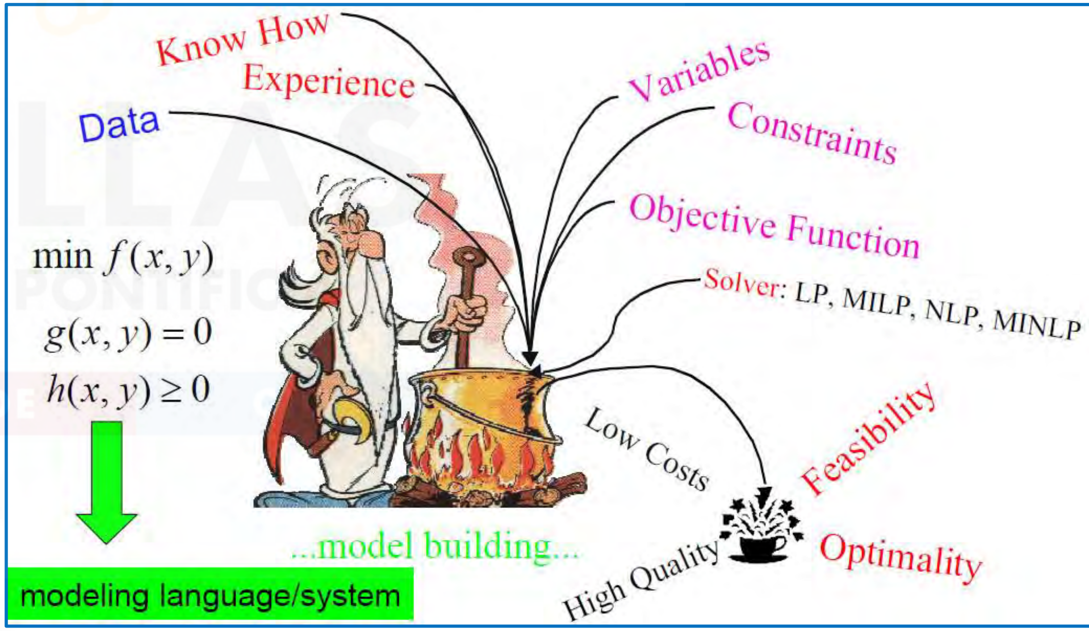
Good Optimization Modeling Practices with Pyomo

All You Wanted to Know About Practical Optimization but Were Afraid to Ask

Andres Ramos
https://www.iit.comillas.edu/aramos/
Andres.Ramos@comillas.edu

Do not confuse the ingredients of the recipe

- Mathematical formulation
  - LP, MIP, QCP, MCP
- Language
  - GAMS, Pyomo
- Solver
  - CPLEX, Gurobi
- Optimization algorithm
  - Primal simplex, dual simplex, interior point
- Input/output interfaces
  - Text file, CSV, Microsoft Excel
- Operating system
  - Windows, Linux, macOS



Source: http://www.gams.com/presentations/present_modlang.pdf

Why Pyomo?

https://pyomo.readthedocs.io/en/stable/
https://groups.google.com/forum/?nomobile=true#!forum/pyomo-forum
https://jckantor.github.io/ND-Pyomo-Cookbook/

- "Open-source optimization modeling language with a diverse set of optimization capabilities"
- No language license is required. Install

```
conda install -c conda-forge pyomo
```

- Allows the use of several solvers (open source –SCIP, GLPK and CBC– or proprietary –Gurobi, CPLEX–)

```
pyomo help -s
```

GAMS

- Pros:
  - Consistency
  - Maturity. Everything has already been written
  - Documentation
  - Customer support
- Cons:
  - No debug in the classical way

Pyomo

- Pros:
  - Flexibility, multiple choices
  - Powerful Python libraries to be used (e.g., input data, output results, visualization)
- Cons:
  - Documentation is a babel tower
  - Getting the duals

1. My First Example
2. Additional Features
3. Power Systems Models

1

My first example

1. My First Example
2. Additional Features
3. Power Systems Models

2

Additional Features

1. My First Example
2. Additional Features
3. Power Systems Models

3

Power Systems Models

openSDUC

version 1.3.23

Navigation

Introduction
Input Data
Output Results
Mathematical Formulation
Download
Contact Us

Quick search

Go

openSDUC Documentation

Open Stochastic **D**aily **U**nit **C**ommitment of Thermal and ESS Units (**openSDUC**)

"Simplicity and Transparency in Power Systems Planning"

The **openSDUC** model has been developed at the Instituto de Investigación Tecnológica (IIT) of the Universidad Pontificia Comillas.

Index

- Introduction
- Input Data
  - Dictionaries. Sets
  - Input files
  - Parameters
  - Scenario
  - Demand
  - Operating reserves
  - Duration
  - Generation
  - Energy inflows
  - Variable generation
  - Variable maximum and minimum storage
- Output Results
- Mathematical Formulation
  - Indices
  - Parameters
  - Variables
  - Equations
- Download
  - Cases

Good Optimi

1

My first example

Code conventions

- Must be defined in blocks. For example, a set and all its subsets should constitute one block in the sets section.
- Names are intended to be meaningful. Follow conventions
  - Items with the same name represent the same concept in different models
  - Units should be used in all definitions
  - Parameters are named pParameterName (e.g., pOperReserveDw)
  - Variables are named vVariableName (e.g., vReserveDown)
  - Equations are named eEquationName (e.g., eOperReserveDw)
  - Use short set names (one or two letters) for easier reading
- Equations are laid out as clearly as possible

Transportation model

There are i can factories and j consumption markets. Each factory has a maximum capacity of a_i cases, and each market demands a quantity of b_j cases (it is assumed that the total production capacity is greater than the total market demand for the problem to be feasible). The transportation cost between each factory i and each market j for each case is c_{ij}. The demand must be satisfied at minimum cost.

The decision variables of the problem will be cases transported between each factory i and each market j, x_{ij}.

My first GAMS transportation model

```
sets
   I origins      / VIGO, ALGECIRAS /
   J destinations / MADRID, BARCELONA, VALENCIA /

parameters
   pA(i) origin capacity
       / VIGO      350
         ALGECIRAS 700 /

   pB(j) destination demand
       / MADRID    400
         BARCELONA 450
         VALENCIA  150 /

table pC(i,j) per unit transportation cost
          MADRID BARCELONA VALENCIA
VIGO       0.06     0.12     0.09
ALGECIRAS  0.05     0.15     0.11

variables
   vX(i,j) units transported
   vCost   transportation cost

positive variable vX

equations
   eCost        transportation cost
   eCapacity(i) maximum capacity of each origin
   eDemand   (j) demand supply at destination ;

eCost        .. sum[(i,j), pC(i,j) * vX(i,j)] =e= vCost ;
eCapacity(i) .. sum[   j ,            vX(i,j)] =l= pA(i) ;
eDemand   (j) .. sum[ i   ,           vX(i,j)] =g= pB(j) ;

model mTransport / all /
solve mTransport using LP minimizing vCost
```

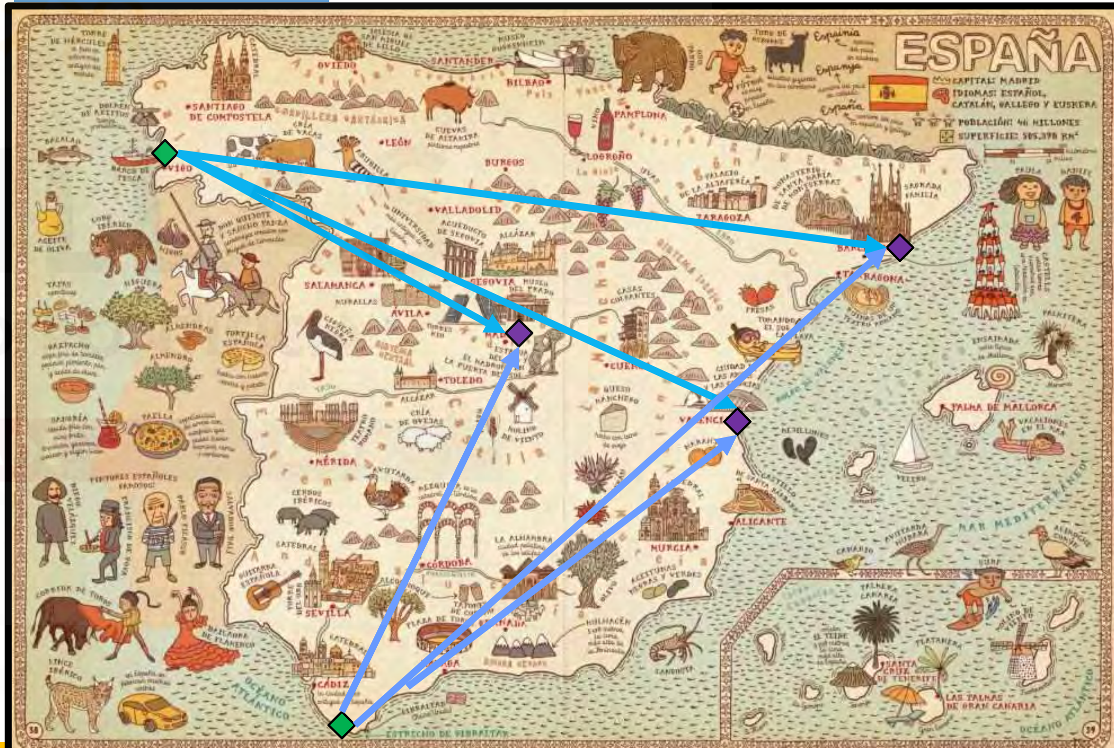
$$\min_{x_{ij}} \sum_{ij} c_{ij} x_{ij}$$

$$\sum_{j} x_{ij} \le a_i \quad \forall i$$

$$\sum_{i} x_{ij} \ge b_j \quad \forall j$$

$$x_{ij} \ge 0$$

A. Mizielinska y D. Mizielinski *Atlas del mundo: Un insólito viaje por las mil curiosidades y maravillas del mundo* Ed. Maeva 2015



My first Pyomo transportation model

```python
import pyomo.environ as pyo
from   pyomo.environ import ConcreteModel, Set, Param, Var, NonNegativeReals, Constraint, Objective, minimize, Suffix
from   pyomo.opt     import SolverFactory

mTransport = ConcreteModel('Transportation Problem')

mTransport.i = Set(initialize=['Vigo',   'Algeciras'            ], doc='origins'      )
mTransport.j = Set(initialize=['Madrid', 'Barcelona', 'Valencia'], doc='destinations')

mTransport.pA = Param(mTransport.i, initialize={'Vigo'  : 350, 'Algeciras': 700                 }, doc='origin capacity'   )
mTransport.pB = Param(mTransport.j, initialize={'Madrid': 400, 'Barcelona': 450, 'Valencia': 150}, doc='destination demand')

TransportationCost = {
    ('Vigo',      'Madrid'   ): 0.06,
    ('Vigo',      'Barcelona'): 0.12,
    ('Vigo',      'Valencia' ): 0.09,
    ('Algeciras', 'Madrid'   ): 0.05,
    ('Algeciras', 'Barcelona'): 0.15,
    ('Algeciras', 'Valencia' ): 0.11,
    }

mTransport.pC = Param(mTransport.i, mTransport.j, initialize=TransportationCost, doc='per unit transportation cost')

mTransport.vX = Var  (mTransport.i, mTransport.j, bounds=(0.0,None), doc='units transported', within=NonNegativeReals)

def eCapacity(mTransport, i):
    return sum(mTransport.vX[i,j] for j in mTransport.j) <= mTransport.pA[i]
mTransport.eCapacity = Constraint(mTransport.i, rule=eCapacity, doc='maximum capacity of each origin')

def eDemand  (mTransport, j):
    return sum(mTransport.vX[i,j] for i in mTransport.i) >= mTransport.pB[j]
mTransport.eDemand   = Constraint(mTransport.j, rule=eDemand,   doc='demand supply at destination'   )

def eCost(mTransport):
    return sum(mTransport.pC[i,j]*mTransport.vX[i,j] for i,j in mTransport.i*mTransport.j)
mTransport.eCost = Objective(rule=eCost, sense=minimize, doc='transportation cost')

mTransport.write('mTransport.lp', io_options={'symbolic_solver_labels': True})

mTransport.dual = Suffix(direction=Suffix.IMPORT)

Solver = SolverFactory('gurobi')
Solver.options['LogFile'] = 'mTransport.log'
SolverResults = Solver.solve(mTransport, tee=True)

SolverResults.write()
mTransport.pprint()

mTransport.vX.display()
for j in mTransport.j:
    print(mTransport.dual[mTransport.eDemand[j]])
```

