COMILLAS
UNIVERSIDAD PONTIFICIA
ICAI ICADE CIHS

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/

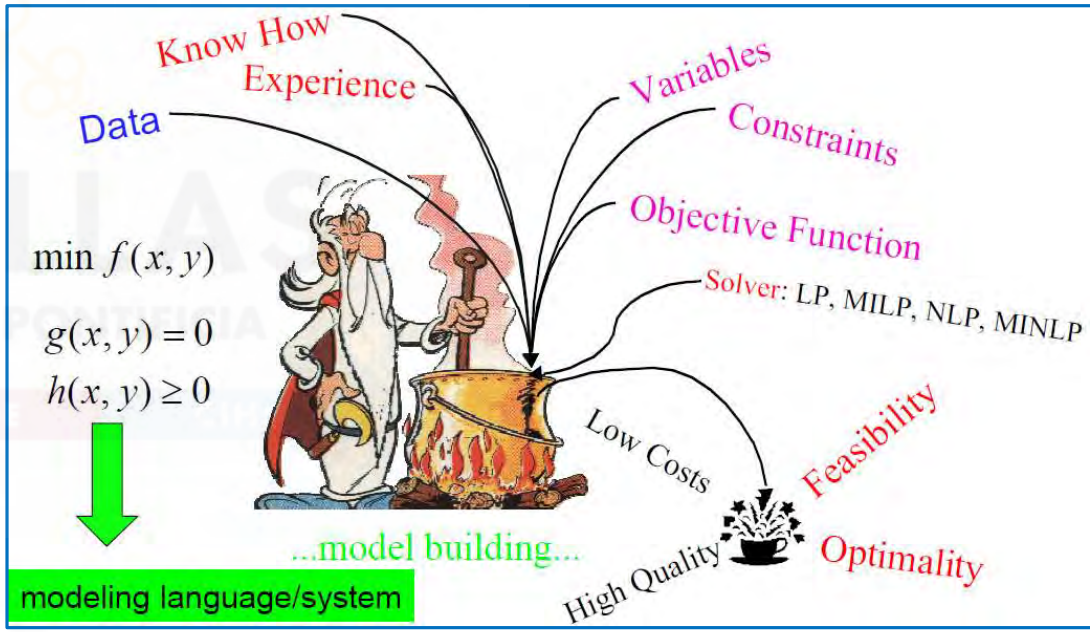
Andres.Ramos@comillas.edu

arght@mit.edu

comillas.edu

Do not confuse the ingredients of the recipe

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



Why Pyomo?

https://pyomo.readthedocs.io/en/stable/
https://groups.google.com/forum/?nomobile=true#!forum/pyomo-forum
https://mobook.github.io/MO-book/intro.html

- "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

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

GAMS "Performance in Optimization Models: A Comparative Analysis of GAMS, Pyomo, GurobiPy, and JuMP" July 2023

Comparison between GAMS and Pyomo (by chatGPT)

1. GAMS has been around for longer than Pyomo, which means it has a more established user community and more extensive documentation and support.
2. GAMS is specifically designed for mathematical programming, whereas Pyomo is a general-purpose modeling tool that includes mathematical programming as one of its features.
3. GAMS provides a powerful, integrated modeling language that allows you to specify models using a concise syntax, while Pyomo requires you to write Python code to define your models.
4. GAMS has a wide range of solvers available, including commercial and open-source options, and it can seamlessly switch between solvers, whereas Pyomo requires more effort to switch between solvers.
5. GAMS includes built-in functionality for handling linear, nonlinear, and mixed-integer programming problems, whereas Pyomo requires additional libraries to handle some types of optimization problems.

1. Pyomo is open-source software, which means it is free to use and modify, while GAMS is a commercial software that requires a license to use.
2. Pyomo is based on Python, a popular general-purpose programming language that has a large and active user community, while GAMS has its own proprietary modeling language that can have a steeper learning curve.
3. Pyomo provides more flexibility than GAMS, allowing users to integrate their optimization models with other Python libraries and tools.
4. Pyomo can be used for a wide range of optimization problems, not just mathematical programming, such as stochastic programming, optimization under uncertainty, and mixed-integer nonlinear programming.
5. Pyomo has a more modular design that makes it easier to add new features or extensions, while GAMS has a more monolithic architecture that can be harder to customize.

Advantages/disadvantage of Pyomo (by ChatGPT)

1. Extensive modeling capabilities: Pyomo offers a wide range of modeling components and constructs that allow users to represent complex optimization problems in a flexible and expressive manner. It supports a variety of mathematical formulations, including linear programming (LP), mixed-integer programming (MIP), and nonlinear programming (NLP).
2. Solver compatibility: Pyomo has built-in support for interfacing with a variety of optimization solvers, both open-source and commercial. This allows users to leverage the strengths of different solvers and choose the most suitable one for their problem. Pyomo provides a unified interface for interacting with solvers, making it easy to switch between different solver backends.
3. Python integration: Pyomo is implemented in Python, a widely-used and accessible programming language. This integration with Python provides several advantages, including a large ecosystem of libraries and tools, extensive documentation and support, and the ability to leverage Python's data manipulation and analysis capabilities.
4. Scalability: Pyomo is designed to handle large-scale optimization problems efficiently. It supports parallel computation and provides options for distributed computing, enabling users to solve computationally intensive problems more effectively. Additionally, Pyomo can leverage the performance benefits of optimization solvers to handle large models and datasets.
5. Active development and community support: Pyomo benefits from an active development community, ensuring regular updates, bug fixes, and improvements. It also has an active user community that provides support, shares examples, and contributes to the development of additional libraries and extensions, making it easier for users to get started and find solutions to their optimization problems.

1. Learning curve: Pyomo's powerful modeling capabilities come with a learning curve, especially for users who are new to mathematical optimization or programming in Python. Understanding the syntax, constructs, and best practices of Pyomo may require some initial effort and familiarity with optimization modeling concepts.
2. Limited solver support: While Pyomo supports interfacing with multiple solvers, the availability of specific solvers may be limited compared to other optimization libraries. Some specialized or proprietary solvers may not have direct support in Pyomo, requiring additional effort to integrate or interface with them.
3. Performance overhead: Pyomo's flexibility and generality in modeling can sometimes come with a performance overhead. Depending on the complexity of the model and the chosen solver, Pyomo may not always achieve the same level of performance as more specialized modeling languages or libraries.
4. Documentation completeness: While Pyomo has documentation available, it may not cover all aspects of the library or provide in-depth explanations for specific modeling techniques or functionalities. Users may need to refer to external resources or rely on community support to find detailed information on certain topics.
5. Tool maturity: Pyomo, like other optimization libraries, has undergone significant development over the years. However, it may still be considered less mature compared to some other well-established optimization libraries. This means that there may be areas where Pyomo's toolset or functionality is less refined or lacks certain advanced features available in other libraries.

Creating an optimization model

- Interactive Development Environment (IDE)
  - PyCharm (https://www.jetbrains.com/pycharm/)
  - Visual Studio Code (https://code.visualstudio.com/)
  - JupyterLab: A Next-Generation Notebook Interface (https://jupyter.org/)

- Data manipulation
  - pandas - Python Data Analysis Library (https://pandas.pydata.org/)

- Plotting
  - Plotly Open Source Graphing Library for Python (https://plotly.com/python/)
  - Matplotlib: Visualization with Python (https://matplotlib.org/)
  - Vega-Altair: Declarative Visualization in Python (https://altair-viz.github.io/)

- Documentation
  - reStructuredText (https://www.sphinx-doc.org/en/master/usage/restructuredtext/index.html)
  - Sphinx makes it easy to create intelligent and beautiful documentation (https://www.sphinx-doc.org/en/master/)
  - ReadtheDocs. Build, host, and share documentation, all with a single platform (https://about.readthedocs.com/?ref=readthedocs.com)

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

1

My first example

COMILLAS

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

2

Additional Features

COMILLAS

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

3

Power Systems Models

COMILLAS

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.