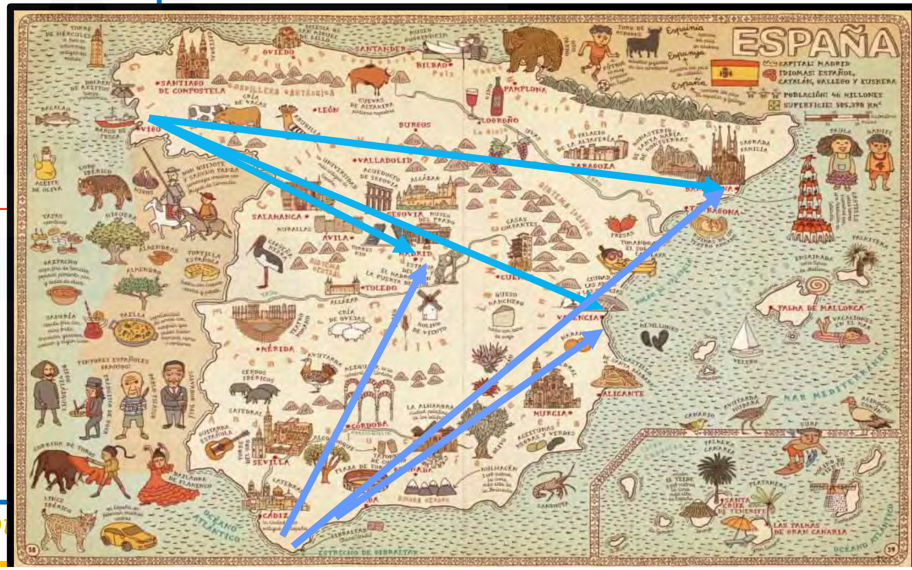
$$\min_{x_{ij}} \sum_{ij} c_{ij} x_{ij}$$

$$\sum_{j} x_{ij} \leq a_i \quad \forall i$$

$$\sum_{i} x_{ij} \geq b_j \quad \forall j$$

$$x_{ij} \geq 0$$

A. Mizielinska y D. Mizielinski *Atlas del mundo: Un insólito viaje por las mil curiosidades y maravillas del mundo* Ed. Maeva 2015



LP File: `mTransport.write('mTransport.lp', io_options={'symbolic_solver_labels': True})`

```
\* Source Pyomo model name=unknown *\

min
eCost:
+0.14999999999999999 vX(Algeciras_Barcelona)
+0.050000000000000003 vX(Algeciras_Madrid)
+0.11 vX(Algeciras_Valencia)
+0.12 vX(Vigo_Barcelona)
+0.059999999999999998 vX(Vigo_Madrid)
+0.089999999999999997 vX(Vigo_Valencia)

s.t.

c_u_eCapacity(Algeciras)_:
+1 vX(Algeciras_Barcelona)
+1 vX(Algeciras_Madrid)
+1 vX(Algeciras_Valencia)
<= 700

c_u_eCapacity(Vigo)_:
+1 vX(Vigo_Barcelona)
+1 vX(Vigo_Madrid)
+1 vX(Vigo_Valencia)
<= 350
```

```
c_l_eDemand(Barcelona)_:
+1 vX(Algeciras_Barcelona)
+1 vX(Vigo_Barcelona)
>= 450

c_l_eDemand(Madrid)_:
+1 vX(Algeciras_Madrid)
+1 vX(Vigo_Madrid)
>= 400

c_l_eDemand(Valencia)_:
+1 vX(Algeciras_Valencia)
+1 vX(Vigo_Valencia)
>= 150

c_e_ONE_VAR_CONSTANT:
ONE_VAR_CONSTANT = 1.0

bounds
   0 <= vX(Algeciras_Barcelona) <= +inf
   0 <= vX(Algeciras_Madrid) <= +inf
   0 <= vX(Algeciras_Valencia) <= +inf
   0 <= vX(Vigo_Barcelona) <= +inf
   0 <= vX(Vigo_Madrid) <= +inf
   0 <= vX(Vigo_Valencia) <= +inf
end
```

Problem summary: `SolverResults.write()`

```
# ==========================================================
# = Solver Results                                         =
# ==========================================================
# ----------------------------------------------------------
#   Problem Information
# ----------------------------------------------------------
Problem:
- Name: x7
  Lower bound: 93.5
  Upper bound: 93.5
  Number of objectives: 1
  Number of constraints: 6
  Number of variables: 7
  Number of binary variables: 0
  Number of integer variables: 0
  Number of continuous variables: 7
  Number of nonzeros: 13
  Sense: minimize
# ----------------------------------------------------------
#   Solver Information
# ----------------------------------------------------------
Solver:
- Status: ok
  Return code: 0
  Message: Model was solved to optimality (subject to tolerances), and an optimal solution is available.
  Termination condition: optimal
  Termination message: Model was solved to optimality (subject to tolerances), and an optimal solution is available.
  Wall time: 0.020067214965820312
  Error rc: 0
  Time: 0.30008649826049805
# ----------------------------------------------------------
#   Solution Information
# ----------------------------------------------------------
Solution:
- number of solutions: 0
  number of solutions displayed: 0
```

Optimal results: `mTransport.pprint()`

```
4 Set Declarations
    i : origins
        Dim=0, Dimen=1, Size=2, Domain=None, Ordered=False, Bounds=None
        ['Algecirass', 'Vigo']
    j : destinations
        Dim=0, Dimen=1, Size=3, Domain=None, Ordered=False, Bounds=None
        ['Barcelona', 'Madrid', 'Valencia']
    pC_index : Dim=0, Dimen=2, Size=6, Domain=None, Ordered=False, Bounds=None
        Virtual
    vX_index : Dim=0, Dimen=2, Size=6, Domain=None, Ordered=False, Bounds=None
        Virtual
3 Param Declarations
    pA : origin capacity
        Size=2, Index=i, Domain=Any, Default=None, Mutable=False
        Key       : Value
        Algeciras :   700
             Vigo :   350
    pB : destination demand
        Size=3, Index=j, Domain=Any, Default=None, Mutable=False
        Key       : Value
        Barcelona :   450
           Madrid :   400
         Valencia :   150
    pC : per unit transportation cost
        Size=6, Index=pC_index, Domain=Any, Default=None, Mutable=False
        Key                       : Value
        ('Algeciras', 'Barcelona') :  0.15
           ('Algeciras', 'Madrid') :  0.05
         ('Algeciras', 'Valencia') :  0.11
             ('Vigo', 'Barcelona') :  0.12
                ('Vigo', 'Madrid') :  0.06
              ('Vigo', 'Valencia') :  0.09
```

```
1 Var Declarations
    vX : units transported
        Size=6, Index=vX_index
        Key                        : Lower : Value : Upper : Fixed : Stale : Domain
        ('Algeciras', 'Barcelona') :   0.0 : 100.0 :  None : False : False :  Reals
           ('Algeciras', 'Madrid') :   0.0 : 400.0 :  None : False : False :  Reals
         ('Algeciras', 'Valencia') :   0.0 : 150.0 :  None : False : False :  Reals
             ('Vigo', 'Barcelona') :   0.0 : 350.0 :  None : False : False :  Reals
                ('Vigo', 'Madrid') :   0.0 :   0.0 :  None : False : False :  Reals
              ('Vigo', 'Valencia') :   0.0 :   0.0 :  None : False : False :  Reals
1 Objective Declarations
    eCost : transportation cost
        Size=1, Index=None, Active=True
        Key  : Active : Sense    : Expression
        None :   True : minimize : 0.06*vX[Vigo,Madrid] + 0.12*vX[Vigo,Barcelona] + 0.09*vX[Vigo,Valencia] +
0.05*vX[Algeciras,Madrid] + 0.15*vX[Algeciras,Barcelona] + 0.11*vX[Algeciras,Valencia]
2 Constraint Declarations
    eCapacity : maximum capacity of each origin
        Size=2, Index=i, Active=True
        Key       : Lower : Body                                                                  : Upper : Active
        Algeciras :  -Inf : vX[Algeciras,Madrid] + vX[Algeciras,Barcelona] + vX[Algeciras,Valencia] : 700.0 :   True
             Vigo :  -Inf :                     vX[Vigo,Madrid] + vX[Vigo,Barcelona] + vX[Vigo,Valencia] : 350.0 :   True
    eDemand : demand supply at destination
        Size=3, Index=j, Active=True
        Key       : Lower : Body                                          : Upper : Active
        Barcelona : 450.0 : vX[Vigo,Barcelona] + vX[Algeciras,Barcelona] :  +Inf :   True
           Madrid : 400.0 :       vX[Vigo,Madrid] + vX[Algeciras,Madrid] :  +Inf :   True
         Valencia : 150.0 :   vX[Vigo,Valencia] + vX[Algeciras,Valencia] :  +Inf :   True
11 Declarations: j pA pB pC_index pC vX_index vX eCapacity eDemand eCost i
```

2

Additional Features

Sets

Indices	
ω	Scenario
n	Load level
ν	Time step. Duration of each load level (e.g., 2 h, 3 h)
g	Generator (thermal or hydro unit or ESS)
t	Thermal unit
e	Energy Storage System (ESS)

- Subsets

```
mSDUC.g = Set(initialize=mSDUC.gg, ordered=False, doc='generating units', filter=lambda mSDUC,gg: gg in mSDUC.gg and pRatedMaxPower[gg] >  0)
mSDUC.t = Set(initialize=mSDUC.g , ordered=False, doc='thermal    units', filter=lambda mSDUC,g : g  in mSDUC.g  and pLinearVarCost [g] >  0)
```

- Lag and lead operators: `first/last`, `prev/next`

```
def eUCStrShut(mTEPES,sc,p,n,nr):
    if n == mTEPES.n.first():
        return mTEPES.vCommitment[sc,p,n,nr] - pInitialUC[nr]                                 == mTEPES.vStartUp[sc,p,n,nr] - mTEPES.vShutDown[sc,p,n,nr]
    else:
        return mTEPES.vCommitment[sc,p,n,nr] - mTEPES.vCommitment[sc,p,mTEPES.n.prev(n),nr] == mTEPES.vStartUp[sc,p,n,nr] - mTEPES.vShutDown[sc,p,n,nr]
mTEPES.eUCStrShut = Constraint(mTEPES.sc, mTEPES.p, mTEPES.n, mTEPES.nr, rule=eUCStrShut, doc='relation among commitment startup and shutdown')
```

$$\boxed{uc_{ng} - uc_{n-\nu,g} = su_{ng} - sd_{ng} \quad \forall ng}$$

```
# fixing the ESS inventory at the last load level
for sc,p,es in mTEPES.sc*mTEPES.p*mTEPES.es:
    mTEPES.vESSInventory[sc,p,mTEPES.n.last(),es].fix(pESSInitialInventory[es])
```

- Circular indexes (`prew/nextw`)
- Union, intersection of sets

Sparse index sets in a network

- Set of lines (initial node, final node, circuit)

```
mTEPES.la = Set(initialize=mTEPES.ni*mTEPES.nf*mTEPES.cc, ordered=False, doc='all lines', filter=lambda mTEPES,ni,nf,cc:(ni,nf,cc) in pLineX)
```

- All the connections (`ni,nf,cc`) vs. real lines (`lc`)

```
def eInstalNetCap1(mTEPES,sc,p,n,ni,nf,cc):
    return mTEPES.vFlow[sc,p,n,ni,nf,cc] / mTEPES.pLineNTC[ni,nf,cc] >= - mTEPES.vNetworkInvest[ni,nf,cc]
mTEPES.eInstalNetCap1 = Constraint(mTEPES.sc, mTEPES.p, mTEPES.n, mTEPES.lc, rule=eInstalNetCap1, doc='maximum flow by installed network capacity [p.u.]')
```

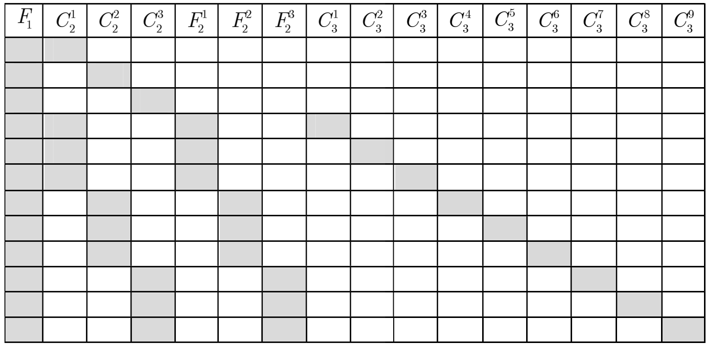
Constraint $\mathrm{l} \leq Ax \leq u$

```
#%% maximum angular difference between any two nodes
def eMaxThetaDiff(mTEPES,sc,p,n,ni,nf):
    if ni > nf and nf != mTEPES.pReferenceNode:
        return (-pMaxThetaDiff.loc[ni,nf], vTheta[sc,p,n,ni] - vTheta[sc,p,n,nf], pMaxThetaDiff.loc[ni,nf])
    else:
        return Constraint.Skip
mTEPES.eMaxThetaDiff = Constraint(mTEPES.sc, mTEPES.p, mTEPES.n, mTEPES.ni, mTEPES.nf, rule=eMaxThetaDiff, doc='maximum angle difference [rad]')
```