COMILLAS

iit

Index

comillas.edu

COMILLAS
UNIVERSIDAD PONTIFICIA
ICAI ICADE CIHS

Good Optimizatic

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 a 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
```

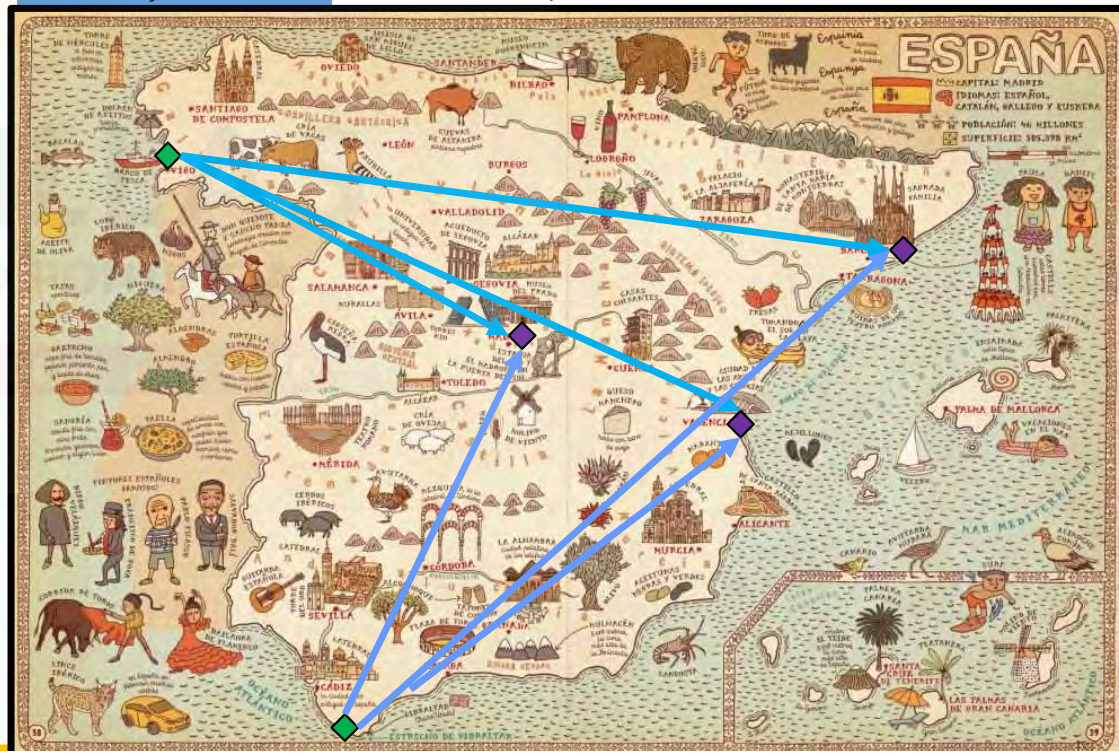
$$\min_{x_{ij}} \sum_{ij} c_{ij} x_{ij}$$

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

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

$$x_{ij} \geq 0$$

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



My first Pyomo transportation model

```
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]])
```

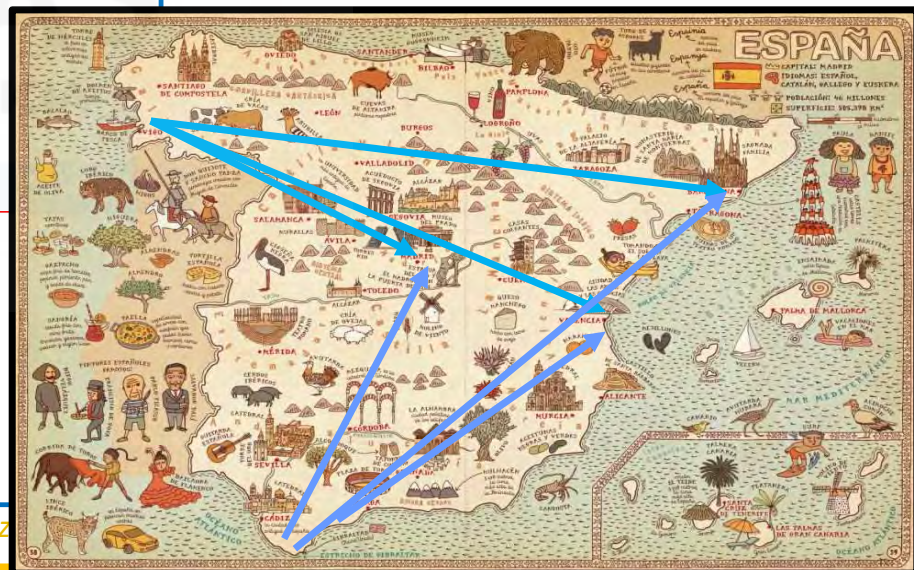
$$\min_{x_{ij}} \sum_{ij} c_{ij} x_{ij}$$

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

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

$$x_{ij} \geq 0$$

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
```

GAMSPy 0.12.2 (https://gamspy.readthedocs.io/en/latest/index.html)

- Combines the high-performance GAMS execution system with the flexible Python language, creating a powerful mathematical optimization package
- Acts as a bridge between the expressive Python language and the robust GAMS system, allowing you to create complex mathematical models effortlessly

Transportation model

```python
import pandas as pd
capacities = pd.DataFrame(
    [["seattle", 350], ["san-diego", 600]], columns=["city", "capacity"]
).set_index("city")
demands = pd.DataFrame(
    [["new-york", 325], ["chicago", 300], ["topeka", 275]], columns=["city", "demand"]
).set_index("city")
distances = pd.DataFrame(
    [
        ["seattle", "new-york", 2.5],
        ["seattle", "chicago", 1.7],
        ["seattle", "topeka", 1.8],
        ["san-diego", "new-york", 2.5],
        ["san-diego", "chicago", 1.8],
        ["san-diego", "topeka", 1.4],
    ],
    columns=["from", "to", "distance"],
).set_index(["from", "to"])
freight_cost = 90
from gamspy import Container, Set, Parameter, Variable, Equation, Model, Sum, Sense
m = Container()
i = Set(container=m, name="i", description="plants")
i.setRecords(capacities.index)
j = Set(container=m, name="j", description="markets", records=demands.index)
a = Parameter(
    container=m,
    name="a",
    domain=i,
    description="supply of commodity at plant i (in cases)",
    records=capacities.reset_index(),
)
b = Parameter(
    container=m,
    name="b",
    domain=j,
    description="demand for commodity at market j (in cases)",
    records=demands.reset_index(),
)
```

```python
c = Parameter(
    container=m,
    name="c",
    domain=[i, j],
    description="cost per unit of shipment between plant i and market j",
)
cost = freight_cost * distances / 1000
c.setRecords(cost.reset_index())
x = Variable(
    container=m,
    name="x",
    domain=[i, j],
    type="Positive",
    description="amount of commodity to ship from plant i to market j",
)
supply = Equation(
    container=m, name="supply", domain=i, description="observe supply limit at plant i"
)
demand = Equation(
    container=m, name="demand", domain=j, description="satisfy demand at market j"
)
obj = Sum((i, j), c[i, j] * x[i, j])
transport = Model(
    m,
    name="transport",
    equations=[supply, demand],
    problem="LP",
    sense=Sense.MIN,
    objective=obj,
)
import sys
transport.solve(output=sys.stdout)
x.records.set_index(["i", "j"])
transport.objective_value
```

https://gamspy.readthedocs.io/en/latest/user/notebooks/trnsport.html

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] > 0.0)
mSDUC.t = Set(initialize=mSDUC.g , ordered=False, doc='thermal    units', filter=lambda mSDUC,g : g  in mSDUC.g  and pLinearVarCost [g] > 0.0)
```

- 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')
```

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

```
# 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

```
# non-RES units, they can be committed and also contribute to the operating reserves
mTEPES.nr = mTEPES.g - mTEPES.re
# existing electric lines (le)
mTEPES.le = mTEPES.la - mTEPES.lc
# ESS and hydro units
mTEPES.eh = mTEPES.es | mTEPES.h
```

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.]')
```

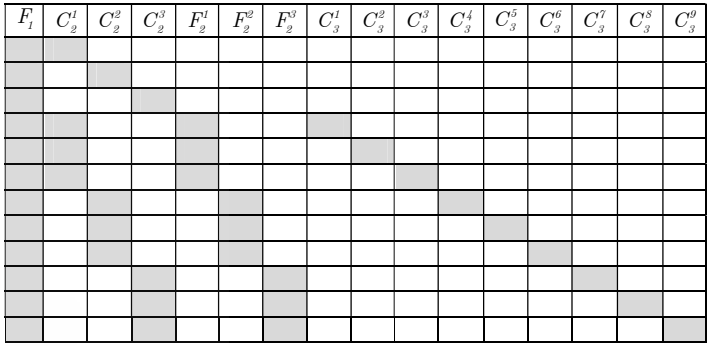
Constraint $l \leq Ax \leq u$

```
#%% 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()
```

Modeling extensions

- Auto-Persistent Pyomo Solver Interfaces (APPSI). Sensitivity analysis
- Stochastic Programming
  - Equivalent to GAMS/EMP. Formulate the deterministic problem and define the scenario tree
- Bilevel Programming
- Dynamic Optimization with pyomo.DAE
- MPEC
- Generalized Disjunctive Programming
- Pyomo Network

Preprocessing by Gurobi Python shell (`gurobipy`)

- Before and after presolve can help you detect improvements in the formulation
- Allows getting the optimization problem after the presolve

```
ModelName          = read("OriginalProblem.lp")
ModelNamePresolved = ModelName.presolve()
ModelNamePresolved.write("PresolvedProblem.lp")
```

- Gurobi Model Analyzer (`gurobi_modelanalyzer`) allows to detect numerical problems