Sizing and timing

- Counting constraints

```
print('eBalance ... ', len(mTEPES.eBalance), ' rows’)
```



- Counting time

```
GeneratingRBITime = time.time() - StartTime
StartTime         = time.time()
print('Generating reserves/balance/inventory ... ', round(GeneratingRBITime), 's')
```

Boosting performance

- Threads

```
Solver.options['Threads'] = int((psutil.cpu_count(logical=True) +
psutil.cpu_count(logical=False))/2)
```

- Sensitivity analysis with persistent solvers
  - Sequential resolution of similar problems in memory

```
Solver.remove_constraint(model.ConstraintName)
model.del_component(model.SetName)
Solver.add_constraint(model.ConstraintName)
```

- Distributed computing
  - Create the problems and send them to be solved in parallel
  - Retrieve the solution once solved

```
model.ConstraintName.deactivate()
model.del_component(model.SetName)
model.ConstraintName.activate()
```

Fixed-Charge Transportation Problem (FCTP)

Flows
(second stage)

Investment decisions
(first stage)

Capacity of each origin

Demand of each destination

Flow can pass only for installed connections

Complete problem

$$\min_{x_{ij},y_{ij}} \sum_{ij} (f_{ij}y_{ij} + c_{ij}x_{ij})$$

$$\sum_{j} x_{ij} \leq a_i \qquad \forall i$$

$$\sum_{i} x_{ij} \geq b_j \qquad \forall j$$

$$x_{ij} \leq M_{ij}y_{ij} \qquad \forall ij$$

$$x_{ij} \geq 0, y_{ij} \in \{0,1\}$$

- Bd Relaxed Master

$$\min_{y_{ij},\theta} \sum_{ij} (f_{ij}y_{ij}) + \theta$$

$$\delta^l\theta - \theta^l \geq \sum_{ij} \pi_{ij}^l M_{ij}(y_{ij}^l - y_{ij}) \qquad l = 1, \dots, k$$

$$y_{ij} \in \{0,1\}$$

O.F. of the subproblem at iteration l

Dual variables of linking constraints at iteration l

Master proposal at iteration l

- Bd Subproblem

$$\min_{x_{ij}} \sum_{ij} (c_{ij}x_{ij})$$

$$\sum_{j} x_{ij} \leq a_i \qquad \forall i$$

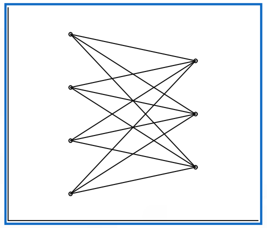
$$\sum_{i} x_{ij} \geq b_j \qquad \forall j$$

$$x_{ij} \leq M_{ij}y_{ij}^k \qquad \forall ij \qquad : \pi_{ij}^k$$

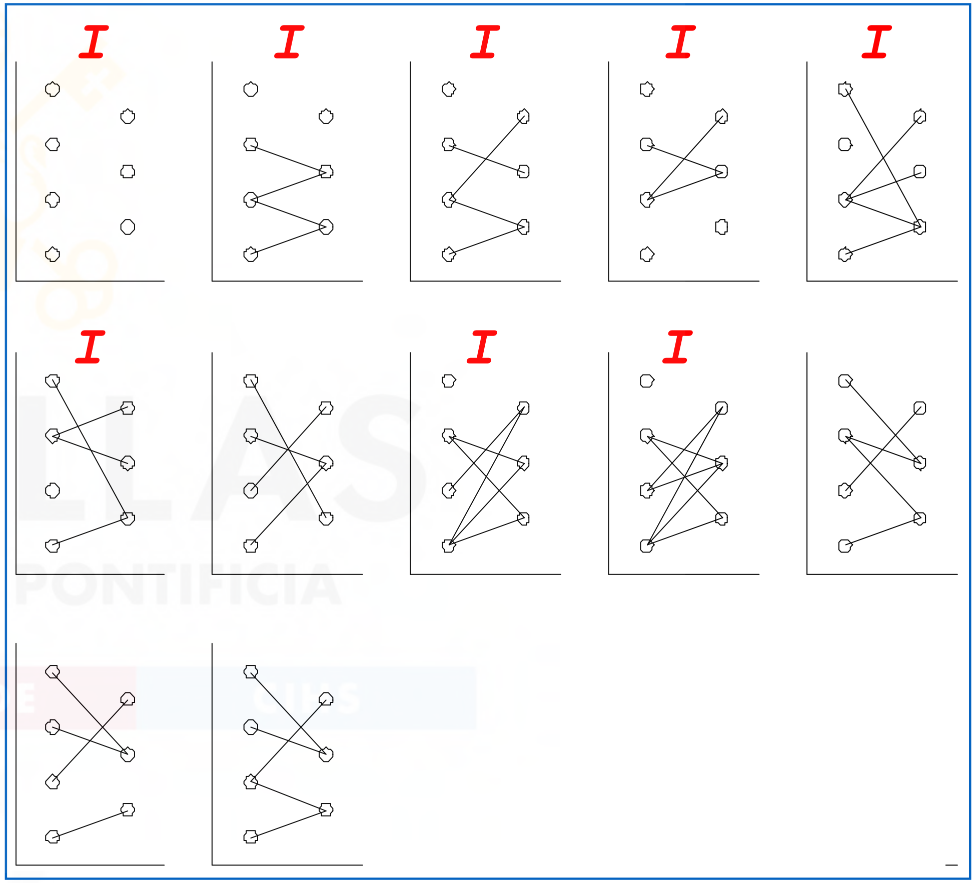
$$x_{ij} \geq 0$$

Fixed-Charge Transportation Problem. Bd Solution

- Possible arcs

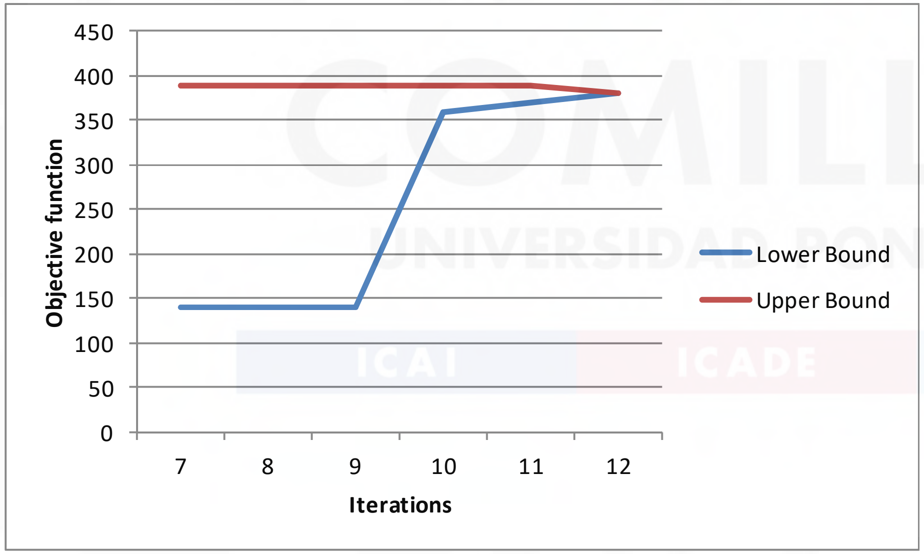


- Solutions along Benders decomposition iterations



Fixed-Charge Transportation Problem. Bd Convergence

Iteration	Lower Bound	Upper Bound
1 a 6	$-\infty$	∞
7	140	390
8	140	390
9	140	390
10	360	390
11	370	390
12	380	380



FCTP solved by Benders decomposition (i)

```
import pandas as pd
import pyomo.environ as pyo
from   pyomo.environ import ConcreteModel, Set, Param, Var, Binary, NonNegativeReals, RealSet, Constraint, Objective, minimize, Suffix, TerminationCondition
from   pyomo.opt     import SolverFactory

mFCTP      = ConcreteModel('Fixed-Charge Transportation Problem')
mMaster_Bd = ConcreteModel('Master problem')

mFCTP.i       = Set(initialize=['i1', 'i2', 'i3', 'i4'], doc='origins'      )
mFCTP.j       = Set(initialize=['j1', 'j2', 'j3'      ], doc='destinations')

mMaster_Bd.l  = Set(initialize=['it1', 'it2', 'it3', 'it4', 'it5', 'it6', 'it7', 'it8', 'it9', 'it10'], ordered=True, doc='iterations')
mMaster_Bd.ll = Set(                                                                                                    doc='iterations')

mFCTP.pA      = Param(mFCTP.i, initialize={'i1': 20, 'i2': 30, 'i3': 40, 'i4': 20}, doc='origin capacity'   )
mFCTP.pB      = Param(mFCTP.j, initialize={'j1': 20, 'j2': 50, 'j3':30          }, doc='destination demand')

FixedCost = {
    ('i1', 'j1'): 10,
    ('i1', 'j2'): 20,
    ('i1', 'j3'): 30,
    ('i2', 'j1'): 20,
    ('i2', 'j2'): 30,
    ('i2', 'j3'): 40,
    ('i3', 'j1'): 30,
    ('i3', 'j2'): 40,
    ('i3', 'j3'): 50,
    ('i4', 'j1'): 40,
    ('i4', 'j2'): 50,
    ('i4', 'j3'): 60,
    }

TransportationCost = {
    ('i1', 'j1'): 1,
    ('i1', 'j2'): 2,
    ('i1', 'j3'): 3,
    ('i2', 'j1'): 3,
    ('i2', 'j2'): 2,
    ('i2', 'j3'): 1,
    ('i3', 'j1'): 2,
    ('i3', 'j2'): 3,
    ('i3', 'j3'): 4,
    ('i4', 'j1'): 4,
    ('i4', 'j2'): 3,
    ('i4', 'j3'): 2,
    }
```

FCTP solved by Benders decomposition (ii)

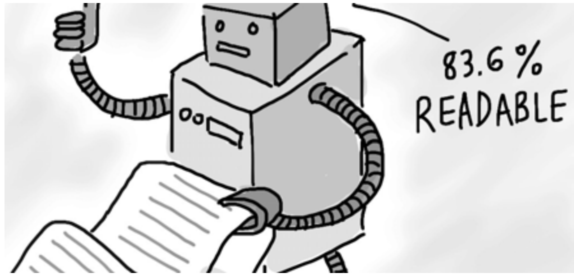
```
mFCTP.pF      = Param(mFCTP.i, mFCTP.j, initialize=FixedCost,          doc='fixed investment cost'       )
mFCTP.pC      = Param(mFCTP.i, mFCTP.j, initialize=TransportationCost, doc='per unit transportation cost')

mFCTP.vY      = Var  (mFCTP.i, mFCTP.j, bounds=(0,1), doc='units transported', within=Binary)
mMaster_Bd.vY = Var  (mFCTP.i, mFCTP.j, bounds=(0,1), doc='units transported', within=Binary)

mMaster_Bd.vTheta = Var(doc='transportation cost', within=RealSet)

mFCTP.vX      = Var  (mFCTP.i, mFCTP.j, bounds=(0.0,None), doc='units transported', within=NonNegativeReals)
mFCTP.vDNS    = Var  (         mFCTP.j, bounds=(0.0,None), doc='demand not served', within=NonNegativeReals)

def eCostMst(mMaster_Bd):
    return sum(mFCTP.pF[i,j]*mMaster_Bd.vY[i,j] for i,j in mFCTP.i*mFCTP.j) + mMaster_Bd.vTheta
mMaster_Bd.eCostMst = Objective(rule=eCostMst, sense=minimize, doc='total cost')

def eBd_Cuts(mMaster_Bd, ll):
    return mMaster_Bd.vTheta - Z2_L[ll] >= - sum(PI_L[ll,i,j] * min(mFCTP.pA[i],mFCTP.pB[j]) * (Y_L[ll,i,j] - mMaster_Bd.vY[i,j]) for i,j in mFCTP.i*mFCTP.j)

def eCostSubp(mFCTP):
    return sum(mFCTP.pC[i,j]*mFCTP.vX[i,j] for i,j in mFCTP.i*mFCTP.j) + sum(mFCTP.vDNS[j]*1000 for j in mFCTP.j)
mFCTP.eCostSubp = Objective(rule=eCostSubp, sense=minimize, doc='transportation cost')

def eCapacity(mFCTP, i):
    return sum(mFCTP.vX[i,j] for j in mFCTP.j) <= mFCTP.pA[i]
mFCTP.eCapacity  = Constraint(mFCTP.i,         rule=eCapacity,  doc='maximum capacity of each origin')

def eDemand  (mFCTP, j):
    return sum(mFCTP.vX[i,j] for i in mFCTP.i) + mFCTP.vDNS[j] >= mFCTP.pB[j]
mFCTP.eDemand    = Constraint(        mFCTP.j, rule=eDemand,    doc='demand supply at destination'    )

def eFlowLimit(mFCTP, i, j):
    return mFCTP.vX[i,j] <= min(mFCTP.pA[i],mFCTP.pB[j])*mFCTP.vY[i,j]
mFCTP.eFlowLimit = Constraint(mFCTP.i*mFCTP.j, rule=eFlowLimit, doc='arc flow limit'                  )

Solver = SolverFactory('gurobi')
Solver.options['LogFile'] = 'mFCTP.log'
mFCTP.dual = Suffix(direction=Suffix.IMPORT)

# initialization
Z_Lower = float('-inf')
Z_Upper = float(' inf')
BdTol   = 1e-6

Y_L   = pd.Series([0 ]*len(mMaster_Bd.l*mFCTP.i*mFCTP.j), index=pd.MultiIndex.from_tuples(mMaster_Bd.l*mFCTP.i*mFCTP.j))
PI_L  = pd.Series([0.]*len(mMaster_Bd.l*mFCTP.i*mFCTP.j), index=pd.MultiIndex.from_tuples(mMaster_Bd.l*mFCTP.i*mFCTP.j))
Z2_L  = pd.Series([0.]*len(mMaster_Bd.l                ), index=mMaster_Bd.l)
Delta = pd.Series([0 ]*len(mMaster_Bd.l                ), index=mMaster_Bd.l)
```