Transportation Problem with demand scenarios
Auto-Persistent Pyomo Solver Interfaces (APPSI)

```python
import numpy               as np
import pyomo.environ       as pyo
from   pyomo.environ       import ConcreteModel, Set, Param, Var, NonNegativeReals, Constraint, Objective, minimize, Suffix
from   pyomo.opt           import SolverFactory
from   pyomo.contrib       import appsi
from   pyomo.common.timing import HierarchicalTimer

# 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
#
#    May 8, 2023

mTransport = ConcreteModel('Transportation Problem with many demand scenarios')

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', mutable=True)
mTransport.pD = Param(mTransport.j, initialize={'Madrid': 400, 'Barcelona': 450, 'Valencia': 150}, doc='destination demand', mutable=True)

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)

opt = appsi.solvers.Gurobi()
timer = HierarchicalTimer()
for p_val in np.linspace(0.8, 1):
    for j in mTransport.j:
        mTransport.pB[j] = float(p_val)*mTransport.pD[j]
    res = opt.solve(mTransport, timer=timer)
    assert res.termination_condition == appsi.base.TerminationCondition.optimal
    print(mTransport.eCost.expr())
print(timer)
```

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

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

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

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

$$x_{ij} \le M_{ij}y_{ij} \quad \forall ij$$

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

- Bd Relaxed Master

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

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

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

O.F. of the subproblem at iteration l

Dual variables of linking constraints at iteration l

Master proposal at iteration l

- Bd Subproblem

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

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

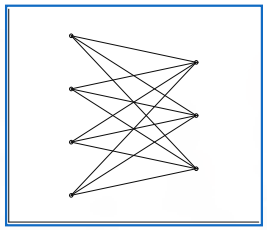
$$\sum_{i} x_{ij} \ge b_j \quad \forall j$$