FCTP solved by Benders decomposition (iii)

```
# Benders algorithm
mMaster_Bd.vTheta.fix(0)
for l in mMaster_Bd.l:
    if abs(1-Z_Lower/Z_Upper) > BdTol or l == mMaster_Bd.l.first():

        # solving master problem
        SolverResultsMst = Solver.solve(mMaster_Bd)
        Z1               = mMaster_Bd.eCostMst.expr()

        for i,j in mFCTP.i*mFCTP.j:
            # storing the master solution
            Y_L[l,i,j] = mMaster_Bd.vY[i,j]()
            # fix investment decision for the subproblem
            mFCTP.vY[i,j].fix(Y_L[l,i,j])

        # solving subproblem
        SolverResultsSbp    = Solver.solve(mFCTP)
        Z2                  = mFCTP.eCostSubp.expr()
        Z2_L[l]             = Z2

        # storing parameters to build a new Benders cut
        if SolverResultsSbp.solver.termination_condition == TerminationCondition.infeasible:
            # the problem has to be feasible because I am not able to obtain the sum of infeasibilities of the phase I
            Delta[l] =  0
        else:
            # updating lower and upper bound
            Z_Lower =              Z1
            Z_Upper = min(Z_Upper, Z1 - mMaster_Bd.vTheta() + Z2)
            print('Iteration ', l, ' Z_Lower ... ', Z_Lower)
            print('Iteration ', l, ' Z_Upper ... ', Z_Upper)

            mMaster_Bd.vTheta.free()

            Delta[l] =  1

        for i,j in mFCTP.i*mFCTP.j:
            PI_L[l,i,j] = mFCTP.dual[mFCTP.eFlowLimit[i,j]]

        mMaster_Bd.vY.unfix()

        # add one cut
        mMaster_Bd.ll.add(l)
        ll = mMaster_Bd.ll
        mMaster_Bd.eBd_Cuts = Constraint(mMaster_Bd.ll, rule=eBd_Cuts, doc='Benders cuts')

mFCTP.eCostSubp.deactivate()
mFCTP.vY.unfix()

def eCost(mFCTP):
    return sum(mFCTP.pF[i,j]*mFCTP.vY[i,j] for i,j in mFCTP.i*mFCTP.j) + sum(mFCTP.pC[i,j]*mFCTP.vX[i,j] for i,j in mFCTP.i*mFCTP.j) + sum(mFCTP.vDNS[j]*1000 for j in mFCTP.j)
mFCTP.eCost = Objective(rule=eCost, sense=minimize, doc='total cost')

SolverResults = Solver.solve(mFCTP, tee=True)
SolverResults.write()
```

3

Power Systems Models



Simplicity and transparency

- Replicating the GAMS structure and elegance in Python/Pyomo
- Separation between
  - Dictionaries of sets
  - Parameters
  - Variables
  - Equations
  - Solve
  - Output results
- Input data and output results in text format (csv)

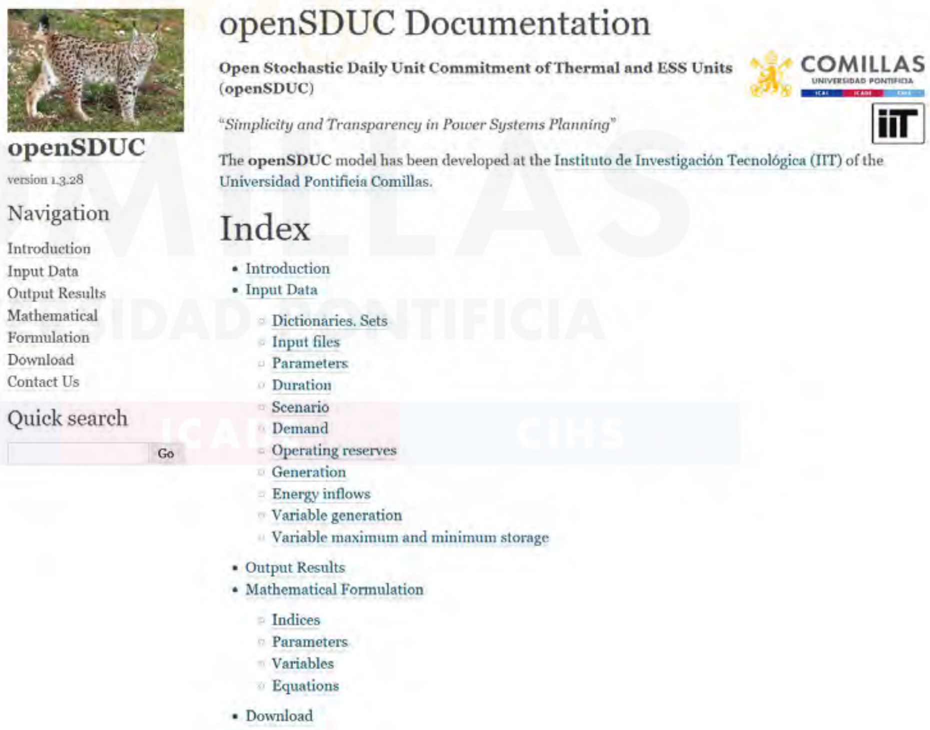
openSDUC

https://pascua.iit.comillas.edu/aramos/openSDUC/index.html

- Open Stochastic Daily Unit Commitment of Thermal and ESS Units
  - Web page created with `sphinx`

The openSDUC code is provided under the [GNU General Public License](#):
•the code can't become part of a closed-source commercial software product
•any future changes and improvements to the code remain free and open

Disclaimer:
This model is a work in progress and will be updated accordingly.



Licence

```
#                    GNU GENERAL PUBLIC LICENSE
#                       Version 3, 29 June 2007
#
#  Copyright (C) 2007 Free Software Foundation, Inc. <https://fsf.org/>
#  Everyone is permitted to copy and distribute verbatim copies
#  of this license document, but changing it is not allowed.
#
#                            Preamble
#
#   The GNU General Public License is a free, copyleft license for
# software and other kinds of works.
#
#   The licenses for most software and other practical works are designed
# to take away your freedom to share and change the works.  By contrast,
# the GNU General Public License is intended to guarantee your freedom to
# share and change all versions of a program--to make sure it remains free
# software for all its users.  We, the Free Software Foundation, use the
# GNU General Public License for most of our software; it applies also to
# any other work released this way by its authors.  You can apply it to
# your programs, too.
#
#   When we speak of free software, we are referring to freedom, not
# price.  Our General Public Licenses are designed to make sure that you
# have the freedom to distribute copies of free software (and charge for
# them if you wish), that you receive source code or can get it if you
# want it, that you can change the software or use pieces of it in new
# free programs, and that you know you can do these things.
#
#   To protect your rights, we need to prevent others from denying you
# these rights or asking you to surrender the rights.  Therefore, you have
# certain responsibilities if you distribute copies of the software, or if
# you modify it: responsibilities to respect the freedom of others.
```

```
#   For example, if you distribute copies of such a program, whether
# gratis or for a fee, you must pass on to the recipients the same
# freedoms that you received.  You must make sure that they, too, receive
# or can get the source code.  And you must show them these terms so they
# know their rights.
#
#   Developers that use the GNU GPL protect your rights with two steps:
# (1) assert copyright on the software, and (2) offer you this License
# giving you legal permission to copy, distribute and/or modify it.
#
#   For the developers' and authors' protection, the GPL clearly explains
# that there is no warranty for this free software.  For both users' and
# authors' sake, the GPL requires that modified versions be marked as
# changed, so that their problems will not be attributed erroneously to
# authors of previous versions.
#
#   Some devices are designed to deny users access to install or run
# modified versions of the software inside them, although the manufacturer
# can do so.  This is fundamentally incompatible with the aim of
# protecting users' freedom to change the software.  The systematic
# pattern of such abuse occurs in the area of products for individuals to
# use, which is precisely where it is most unacceptable.  Therefore, we
# have designed this version of the GPL to prohibit the practice for those
# products.  If such problems arise substantially in other domains, we
# stand ready to extend this provision to those domains in future versions
# of the GPL, as needed to protect the freedom of users.
#
#   Finally, every program is threatened constantly by software patents.
# States should not allow patents to restrict development and use of
# software on general-purpose computers, but in those that do, we wish to
# avoid the special danger that patents applied to a free program could
# make it effectively proprietary.  To prevent this, the GPL assures that
# patents cannot be used to render the program non-free.
```

SDUC (i)

```python
# Open Stochastic Daily Unit Commitment of Thermal and Hydro Units (openSDUC) - Version 1.3.24 - January 24, 2021
# simplicity and transparency in power systems planning

# Developed by

#    Andres Ramos
#    Instituto de Investigacion Tecnologica
#    Escuela Tecnica Superior de Ingenieria - ICAI
#    UNIVERSIDAD PONTIFICIA COMILLAS
#    Alberto Aguilera 23
#    28015 Madrid, Spain
#    Andres.Ramos@comillas.edu
#    https://pascua.iit.comillas.edu/aramos/Ramos_CV.htm

#    with the very valuable collaboration from David Dominguez (david.dominguez@comillas.edu) and Alejandro Rodriguez (argallego@comillas.edu), our local Python gurus

#%% Libraries
import pandas        as pd
import time          # count clock time
import psutil        # access the number of CPUs
import pyomo.environ as pyo
from   pyomo.environ import Set, Var, Binary, NonNegativeReals, RealSet, Constraint, ConcreteModel, Objective, minimize, Suffix, DataPortal
from   pyomo.opt     import SolverFactory

import matplotlib.pyplot as plt

StartTime = time.time()

CaseName   = '16g'                           # To select the case
SolverName = 'gurobi'

#%% model declaration
mSDUC = ConcreteModel('Open Stochastic Daily Unit Commitment of Thermal and Hydro Units (openSDUC) - Version 1.3.24 - January 24, 2021')
```

SDUC (ii)

```
#%% reading the sets
dictSets = DataPortal()
dictSets.load(filename='oUC_Dict_Scenario_'  +CaseName+'.csv', set='sc', format='set')
dictSets.load(filename='oUC_Dict_LoadLevel_' +CaseName+'.csv', set='n' , format='set')
dictSets.load(filename='oUC_Dict_Generation_'+CaseName+'.csv', set='g' , format='set')
dictSets.load(filename='oUC_Dict_Storage_'   +CaseName+'.csv', set='st', format='set')
dictSets.load(filename='oUC_Dict_Technology_'+CaseName+'.csv', set='gt', format='set')
dictSets.load(filename='oUC_Dict_Company_'   +CaseName+'.csv', set='co', format='set')

mSDUC.sc = Set(initialize=dictSets['sc'], ordered=True,  doc='scenarios'   )
mSDUC.nn = Set(initialize=dictSets['n' ], ordered=True,  doc='load levels' )
mSDUC.gg = Set(initialize=dictSets['g' ], ordered=False, doc='units'       )
mSDUC.gt = Set(initialize=dictSets['gt'], ordered=False, doc='technologies')
mSDUC.co = Set(initialize=dictSets['co'], ordered=False, doc='companies'   )
mSDUC.st = Set(initialize=dictSets['st'], ordered=False, doc='ESS types'   )

#%% reading data from CSV files
dfParameter          = pd.read_csv('oUC_Data_Parameter_'         +CaseName+'.csv', index_col=[0  ])
dfDuration           = pd.read_csv('oUC_Data_Duration_'          +CaseName+'.csv', index_col=[0  ])
dfScenario           = pd.read_csv('oUC_Data_Scenario_'          +CaseName+'.csv', index_col=[0  ])
dfDemand             = pd.read_csv('oUC_Data_Demand_'            +CaseName+'.csv', index_col=[0,1])
dfOperatingReserve   = pd.read_csv('oUC_Data_OperatingReserve_'  +CaseName+'.csv', index_col=[0,1])
dfGeneration         = pd.read_csv('oUC_Data_Generation_'        +CaseName+'.csv', index_col=[0  ])
dfVariableMaxPower   = pd.read_csv('oUC_Data_VariableGeneration_'+CaseName+'.csv', index_col=[0,1])
dfVariableMinStorage = pd.read_csv('oUC_Data_MinimumStorage_'    +CaseName+'.csv', index_col=[0,1])
dfVariableMaxStorage = pd.read_csv('oUC_Data_MaximumStorage_'    +CaseName+'.csv', index_col=[0,1])
dfEnergyInflows      = pd.read_csv('oUC_Data_EnergyInflows_'     +CaseName+'.csv', index_col=[0,1])

# substitute NaN by 0
dfParameter.fillna         (0.0, inplace=True)
dfDuration.fillna          (0.0, inplace=True)
dfScenario.fillna          (0.0, inplace=True)
dfDemand.fillna            (0.0, inplace=True)
dfOperatingReserve.fillna  (0.0, inplace=True)
dfGeneration.fillna        (0.0, inplace=True)
dfVariableMaxPower.fillna  (0.0, inplace=True)
dfVariableMinStorage.fillna(0.0, inplace=True)
dfVariableMaxStorage.fillna(0.0, inplace=True)
dfEnergyInflows.fillna     (0.0, inplace=True)
```

SDUC (iii)

```
#%% general parameters
pENSCost           = dfParameter['ENSCost' ][0] * 1e-3                                                        # cost of energy not served       [MEUR/GWh]
pCO2Cost           = dfParameter['CO2Cost' ][0]                                                               # cost of CO2 emission            [EUR/CO2
ton]
pTimeStep          = dfParameter['TimeStep'][0].astype('int')                                                 # duration of the unit time step  [h]

pDuration          = dfDuration          ['Duration'     ] * pTimeStep                                        # duration of load levels         [h]
pScenProb          = dfScenario          ['Probability' ]                                                     # probabilities of scenarios      [p.u.]
pDemand            = dfDemand            ['Demand'       ] * 1e-3                                             # demand                          [GW]
pOperReserveUp     = dfOperatingReserve  ['Up'           ] * 1e-3                                             # operating reserve up            [GW]
pOperReserveDw     = dfOperatingReserve  ['Down'         ] * 1e-3                                             # operating reserve down          [GW]
pVariableMaxPower  = dfVariableMaxPower  [list(mSDUC.gg)] * 1e-3                                              # dynamic variable maximum power  [GW]
pVariableMinStorage = dfVariableMinStorage[list(mSDUC.gg)]                                                    # dynamic variable minimum storage [GWh]
pVariableMaxStorage = dfVariableMaxStorage[list(mSDUC.gg)]                                                    # dynamic variable maximum storage [GWh]
pEnergyInflows     = dfEnergyInflows     [list(mSDUC.gg)] * 1e-3                                              # dynamic energy inflows          [GW]

# compute the demand as the mean over the time step load levels and assign it to active load levels. Idem for operating reserve, variable max power, variable min and max
storage capacity and inflows
pDemand            = pDemand.rolling            (pTimeStep).mean()
pOperReserveUp     = pOperReserveUp.rolling     (pTimeStep).mean()
pOperReserveDw     = pOperReserveDw.rolling     (pTimeStep).mean()
pVariableMaxPower  = pVariableMaxPower.rolling  (pTimeStep).mean()
pVariableMinStorage = pVariableMinStorage.rolling(pTimeStep).mean()
pVariableMaxStorage = pVariableMaxStorage.rolling(pTimeStep).mean()
pEnergyInflows     = pEnergyInflows.rolling     (pTimeStep).mean()

pDemand.fillna            (0.0, inplace=True)
pOperReserveUp.fillna     (0.0, inplace=True)
pOperReserveDw.fillna     (0.0, inplace=True)
pVariableMaxPower.fillna  (0.0, inplace=True)
pVariableMinStorage.fillna(0.0, inplace=True)
pVariableMaxStorage.fillna(0.0, inplace=True)
pEnergyInflows.fillna     (0.0, inplace=True)
```

SDUC (iv)

```
if pTimeStep > 1:
    # assign duration 0 to load levels not being considered, active load levels are at the end of every pTimeStep
    for i in range(pTimeStep-2,-1,-1):
        pDuration[range(i,len(mSDUC.nn),pTimeStep)] = 0
    # drop levels with duration 0
    pDemand             = pDemand.loc            [pDemand.index            [range(pTimeStep-1,len(mSDUC.sc*mSDUC.nn),pTimeStep)]]
    pOperReserveUp      = pOperReserveUp.loc     [pOperReserveUp.index     [range(pTimeStep-1,len(mSDUC.sc*mSDUC.nn),pTimeStep)]]
    pOperReserveDw      = pOperReserveDw.loc     [pOperReserveDw.index     [range(pTimeStep-1,len(mSDUC.sc*mSDUC.nn),pTimeStep)]]
    pVariableMaxPower   = pVariableMaxPower.loc  [pVariableMaxPower.index  [range(pTimeStep-1,len(mSDUC.sc*mSDUC.nn),pTimeStep)]]
    pVariableMinStorage = pVariableMinStorage.loc[pVariableMinStorage.index[range(pTimeStep-1,len(mSDUC.sc*mSDUC.nn),pTimeStep)]]
    pVariableMaxStorage = pVariableMaxStorage.loc[pVariableMaxStorage.index[range(pTimeStep-1,len(mSDUC.sc*mSDUC.nn),pTimeStep)]]
    pEnergyInflows      = pEnergyInflows.loc     [pEnergyInflows.index     [range(pTimeStep-1,len(mSDUC.sc*mSDUC.nn),pTimeStep)]]

#%% generation parameters
pGenToTechnology  = dfGeneration['Technology'      ]                                                      # generator association to technology
pGenToCompany     = dfGeneration['Company'         ]                                                      # generator association to company
pRatedMinPower    = dfGeneration['MinimumPower'    ] * 1e-3                                               # rated minimum power          [GW]
pRatedMaxPower    = dfGeneration['MaximumPower'    ] * 1e-3                                               # rated maximum power          [GW]
pLinearVarCost    = dfGeneration['LinearTerm'      ] * 1e-3 * dfGeneration['FuelCost'] + dfGeneration['OMVariableCost'] * 1e-3  # linear   term variable cost
[MEUR/GWh]
pConstantVarCost  = dfGeneration['ConstantTerm'    ] * 1e-6 * dfGeneration['FuelCost']                    # constant term variable cost  [MEUR/h]
pStartUpCost      = dfGeneration['StartUpCost'     ]                                                      # startup  cost                [MEUR]
pShutDownCost     = dfGeneration['ShutDownCost'    ]                                                      # shutdown cost                [MEUR]
pRampUp           = dfGeneration['RampUp'          ] * 1e-3                                               # ramp up   rate               [GW/h]
pRampDw           = dfGeneration['RampDown'        ] * 1e-3                                               # ramp down rate               [GW/h]
pCO2EmissionRate  = dfGeneration['CO2EmissionRate' ] * 1e-3                                               # emission  rate               [t
CO2/MWh]
pUpTime           = dfGeneration['UpTime'          ]                                                      # minimum up   time            [h]
pDwTime           = dfGeneration['DownTime'        ]                                                      # minimum down time            [h]
pMaxCharge        = dfGeneration['MaximumCharge'   ] * 1e-3                                               # maximum ESS charge           [GW]
pInitialInventory = dfGeneration['InitialStorage'  ]                                                      # initial ESS storage          [GWh]
pRatedMinStorage  = dfGeneration['MinimumStorage'  ]                                                      # minimum ESS storage          [GWh]
pRatedMaxStorage  = dfGeneration['MaximumStorage'  ]                                                      # maximum ESS storage          [GWh]
pEfficiency       = dfGeneration['Efficiency'      ]                                                      #         ESS efficiency       [p.u.]
pStorageType      = dfGeneration['StorageType'     ]                                                      #         ESS type

ReadingDataTime = time.time() - StartTime
StartTime       = time.time()
print('Reading    input data                 ... ', round(ReadingDataTime), 's')
```

SDUC (v)

```
#%% defining subsets: active load levels (n), thermal units (t), ESS units (es), all the lines (la), candidate lines (lc) and lines with losses (ll)
mSDUC.n  = Set(initialize=mSDUC.nn, ordered=True , doc='load levels'      , filter=lambda mSDUC,nn: nn in mSDUC.nn and pDuration     [nn] >  0  )
mSDUC.n2 = Set(initialize=mSDUC.nn, ordered=True , doc='load levels'      , filter=lambda mSDUC,nn: nn in mSDUC.nn and pDuration     [nn] >  0  )
mSDUC.g  = Set(initialize=mSDUC.gg, ordered=False, doc='generating units', filter=lambda mSDUC,gg: gg in mSDUC.gg and pRatedMaxPower[gg] >  0.0)
mSDUC.t  = Set(initialize=mSDUC.g , ordered=False, doc='thermal    units', filter=lambda mSDUC,g : g  in mSDUC.g  and pLinearVarCost [g] >  0.0)
mSDUC.r  = Set(initialize=mSDUC.g , ordered=False, doc='RES        units', filter=lambda mSDUC,g : g  in mSDUC.g  and pLinearVarCost [g] == 0.0 and pRatedMaxStorage[g] == 0.0)
mSDUC.es = Set(initialize=mSDUC.g , ordered=False, doc='ESS        units', filter=lambda mSDUC,g : g  in mSDUC.g  and                             pRatedMaxStorage[g] >  0.0)

# non-RES units
mSDUC.nr = mSDUC.g - mSDUC.r

# variable minimum and maximum power
pVariableMaxPower = pVariableMaxPower.replace(0.0, float('nan'))[mSDUC.g]
pMinPower         = pd.DataFrame([pRatedMinPower]*len(pVariableMaxPower.index), index=pd.MultiIndex.from_tuples(pVariableMaxPower.index), columns=pRatedMinPower.index)[mSDUC.g]
pMaxPower         = pd.DataFrame([pRatedMaxPower]*len(pVariableMaxPower.index), index=pd.MultiIndex.from_tuples(pVariableMaxPower.index), columns=pRatedMaxPower.index)[mSDUC.g]
pMaxPower         = pVariableMaxPower.where(pVariableMaxPower < pMaxPower, other=pMaxPower)
pMaxPower2ndBlock = pMaxPower - pMinPower

# variable minimum and maximum storage capacity
pVariableMinStorage = pVariableMinStorage.replace(0.0, float('nan'))[mSDUC.g]
pVariableMaxStorage = pVariableMaxStorage.replace(0.0, float('nan'))[mSDUC.g]
pMinStorage         = pd.DataFrame([pRatedMinStorage]*len(pVariableMinStorage.index), index=pd.MultiIndex.from_tuples(pVariableMinStorage.index), columns=pRatedMinStorage.index)[mSDUC.g]
pMaxStorage         = pd.DataFrame([pRatedMaxStorage]*len(pVariableMaxStorage.index), index=pd.MultiIndex.from_tuples(pVariableMaxStorage.index), columns=pRatedMaxStorage.index)[mSDUC.g]
pMinStorage         = pVariableMinStorage.where(pVariableMinStorage > pMinStorage, other=pMinStorage)
pMaxStorage         = pVariableMaxStorage.where(pVariableMaxStorage < pMaxStorage, other=pMaxStorage)
```