$$x_{ij} \le M_{ij}y_{ij}^k \quad \forall ij \quad : \pi_{ij}^k$$

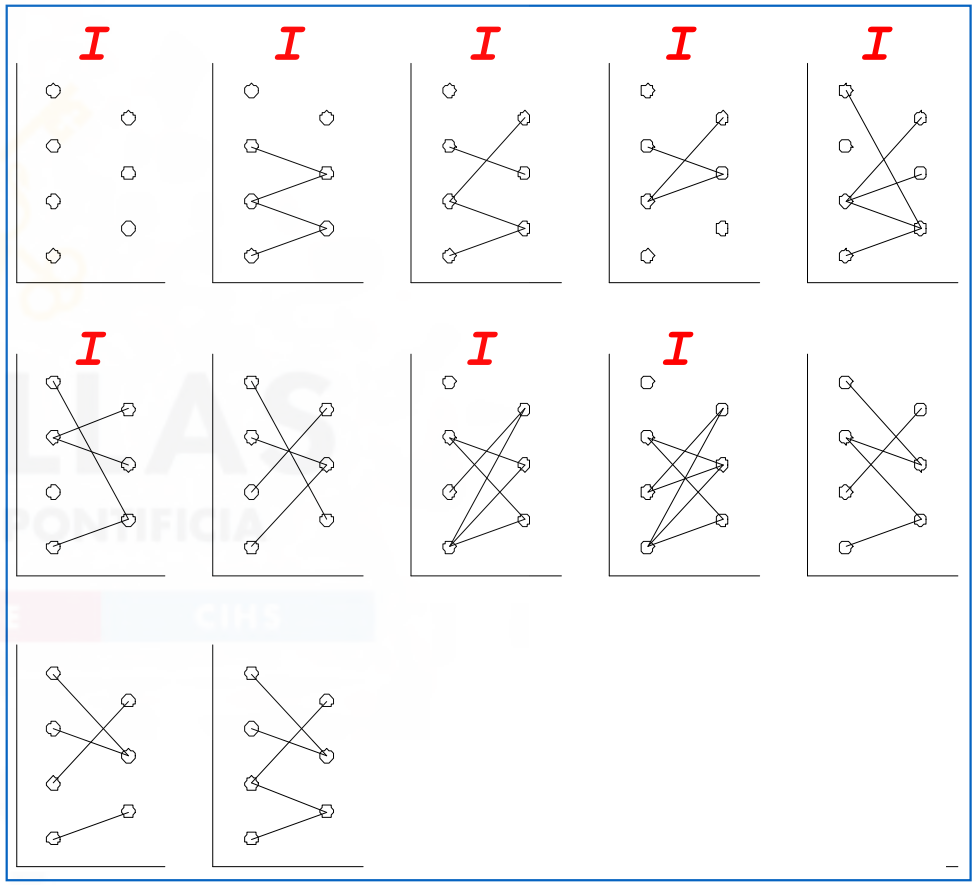
$$x_{ij} \ge 0$$

Fixed-Charge Transportation Problem. Bd Solution

- Possible arcs

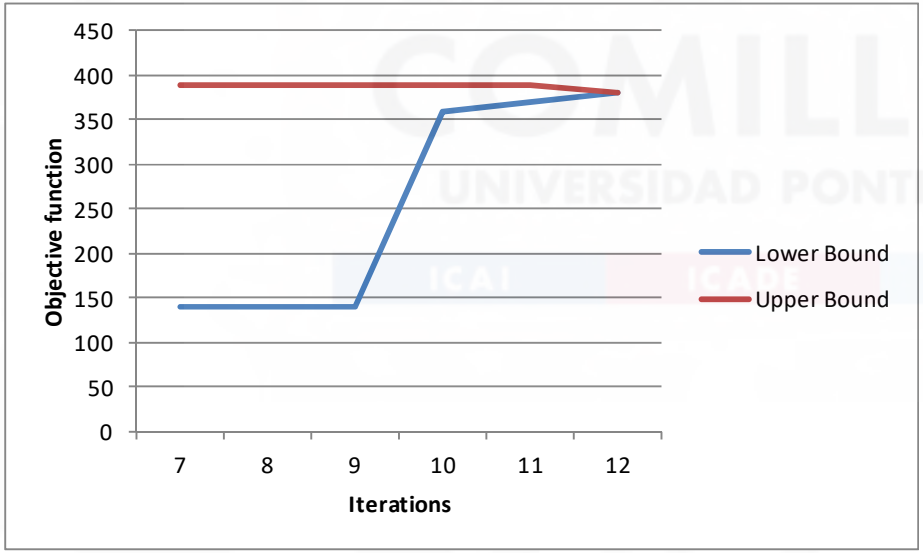


- Solutions along Benders decomposition iterations



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)

https://github.com/IIT-EnergySystemModels/Fixed-Charge-Transportation-Problem-Benders-Decomposition

```
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)

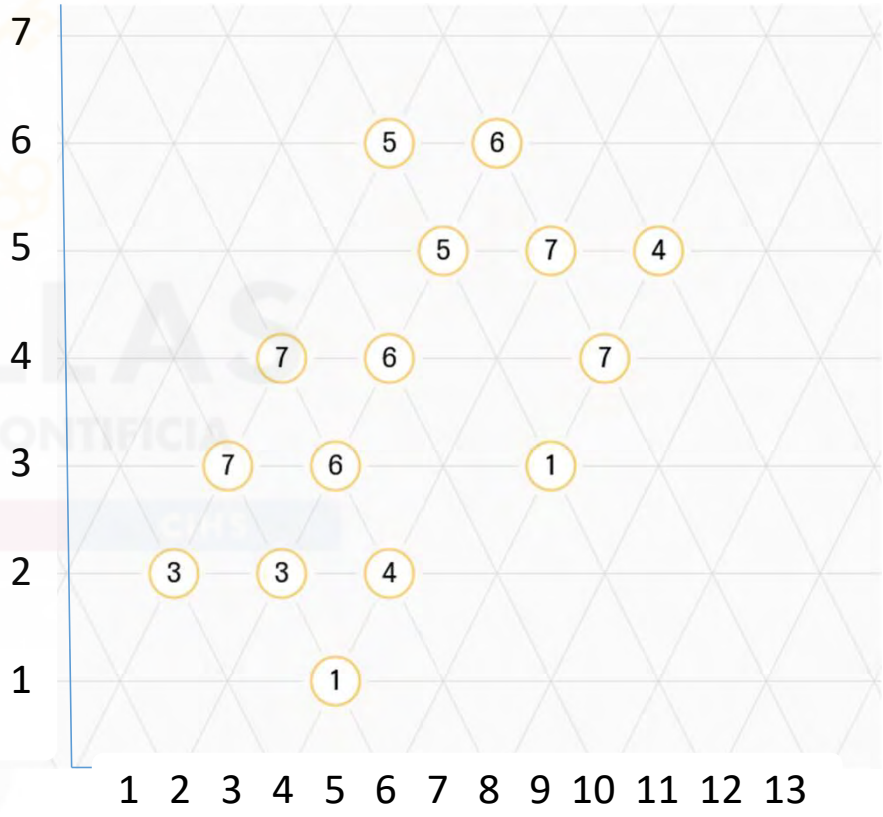
```
mFCTP.pF      = Param(mFCTP.i, mFCTP.j, initialize=FixedCost,          doc='fixed investment cost'       )
mFCTP.pC      = Param(mFCTP.i, mFCTP.j, initialize=TransportationCost, doc='per unit transportation cost')

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()
```

Hashi puzzle (https://en.wikipedia.org/wiki/Hashiwokakero)

Connect bridges between islands, according to their values, to form one interconnecting path

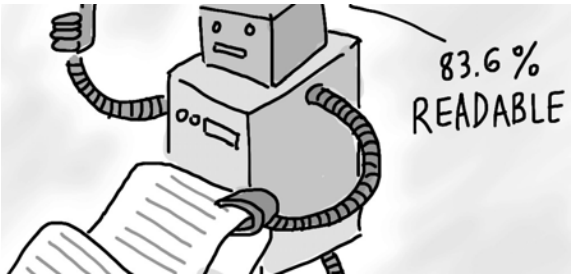


```python
# Developed by
#    Andres Ramos
#    Instituto de Investigacion Tecnologica
#    Escuela Tecnica Superior de Ingenieria - ICAI
#    UNIVERSIDAD PONTIFICIA COMILLAS
#    Alberto Aguilera 23
#    28015 Madrid, Spain
#    Andres.Ramos@comillas.edu

import pandas as pd
from pyomo.environ import ConcreteModel, NonNegativeIntegers, Set, Param, Var, Constraint, Objective, minimize
from pyomo.opt     import SolverFactory
from collections   import defaultdict

SolverName     = 'gurobi'

# model declaration
mHP = ConcreteModel('Hashi puzzle')


def plain_run(mHP, SolverName):

    mHP.x = Set(initialize=['x00', 'x01', 'x02', 'x03', 'x04', 'x05', 'x06', 'x07', 'x08', 'x09', 'x10', 'x11'], ordered=True, doc='abscises' )
    mHP.y = Set(initialize=['y00', 'y01', 'y02', 'y03', 'y04', 'y05', 'y06', 'y07', 'y08', 'y09', 'y10', 'y11'], ordered=True, doc='ordinates')

    Nodes = {
        ('x05', 'y01'): 1,
        ('x02', 'y02'): 3,
        ('x04', 'y02'): 3,
        ('x06', 'y02'): 4,
        ('x03', 'y03'): 7,
        ('x05', 'y03'): 6,
        ('x09', 'y03'): 1,
        ('x04', 'y04'): 7,
        ('x06', 'y04'): 6,
        ('x10', 'y04'): 7,
        ('x07', 'y05'): 5,
        ('x09', 'y05'): 7,
        ('x11', 'y05'): 4,
        ('x06', 'y06'): 5,
        ('x08', 'y06'): 6,
    }

    Arcs = set()
    Neighbors = defaultdict(list)
    for x,y,xx,yy in mHP.x*mHP.y*mHP.x*mHP.y:
        if (x,y) in Nodes and (xx,yy) in Nodes:
            if mHP.x.ord(x) > 2 and mHP.x.ord(x) < len(mHP.x)-2 and mHP.y.ord(y) > 1 and mHP.y.ord(y) < len(mHP.y)-1:
                if ((xx,yy) == (mHP.x.next(x),mHP.y.next(y)) or (xx,yy) == (mHP.x.next(x,2),y) or (xx,yy) == (mHP.x.next(x),mHP.y.prev(y)) or
                    (xx,yy) == (mHP.x.prev(x),mHP.y.prev(y)) or (xx,yy) == (mHP.x.prev(x,2),y) or (xx,yy) == (mHP.x.prev(x),mHP.y.next(y))):
                    # add the arc and its neighbors to the list
                    Arcs.add((x, y, xx, yy))
                    Arcs.add((xx, yy, x, y))
                    Neighbors[x,y].append((xx,yy))

    # parameters
    mHP.pNodes = Param(mHP.x, mHP.y, initialize=Nodes, doc='number of connections per node')

    # Variables
    mHP.vConnection = Var(Arcs, within=NonNegativeIntegers, doc='connections between two neighbor nodes')

    def eTotalConnections(mHP):
        return sum(mHP.vConnection[x,y,xx,yy] for x,y,xx,yy in Arcs)
    mHP.eTotalConnections = Objective(rule=eTotalConnections, sense=minimize, doc='total number of connections')

    def eConnections(mHP, x, y):
        if mHP.x.ord(x) > 2 and mHP.x.ord(x) < len(mHP.x)-2 and mHP.y.ord(y) > 1 and mHP.y.ord(y) < len(mHP.y)-1:
            return sum(mHP.vConnection[x,y,xx,yy] + mHP.vConnection[xx,yy,x,y] for xx,yy in Neighbors[x,y]) == mHP.pNodes[x,y]
        else:
            return Constraint.Skip
    mHP.eConnections = Constraint(Nodes, rule=eConnections, doc='connections on every node')