SDUC (vi)

```
# values < 1e-6 times the maximum system demand are converted to 0
pEpsilon = pDemand.max()*1e-6
# these parameters are in GW
pDemand          [pDemand          < pEpsilon] = 0.0
pOperReserveUp   [pOperReserveUp   < pEpsilon] = 0.0
pOperReserveDw   [pOperReserveDw   < pEpsilon] = 0.0
pMinPower        [pMinPower        < pEpsilon] = 0.0
pMaxPower        [pMaxPower        < pEpsilon] = 0.0
pMaxPower2ndBlock[pMaxPower2ndBlock < pEpsilon] = 0.0
pMaxCharge       [pMaxCharge       < pEpsilon] = 0.0
pEnergyInflows   [pEnergyInflows   < pEpsilon/pTimeStep] = 0.0
# these parameters are in GWh
pMinStorage      [pMinStorage      < pEpsilon] = 0.0
pMaxStorage      [pMaxStorage      < pEpsilon] = 0.0
# this option avoids a warning in the following assignments
pd.options.mode.chained_assignment = None

# minimum up and down time converted to an integer number of time steps
pUpTime = round(pUpTime/pTimeStep).astype('int')
pDwTime = round(pDwTime/pTimeStep).astype('int')

# thermal and variable units ordered by increasing variable cost
mSDUC.go = pLinearVarCost.sort_values().index

# determine the initial committed units and their output
pInitialOutput = pd.Series([0.0]*len(mSDUC.g), dfGeneration.index)
pInitialUC     = pd.Series([0.0]*len(mSDUC.g), dfGeneration.index)
pSystemOutput  = 0.0
for go in mSDUC.go:
    n1 = next(iter(mSDUC.sc*mSDUC.n))
    if pSystemOutput < pDemand[n1]:
        if go in mSDUC.r:
            pInitialOutput[go] = pMaxPower[go][n1]
        else:
            pInitialOutput[go] = pMinPower[go][n1]
        pInitialUC     [go] = 1
        pSystemOutput      += pInitialOutput[go]
```

SDUC (vii)

```python
#%% variables
mSDUC.vTotalVCost     = Var(                             within=NonNegativeReals,                                                                            doc='total system variable cost [MEUR]')
mSDUC.vTotalECost     = Var(                             within=NonNegativeReals,                                                                            doc='total system emission cost [MEUR]')
mSDUC.vTotalOutput    = Var(mSDUC.sc, mSDUC.n, mSDUC.g , within=NonNegativeReals, bounds=lambda mSDUC,sc,n,g :(0.0,pMaxPower          [g ][sc,n]),           doc='total output of the unit     [GW]')
mSDUC.vOutput2ndBlock = Var(mSDUC.sc, mSDUC.n, mSDUC.nr, within=NonNegativeReals, bounds=lambda mSDUC,sc,n,nr:(0.0,pMaxPower2ndBlock  [nr][sc,n]),           doc='second block of the unit     [GW]')
mSDUC.vReserveUp      = Var(mSDUC.sc, mSDUC.n, mSDUC.nr, within=NonNegativeReals, bounds=lambda mSDUC,sc,n,nr:(0.0,pMaxPower2ndBlock  [nr][sc,n]),           doc='operating reserve up         [GW]')
mSDUC.vReserveDown    = Var(mSDUC.sc, mSDUC.n, mSDUC.nr, within=NonNegativeReals, bounds=lambda mSDUC,sc,n,nr:(0.0,pMaxPower2ndBlock  [nr][sc,n]),           doc='operating reserve down       [GW]')
mSDUC.vESSInventory   = Var(mSDUC.sc, mSDUC.n, mSDUC.es, within=NonNegativeReals, bounds=lambda mSDUC,sc,n,es:(pMinStorage[es][sc,n],pMaxStorage[es][sc,n]), doc='ESS inventory               [GWh]')
mSDUC.vESSSpillage    = Var(mSDUC.sc, mSDUC.n, mSDUC.es, within=NonNegativeReals,                                                                            doc='ESS spillage                [GWh]')
mSDUC.vESSCharge      = Var(mSDUC.sc, mSDUC.n, mSDUC.es, within=NonNegativeReals, bounds=lambda mSDUC,sc,n,es:(0.0,pMaxCharge         [es]      ),           doc='ESS    charge power          [GW]')
mSDUC.vENS            = Var(mSDUC.sc, mSDUC.n,           within=NonNegativeReals, bounds=lambda mSDUC,sc,n   :(0.0,pDemand                 [sc,n]),           doc='energy not served in node    [GW]')

mSDUC.vCommitment     = Var(          mSDUC.n, mSDUC.nr, within=Binary,                                                                                      doc='commitment of the unit      {0,1}')
mSDUC.vStartUp        = Var(          mSDUC.n, mSDUC.nr, within=Binary,                                                                                      doc='StartUp    of the unit      {0,1}')
mSDUC.vShutDown       = Var(          mSDUC.n, mSDUC.nr, within=Binary,                                                                                      doc='ShutDown   of the unit      {0,1}')

# fixing the ESS inventory at the last load level at the end of the time scope
for sc,es in mSDUC.sc*mSDUC.es:
    mSDUC.vESSInventory[sc,mSDUC.n.last(),es].fix(pInitialInventory[es])

#%% definition of the time-steps leap to observe the stored energy at ESS
pCycleTimeStep = pUpTime*0
for es in mSDUC.es:
    if  pStorageType[es] == 'Daily'  :
        pCycleTimeStep[es] = int(  24/pTimeStep)
    if  pStorageType[es] == 'Weekly' :
        pCycleTimeStep[es] = int( 168/pTimeStep)
    if  pStorageType[es] == 'Monthly':
        pCycleTimeStep[es] = int( 672/pTimeStep)
    if  pStorageType[es] == 'Yearly' :
        pCycleTimeStep[es] = int(8736/pTimeStep)
```

SDUC (viii)

```
# fixing the ESS inventory at the end of the following pCycleTimeStep (weekly, yearly), i.e., for daily ESS is fixed at the end of the week, for weekly/monthly ESS is fixed at the end of the year
for sc,n,es in mSDUC.sc*mSDUC.n*mSDUC.es:
     if pStorageType[es] == 'Daily'   and mSDUC.n.ord(n) % ( 168/pTimeStep) == 0:
         mSDUC.vESSInventory[sc,n,es].fix(pInitialInventory[es])
     if pStorageType[es] == 'Weekly'  and mSDUC.n.ord(n) % (8736/pTimeStep) == 0:
         mSDUC.vESSInventory[sc,n,es].fix(pInitialInventory[es])
     if pStorageType[es] == 'Monthly' and mSDUC.n.ord(n) % (8736/pTimeStep) == 0:
         mSDUC.vESSInventory[sc,n,es].fix(pInitialInventory[es])

SettingUpDataTime = time.time() - StartTime
StartTime         = time.time()
print('Setting up input data                 ... ', round(SettingUpDataTime), 's')

def eTotalVCost(mSDUC):
    return mSDUC.vTotalVCost == (sum(pScenProb[sc] * pDuration[n] * pENSCost             * mSDUC.vENS        [sc,n   ] for sc,n    in mSDUC.sc*mSDUC.n         ) +
                                 sum(pScenProb[sc] * pDuration[n] * pLinearVarCost  [nr] * mSDUC.vTotalOutput[sc,n,nr] for sc,n,nr in mSDUC.sc*mSDUC.n*mSDUC.nr) +
                                 sum(                pDuration[n] * pConstantVarCost[nr] * mSDUC.vCommitment [   n,nr]                                          +
                                                                    pStartUpCost    [nr] * mSDUC.vStartUp    [   n,nr]                                          +
                                                                    pShutDownCost   [nr] * mSDUC.vShutDown   [   n,nr] for    n,nr in          mSDUC.n*mSDUC.nr) )
mSDUC.eTotalVCost = Constraint(rule=eTotalVCost, doc='total system variable cost [MEUR]')

def eTotalECost(mSDUC):
    return mSDUC.vTotalECost == sum(pScenProb[sc] * pCO2Cost * pCO2EmissionRate[nr] * mSDUC.vTotalOutput[sc,n,nr] for sc,n,nr in mSDUC.sc*mSDUC.n*mSDUC.nr)
mSDUC.eTotalECost = Constraint(rule=eTotalECost, doc='total system emission cost [MEUR]')

def eTotalTCost(mSDUC):
    return mSDUC.vTotalVCost + mSDUC.vTotalECost
mSDUC.eTotalTCost = Objective(rule=eTotalTCost, sense=minimize, doc='total system cost [MEUR]')

GeneratingOFTime = time.time() - StartTime
StartTime        = time.time()
print('Generating objective function         ... ', round(GeneratingOFTime), 's')
```

SDUC (ix)

```
#%% constraints
def eOperReserveUp(mSDUC,sc,n):
    if pOperReserveUp[sc,n]:
        return sum(mSDUC.vReserveUp  [sc,n,nr] for nr in mSDUC.nr) >= pOperReserveUp[sc,n]
    else:
        return Constraint.Skip
mSDUC.eOperReserveUp = Constraint(mSDUC.sc, mSDUC.n, rule=eOperReserveUp, doc='up   operating reserve [GW]')

def eOperReserveDw(mSDUC,sc,n):
    if pOperReserveDw[sc,n]:
        return sum(mSDUC.vReserveDown[sc,n,nr] for nr in mSDUC.nr) >= pOperReserveDw[sc,n]
    else:
        return Constraint.Skip
mSDUC.eOperReserveDw = Constraint(mSDUC.sc, mSDUC.n, rule=eOperReserveDw, doc='down operating reserve [GW]')

def eBalance(mSDUC,sc,n):
    return sum(mSDUC.vTotalOutput[sc,n,g] for g in mSDUC.g) - sum(mSDUC.vESSCharge[sc,n,es] for es in mSDUC.es) + mSDUC.vENS[sc,n] == pDemand[sc,n]
mSDUC.eBalance = Constraint(mSDUC.sc, mSDUC.n, rule=eBalance, doc='load generation balance [GW]')

def eESSInventory(mSDUC,sc,n,es):
    if   mSDUC.n.ord(n) == pCycleTimeStep[es]:
        return pInitialInventory[es]                                        + sum(pDuration[n2]*(pEnergyInflows[es][sc,n2] - mSDUC.vTotalOutput[sc,n2,es] + pEfficiency[es]*mSDUC.vESSCharge[sc,n2,es]) for n2 in list(mSDUC.n2)[mSDUC.n.ord(n)-
pCycleTimeStep[es]:mSDUC.n.ord(n)]) == mSDUC.vESSInventory[sc,n,es] + mSDUC.vESSSpillage[sc,n,es]
    elif mSDUC.n.ord(n) >  pCycleTimeStep[es] and mSDUC.n.ord(n) % pCycleTimeStep[es] == 0:
        return mSDUC.vESSInventory[sc,mSDUC.n.prev(n,pCycleTimeStep[es]),es] + sum(pDuration[n2]*(pEnergyInflows[es][sc,n2] - mSDUC.vTotalOutput[sc,n2,es] + pEfficiency[es]*mSDUC.vESSCharge[sc,n2,es]) for n2 in list(mSDUC.n2)[mSDUC.n.ord(n)-
pCycleTimeStep[es]:mSDUC.n.ord(n)]) == mSDUC.vESSInventory[sc,n,es] + mSDUC.vESSSpillage[sc,n,es]
    else:
        return Constraint.Skip
mSDUC.eESSInventory = Constraint(mSDUC.sc, mSDUC.n, mSDUC.es, rule=eESSInventory, doc='ESS inventory balance [GWh]')

GeneratingRBITime = time.time() - StartTime
StartTime         = time.time()
print('Generating reserves/balance/inventory ... ', round(GeneratingRBITime), 's')
```

SDUC (x)

```
#%%
def eMaxOutput2ndBlock(mSDUC,sc,n,nr):
    if   pOperReserveUp[sc,n] and pMaxPower2ndBlock[nr][sc,n]:
        return (mSDUC.vOutput2ndBlock[sc,n,nr] + mSDUC.vReserveUp  [sc,n,nr]) / pMaxPower2ndBlock[nr][sc,n] <= mSDUC.vCommitment[n,nr]
    else:
        return Constraint.Skip
mSDUC.eMaxOutput2ndBlock = Constraint(mSDUC.sc, mSDUC.n, mSDUC.nr, rule=eMaxOutput2ndBlock, doc='max output of the second block of a committed unit [p.u.]')

def eMinOutput2ndBlock(mSDUC,sc,n,nr):
    if   pOperReserveDw[sc,n] and pMaxPower2ndBlock[nr][sc,n]:
        return (mSDUC.vOutput2ndBlock[sc,n,nr] + mSDUC.vReserveDown[sc,n,nr]) / pMaxPower2ndBlock[nr][sc,n] >= 0.0
    else:
        return Constraint.Skip
mSDUC.eMinOutput2ndBlock = Constraint(mSDUC.sc, mSDUC.n, mSDUC.nr, rule=eMinOutput2ndBlock, doc='min output of the second block of a committed unit [p.u.]')

def eTotalOutput(mSDUC,sc,n,nr):
    if pMinPower[nr][sc,n] == 0.0:
        return mSDUC.vTotalOutput[sc,n,nr]                          ==                           mSDUC.vOutput2ndBlock[sc,n,nr]
    else:
        return mSDUC.vTotalOutput[sc,n,nr] / pMinPower[nr][sc,n] == mSDUC.vCommitment[n,nr] + mSDUC.vOutput2ndBlock[sc,n,nr] / pMinPower[nr][sc,n]

mSDUC.eTotalOutput = Constraint(mSDUC.sc, mSDUC.n, mSDUC.nr, rule=eTotalOutput, doc='total output of a unit [GW]')

def eUCStrShut(mSDUC,n,nr):
    if n == mSDUC.n.first():
        return mSDUC.vCommitment[n,nr] - pInitialUC[nr]                        == mSDUC.vStartUp[n,nr] - mSDUC.vShutDown[n,nr]
    else:
        return mSDUC.vCommitment[n,nr] - mSDUC.vCommitment[mSDUC.n.prev(n),nr] == mSDUC.vStartUp[n,nr] - mSDUC.vShutDown[n,nr]
mSDUC.eUCStrShut = Constraint(mSDUC.n, mSDUC.nr, rule=eUCStrShut, doc='relation among commitment startup and shutdown')

GeneratingGenConsTime = time.time() - StartTime
StartTime             = time.time()
print('Generating generation constraints     ... ', round(GeneratingGenConsTime), 's')

#%%
def eRampUp(mSDUC,sc,n,t):
    if   pRampUp[t] and pRampUp[t] < pMaxPower2ndBlock[t][sc,n] and n == mSDUC.n.first():
        return (mSDUC.vOutput2ndBlock[sc,n,t] - max(pInitialOutput[t]-pMinPower[t][sc,n],0.0) + mSDUC.vReserveUp  [sc,n,t]) / pDuration[n] / pRampUp[t] <= mSDUC.vCommitment[n,t] - mSDUC.vStartUp[n,t]
    elif pRampUp[t] and pRampUp[t] < pMaxPower2ndBlock[t][sc,n]:
        return (mSDUC.vOutput2ndBlock[sc,n,t] - mSDUC.vOutput2ndBlock[sc,mSDUC.n.prev(n),t]   + mSDUC.vReserveUp  [sc,n,t]) / pDuration[n] / pRampUp[t] <= mSDUC.vCommitment[n,t] - mSDUC.vStartUp[n,t]
    else:
        return Constraint.Skip
mSDUC.eRampUp = Constraint(mSDUC.sc, mSDUC.n, mSDUC.t, rule=eRampUp, doc='maximum ramp up   [p.u.]')

def eRampDw(mSDUC,sc,n,t):
    if   pRampDw[t] and pRampDw[t] < pMaxPower2ndBlock[t][sc,n] and n == mSDUC.n.first():
        return (mSDUC.vOutput2ndBlock[sc,n,t] - max(pInitialOutput[t]-pMinPower[t][sc,n],0.0) - mSDUC.vReserveDown[sc,n,t]) / pDuration[n] / pRampDw[t] >= - pInitialUC[t]                          + mSDUC.vShutDown[n,t]
    elif pRampDw[t] and pRampDw[t] < pMaxPower2ndBlock[t][sc,n]:
        return (mSDUC.vOutput2ndBlock[sc,n,t] - mSDUC.vOutput2ndBlock[sc,mSDUC.n.prev(n),t]   - mSDUC.vReserveDown[sc,n,t]) / pDuration[n] / pRampDw[t] >= - mSDUC.vCommitment[mSDUC.n.prev(n),t] + mSDUC.vShutDown[n,t]
    else:
        return Constraint.Skip
mSDUC.eRampDw = Constraint(mSDUC.sc, mSDUC.n, mSDUC.t, rule=eRampDw, doc='maximum ramp down [p.u.]')

GeneratingRampsTime = time.time() - StartTime
StartTime           = time.time()
print('Generating ramps   up/down            ... ', round(GeneratingRampsTime), 's')
```

SDUC (xi)

```python
#%%
def eMinUpTime(mSDUC,n,t):
    if pUpTime[t] > 1 and mSDUC.n.ord(n) >= pUpTime[t]:
        return sum(mSDUC.vStartUp [n2,t] for n2 in list(mSDUC.n2)[mSDUC.n.ord(n)-pUpTime[t]:mSDUC.n.ord(n)]) <=     mSDUC.vCommitment[n,t]
    else:
        return Constraint.Skip
mSDUC.eMinUpTime   = Constraint(mSDUC.n, mSDUC.t, rule=eMinUpTime  , doc='minimum up   time [h]')

def eMinDownTime(mSDUC,n,t):
    if pDwTime[t] > 1 and mSDUC.n.ord(n) >= pDwTime[t]:
        return sum(mSDUC.vShutDown[n2,t] for n2 in list(mSDUC.n2)[mSDUC.n.ord(n)-pDwTime[t]:mSDUC.n.ord(n)]) <= 1 - mSDUC.vCommitment[n,t]
    else:
        return Constraint.Skip
mSDUC.eMinDownTime = Constraint(mSDUC.n, mSDUC.t, rule=eMinDownTime, doc='minimum down time [h]')

GeneratingMinUDTime = time.time() - StartTime
StartTime           = time.time()
print('Generating minimum up/down time       ... ', round(GeneratingMinUDTime), 's')

#%% solving the problem
mSDUC.write('openSDUC_'+CaseName+'.lp', io_options={'symbolic_solver_labels': True}) # create lp-format file
Solver = SolverFactory(SolverName)                                                   # select solver
if SolverName == 'gurobi':
    Solver.options['LogFile'       ] = 'openSDUC_'+CaseName+'.log'
    #Solver.options['IISFile'      ] = 'openSDUC_'+CaseName+'.ilp'                   # should be uncommented to show results of IIS
    #Solver.options['Method'       ] = 2                                             # barrier method
    Solver.options['MIPGap'        ] = 0.02
    Solver.options['Threads'       ] = int((psutil.cpu_count(logical=True) + psutil.cpu_count(logical=False))/2)
    #Solver.options['TimeLimit'     ] = 7200
    #Solver.options['IterationLimit'] = 7200000
SolverResults = Solver.solve(mSDUC, tee=True)                                        # tee=True displays the output of the solver
SolverResults.write()                                                                # summary of the solver results

#%% fix values of binary variables to get dual variables and solve it again
for n,t in mSDUC.n*mSDUC.t:
    mSDUC.vCommitment[n,t].fix(mSDUC.vCommitment[n,t]())
    mSDUC.vStartUp   [n,t].fix(mSDUC.vStartUp   [n,t]())
    mSDUC.vShutDown  [n,t].fix(mSDUC.vShutDown  [n,t]())

if SolverName == 'gurobi':
    Solver.options['relax_integrality'] = 1                                          # introduced to show results of the dual variables
mSDUC.dual    = Suffix(direction=Suffix.IMPORT)
SolverResults = Solver.solve(mSDUC, tee=True)                                        # tee=True displays the output of the solver
SolverResults.write()                                                                # summary of the solver results

SolvingTime = time.time() - StartTime
StartTime   = time.time()
print('Solving                               ... ', round(SolvingTime), 's')

print('Objective function value                  ', mSDUC.eTotalTCost.expr())
```

SDUC (xii)

```
#%% inverse index generator to technology and to company
pTechnologyToGen = pGenToTechnology.reset_index().set_index('Technology').set_axis(['Generator'], axis=1, inplace=False)['Generator']
pCompanyToGen    = pGenToCompany.reset_index().set_index   ('Company'   ).set_axis(['Generator'], axis=1, inplace=False)['Generator']

#%% outputting the generation operation

OutputResults = pd.Series(data=[mSDUC.vCommitment[n,nr]() for n,nr in mSDUC.n*mSDUC.t], index=pd.MultiIndex.from_tuples(list(mSDUC.n*mSDUC.t)))
OutputResults.to_frame(name='p.u.').reset_index().pivot_table(index=['level_0'], columns='level_1', values='p.u.').rename_axis(['LoadLevel'], axis=0).rename_axis([None],
axis=1).to_csv('oUC_Result_GenerationCommitment_'+CaseName+'.csv', sep=',')
OutputResults = pd.Series(data=[mSDUC.vStartUp   [n,nr]() for n,nr in mSDUC.n*mSDUC.t], index=pd.MultiIndex.from_tuples(list(mSDUC.n*mSDUC.t)))
OutputResults.to_frame(name='p.u.').reset_index().pivot_table(index=['level_0'], columns='level_1', values='p.u.').rename_axis(['LoadLevel'], axis=0).rename_axis([None],
axis=1).to_csv('oUC_Result_GenerationStartUp_'   +CaseName+'.csv', sep=',')
OutputResults = pd.Series(data=[mSDUC.vShutDown  [n,nr]() for n,nr in mSDUC.n*mSDUC.t], index=pd.MultiIndex.from_tuples(list(mSDUC.n*mSDUC.t)))
OutputResults.to_frame(name='p.u.').reset_index().pivot_table(index=['level_0'], columns='level_1', values='p.u.').rename_axis(['LoadLevel'], axis=0).rename_axis([None],
axis=1).to_csv('oUC_Result_GenerationShutDown_'  +CaseName+'.csv', sep=',')

if sum(pOperReserveUp[sc,n] for sc,n in mSDUC.sc*mSDUC.n):
    OutputResults = pd.Series(data=[mSDUC.vReserveUp  [sc,n,nr]()*1e3 for sc,n,nr in mSDUC.sc*mSDUC.n*mSDUC.nr], index=pd.MultiIndex.from_tuples(list(mSDUC.sc*mSDUC.n*mSDUC.nr)))
    OutputResults.to_frame(name='MW').reset_index().pivot_table(index=['level_0','level_1'], columns='level_2', values='MW').rename_axis(['Scenario','LoadLevel'], axis=0).rename_axis([None],
axis=1).to_csv('oUC_Result_GenerationReserveUp_'+CaseName+'.csv', sep=',')

if sum(pOperReserveDw[sc,n] for sc,n in mSDUC.sc*mSDUC.n):
    OutputResults = pd.Series(data=[mSDUC.vReserveDown[sc,n,nr]()*1e3 for sc,n,nr in mSDUC.sc*mSDUC.n*mSDUC.nr], index=pd.MultiIndex.from_tuples(list(mSDUC.sc*mSDUC.n*mSDUC.nr)))
    OutputResults.to_frame(name='MW').reset_index().pivot_table(index=['level_0','level_1'], columns='level_2', values='MW').rename_axis(['Scenario','LoadLevel'], axis=0).rename_axis([None],
axis=1).to_csv('oUC_Result_GenerationReserveDown_'+CaseName+'.csv', sep=',')

OutputResults = pd.Series(data=[mSDUC.vTotalOutput[sc,n,g]()*1e3 for sc,n,g in mSDUC.sc*mSDUC.n*mSDUC.g], index=pd.MultiIndex.from_tuples(list(mSDUC.sc*mSDUC.n*mSDUC.g)))
OutputResults.to_frame(name='MW').reset_index().pivot_table(index=['level_0','level_1'], columns='level_2', values='MW').rename_axis(['Scenario','LoadLevel'], axis=0).rename_axis([None],
axis=1).to_csv('oUC_Result_GenerationOutput_'+CaseName+'.csv', sep=',')

OutputResults = pd.Series(data=[mSDUC.vENS[sc,n]()*1e3 for sc,n in mSDUC.sc*mSDUC.n], index=pd.MultiIndex.from_tuples(list(mSDUC.sc*mSDUC.n)))
OutputResults.to_frame(name='MW').reset_index().pivot_table(index=['level_0','level_1'], values='MW').rename_axis(['Scenario','LoadLevel'], axis=0).rename_axis([None],
axis=1).to_csv('oUC_Result_PNS_'+CaseName+'.csv', sep=',')

OutputResults = pd.Series(data=[mSDUC.vENS[sc,n]()*pDuration[n] for sc,n in mSDUC.sc*mSDUC.n], index=pd.MultiIndex.from_tuples(list(mSDUC.sc*mSDUC.n)))
OutputResults.to_frame(name='GWh').reset_index().pivot_table(index=['level_0','level_1'], values='GWh').rename_axis(['Scenario','LoadLevel'], axis=0).rename_axis([None],
axis=1).to_csv('oUC_Result_ENS_'+CaseName+'.csv', sep=',')

OutputResults = pd.Series(data=[(pMaxPower[g][sc,n]-mSDUC.vTotalOutput[sc,n,g]())*1e3 for sc,n,g in mSDUC.sc*mSDUC.n*mSDUC.r], index=pd.MultiIndex.from_tuples(list(mSDUC.sc*mSDUC.n*mSDUC.r)))
OutputResults.to_frame(name='MW').reset_index().pivot_table(index=['level_0','level_1'], columns='level_2', values='MW').rename_axis(['Scenario','LoadLevel'], axis=0).rename_axis([None],
axis=1).to_csv('oUC_Result_RESCurtailment_'+CaseName+'.csv', sep=',')
```

SDUC (xiii)

```
OutputResults = pd.Series(data=[mSDUC.vTotalOutput[sc,n,g]()*pDuration[n]              for sc,n,g in mSDUC.sc*mSDUC.n*mSDUC.g], index=pd.MultiIndex.from_tuples(list(mSDUC.sc*mSDUC.n*mSDUC.g)))
OutputResults.to_frame(name='GWh' ).reset_index().pivot_table(index=['level_0','level_1'], columns='level_2', values='GWh' ).rename_axis(['Scenario','LoadLevel'], axis=0).rename_axis([None],
axis=1).to_csv('oUC_Result_GenerationEnergy_'+CaseName+'.csv', sep=',')

OutputResults = pd.Series(data=[mSDUC.vTotalOutput[sc,n,nr]()*pCO2EmissionRate[nr]*1e3 for sc,n,nr in mSDUC.sc*mSDUC.n*mSDUC.t], index=pd.MultiIndex.from_tuples(list(mSDUC.sc*mSDUC.n*mSDUC.t)))
OutputResults.to_frame(name='tCO2').reset_index().pivot_table(index=['level_0','level_1'], columns='level_2', values='tCO2').rename_axis(['Scenario','LoadLevel'], axis=0).rename_axis([None],
axis=1).to_csv('oUC_Result_GenerationEmission_'+CaseName+'.csv', sep=',')

#%% outputting the ESS operation
if len(mSDUC.es):
    OutputResults = pd.Series(data=[mSDUC.vESSCharge   [sc,n,es]()*1e3          for sc,n,es in mSDUC.sc*mSDUC.n*mSDUC.es], index=pd.MultiIndex.from_tuples(list(mSDUC.sc*mSDUC.n*mSDUC.es)))
    OutputResults.to_frame(name='MW' ).reset_index().pivot_table(index=['level_0','level_1'], columns='level_2', values='MW' ).rename_axis(['Scenario','LoadLevel'], axis=0).rename_axis([None],
axis=1).to_csv('oUC_Result_ESSChargeOutput_'+CaseName+'.csv', sep=',')

    OutputResults = pd.Series(data=[mSDUC.vESSCharge[sc,n,es]()*pDuration[n]     for sc,n,es in mSDUC.sc*mSDUC.n*mSDUC.es], index=pd.MultiIndex.from_tuples(list(mSDUC.sc*mSDUC.n*mSDUC.es)))
    OutputResults.to_frame(name='GWh').reset_index().pivot_table(index=['level_0','level_1'], columns='level_2', values='GWh').rename_axis(['Scenario','LoadLevel'], axis=0).rename_axis([None],
axis=1).to_csv('oUC_Result_ESSChargeEnergy_'+CaseName+'.csv', sep=',')

    OutputResults = pd.Series(data=[OutputResults[sc,n].filter(pTechnologyToGen[gt]).sum() for sc,n,gt in mSDUC.sc*mSDUC.n*mSDUC.gt], index=pd.MultiIndex.from_tuples(list(mSDUC.sc*mSDUC.n*mSDUC.gt)))
    OutputResults.to_frame(name='GWh').reset_index().pivot_table(index=['level_0','level_1'], columns='level_2', values='GWh').rename_axis(['Scenario','LoadLevel'], axis=0).rename_axis([None],
axis=1).to_csv('oUC_Result_ESSTechnologyEnergy_'+CaseName+'.csv', sep=',')

    OutputResults = pd.Series(data=[mSDUC.vESSInventory[sc,n,es]()              for sc,n,es in mSDUC.sc*mSDUC.n*mSDUC.es], index=pd.MultiIndex.from_tuples(list(mSDUC.sc*mSDUC.n*mSDUC.es)))
    OutputResults *= 1e3
    OutputResults.to_frame(name='GWh').reset_index().pivot_table(index=['level_0','level_1'], columns='level_2', values='GWh', dropna=False).rename_axis(['Scenario','LoadLevel'], axis=0).rename_axis([None],
axis=1).to_csv('oUC_Result_ESSInventory_'+CaseName+'.csv', sep=',')

    OutputResults = pd.Series(data=[mSDUC.vESSSpillage [sc,n,es]()              for sc,n,es in mSDUC.sc*mSDUC.n*mSDUC.es], index=pd.MultiIndex.from_tuples(list(mSDUC.sc*mSDUC.n*mSDUC.es)))
    OutputResults *= 1e3
    OutputResults.to_frame(name='GWh').reset_index().pivot_table(index=['level_0','level_1'], columns='level_2', values='GWh', dropna=False).rename_axis(['Scenario','LoadLevel'], axis=0).rename_axis([None],
axis=1).to_csv('oUC_Result_ESSSpillage_'+CaseName+'.csv', sep=',')

    OutputResults = pd.Series({Key:OptimalSolution.value*1e3 for Key,OptimalSolution in mSDUC.vESSCharge.items()})
    OutputResults = pd.Series(data=[OutputResults[sc,n].filter(pTechnologyToGen[gt]).sum() for sc,n,gt in mSDUC.sc*mSDUC.n*mSDUC.gt], index=pd.MultiIndex.from_tuples(list(mSDUC.sc*mSDUC.n*mSDUC.gt)))
    OutputResults.to_frame(name='MW' ).reset_index().pivot_table(index=['level_0','level_1'], columns='level_2', values='MW' ).rename_axis(['Scenario','LoadLevel'], axis=0).rename_axis([None],
axis=1).to_csv('oUC_Result_TechnologyCharge_'+CaseName+'.csv', sep=',')

    TechnologyCharge = OutputResults.loc[:,:,:]

#%% plot SRMC for all the scenarios
RESCurtailment = OutputResults.loc[:,:,:]

fig, fg = plt.subplots()
for r in mSDUC.r:
    fg.plot(range(len(mSDUC.sc*mSDUC.n)), RESCurtailment[:,:,r], label=r)
fg.set(xlabel='Hours', ylabel='MW')
fg.set_ybound(lower=0)
plt.title('RES Curtailment')
fg.tick_params(axis='x', rotation=90)
fg.legend()
plt.tight_layout()
#plt.show()
plt.savefig('oUC_Plot_RESCurtailment_'+CaseName+'.png', bbox_inches=None)
```

SDUC (xiv)

```
OutputResults = pd.Series({Key:OptimalSolution.value*1e3 for Key,OptimalSolution in mSDUC.vTotalOutput.items()})
OutputResults = pd.Series(data=[OutputResults[sc,n].filter(pTechnologyToGen[gt]).sum() for sc,n,gt in mSDUC.sc*mSDUC.n*mSDUC.gt], index=pd.MultiIndex.from_tuples(list(mSDUC.sc*mSDUC.n*mSDUC.gt)))
OutputResults.to_frame(name='MW').reset_index().pivot_table(index=['level_0','level_1'], columns='level_2', values='MW').rename_axis(['Scenario','LoadLevel'], axis=0).rename_axis([None], axis=1).to_csv('oUC_Result_TechnologyOutput_'+CaseName+'.csv',
sep=',')

TechnologyOutput = OutputResults.loc[:,:,:]

for sc in mSDUC.sc:
    fig, fg = plt.subplots()
    fg.stackplot(range(len(mSDUC.n)),  TechnologyOutput.loc[sc,:,:].values.reshape(len(mSDUC.n),len(mSDUC.gt)).transpose().tolist(), labels=list(mSDUC.gt))
    fg.plot     (range(len(mSDUC.n)), -TechnologyCharge.loc[sc,:,'ESS'], label='ESSCharge', linewidth=0.5, color='b')
    fg.plot     (range(len(mSDUC.n)),  pDemand[sc]*1e3,                  label='Demand'   , linewidth=0.5, color='k')
    fg.set(xlabel='Hours', ylabel='MW')
    plt.title(sc)
    fg.tick_params(axis='x', rotation=90)
    fg.legend()
    plt.tight_layout()
    #plt.show()
    plt.savefig('oUC_Plot_TechnologyOutput_'+sc+'_'+CaseName+'.png', bbox_inches=None)

OutputResults = pd.Series(data=[mSDUC.vTotalOutput[sc,n,g]()*pDuration[n] for sc,n,g in mSDUC.sc*mSDUC.n*mSDUC.g], index=pd.MultiIndex.from_tuples(list(mSDUC.sc*mSDUC.n*mSDUC.g)))
OutputResults = pd.Series(data=[OutputResults[sc,n].filter(pTechnologyToGen[gt]).sum() for sc,n,gt in mSDUC.sc*mSDUC.n*mSDUC.gt], index=pd.MultiIndex.from_tuples(list(mSDUC.sc*mSDUC.n*mSDUC.gt)))
OutputResults.to_frame(name='GWh').reset_index().pivot_table(index=['level_0','level_1'], columns='level_2', values='GWh').rename_axis(['Scenario','LoadLevel'], axis=0).rename_axis([None], axis=1).to_csv('oUC_Result_TechnologyEnergy_'+CaseName+'.csv',
sep=',')

#%% outputting the SRMC
if SolverName == 'gurobi':
    OutputResults = pd.Series(data=[mSDUC.dual[mSDUC.eBalance[sc,n]]*1e3/pScenProb[sc]/pDuration[n] for sc,n in mSDUC.sc*mSDUC.n], index=pd.MultiIndex.from_tuples(list(mSDUC.sc*mSDUC.n)))
    OutputResults.to_frame(name='SRMC').reset_index().pivot_table(index=['level_0','level_1'], values='SRMC').rename_axis(['Scenario','LoadLevel'], axis=0).rename_axis([None], axis=1).to_csv('oUC_Result_SRMC_'+CaseName+'.csv', sep=',')

    #%% plot SRMC for all the scenarios
    SRMC = OutputResults.loc[:,:]

    fig, fg = plt.subplots()
    for sc in mSDUC.sc:
        fg.plot(range(len(mSDUC.n)), SRMC[sc], label=sc)
        fg.set(xlabel='Hours', ylabel='EUR/MWh')
        fg.set_ybound(lower=0, upper=100)
        plt.title('SRMC')
        fg.tick_params(axis='x', rotation=90)
        fg.legend()
        plt.tight_layout()
        #plt.show()
        plt.savefig('oUC_Plot_SRMC_'+CaseName+'.png', bbox_inches=None)

WritingResultsTime = time.time() - StartTime
StartTime          = time.time()
print('Writing output results                ... ', round(WritingResultsTime), 's')
print('Total time                            ... ', round(ReadingDataTime + GeneratingOFTime + GeneratingRBITime + GeneratingGenConsTime + GeneratingRampsTime + GeneratingMinUDTime + SolvingTime + WritingResultsTime), 's')
```

JuanitoJaimitoJorgito

https://gitlab001.iit.comillas.edu/pdeotaola/Ejemplo_Optimizacion_Python-Pyomo

- Simple Unit Commitment with profit maximization against prices of the day-ahead market

openTEPES

https://opentepes.readthedocs.io/en/latest/index.html
https://github.com/IIT-EnergySystemModels/openTEPES

- Open Generation, Storage, and Transmission Operation and Expansion Planning Model with RES and ESS
  - Web page created with sphinx

The openTEPES code is provided under the GNU General Public License:
•the code can't become part of a closed-source commercial software product
•any future changes and improvements to the code remain free and open

Disclaimer:
This model is a work in progress and will be updated accordingly.


openTEPES

version 4.6.1rc1

Navigation

Introduction
Input Data
Output Results
Mathematical Formulation
Research projects
Publications
Download & Installation
Contact Us

Quick search

Go

openTEPES Documentation

Open Generation, Storage, and Transmission Operation and Expansion Planning Model with RES and **ESS** (**openTEPES**)

"Simplicity and Transparency in Power Systems Planning"

The **openTEPES** model has been developed at the Instituto de Investigación Tecnológica (IIT) of the Universidad Pontificia Comillas.

Reference: A. Ramos, E. Quispe, S. Lumbreras "OpenTEPES: Open-source Transmission and Generation Expansion Planning" SoftwareX 18: June 2022 10.1016/j.softx.2022.101070

Index

- Introduction
- Input Data
  - Acronyms
  - Dictionaries. Sets
  - Input files
  - Options
  - Parameters
  - Period
  - Scenario
  - Stage
  - Adequacy reserve margin
  - Duration
  - Demand
  - System inertia
  - Upward and downward operating reserves
  - Generation
  - Variable maximum and minimum generation
  - Variable maximum and minimum consumption
  - Energy inflows

Enjoy Formulating, writing and solving optimization models

Thank you for your attention

Prof. Andres Ramos

https://www.iit.comillas.edu/aramos/

Andres.Ramos@comillas.edu