    Solver = SolverFactory(SolverName)
    mHP.write('connection.lp', io_options={'symbolic_solver_labels': True})
    SolverResults = Solver.solve(mHP, tee=True)
    SolverResults.write()  # summary of the solver results
    mHP.vConnection.pprint()

if __name__ == "__main__":
    plain_run(mHP, SolverName)
```

3

Power Systems Models

Simplicity and transparency

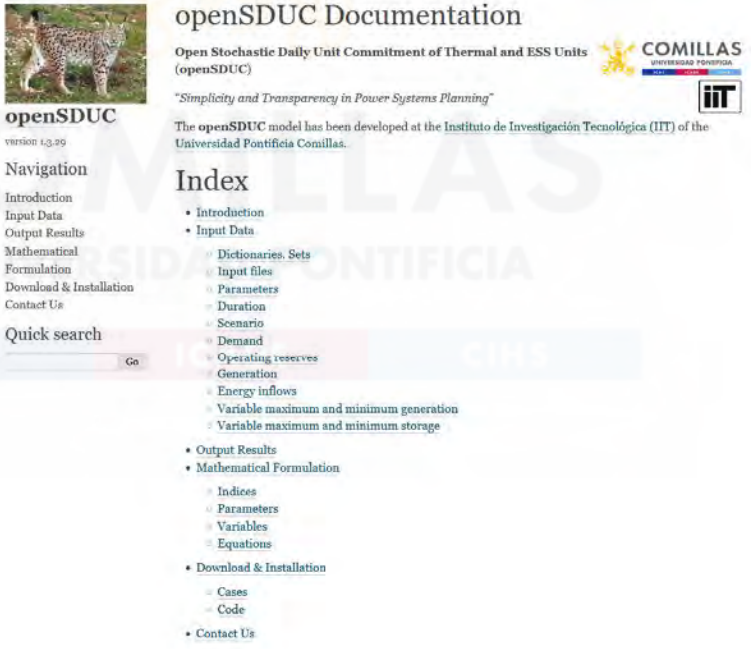


- Replicating the GAMS structure and elegance in Python/Pyomo
- Separation between
  - Dictionaries of sets
  - Parameters
  - Variables
  - Equations
  - Solve
  - Output results
- Input data and output results in text format (csv)

openSDUC

https://opensduc.readthedocs.io/en/latest/index.html
https://github.com/IIT-EnergySystemModels/openSDUC

- 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.

Stochastic Daily Unit Commitment (i)

```
# Open Stochastic Daily Unit Commitment of Thermal and ESS Units (openSDUC) - Version 1.3.29 - January 22, 2023
# 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 argparse
import os
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

print('\n #### Academic research license - for non-commercial use only #### \n')

StartTime = time.time()

parser = argparse.ArgumentParser(description='Introducing main parameters...')
parser.add_argument('--case',   type=str, default=None)
parser.add_argument('--dir',    type=str, default=None)
parser.add_argument('--solver', type=str, default=None)

DIR    = os.path.dirname(__file__)
CASE   = '16g'
SOLVER = 'gurobi'
```

Stochastic Daily Unit Commitment (ii)

```
def main():
    args = parser.parse_args()
    if args.dir is None:
        args.dir    = input('Input Dir    Name (Default {}): '.format(DIR))
        if args.dir == '':
            args.dir = DIR
    if args.case is None:
        args.case   = input('Input Case   Name (Default {}): '.format(CASE))
        if args.case == '':
            args.case = CASE
    if args.solver is None:
        args.solver = input('Input Solver Name (Default {}): '.format(SOLVER))
        if args.solver == '':
            args.solver = SOLVER
    print(args.case)
    print(args.dir)
    print(args.solver)
    import sys
    print(sys.argv)
    print(args)
    openSDUC_run(args.dir, args.case, args.solver)
```

Stochastic Daily Unit Commitment (iii)

```python
def openSDUC_run(DirName, CaseName, SolverName):

    _path = os.path.join(DirName, CaseName)
    StartTime = time.time()

    #%% model declaration
    mSDUC = ConcreteModel('Open Stochastic Daily Unit Commitment of Thermal and ESS Units (openSDUC) - Version 1.3.29 - January 22, 2023')

    #%% reading the sets
    dictSets = DataPortal()
    dictSets.load(filename=_path+'/oUC_Dict_Scenario_'  +CaseName+'.csv', set='sc', format='set')
    dictSets.load(filename=_path+'/oUC_Dict_LoadLevel_' +CaseName+'.csv', set='n' , format='set')
    dictSets.load(filename=_path+'/oUC_Dict_Generation_'+CaseName+'.csv', set='g' , format='set')
    dictSets.load(filename=_path+'/oUC_Dict_Storage_'   +CaseName+'.csv', set='st', format='set')
    dictSets.load(filename=_path+'/oUC_Dict_Technology_'+CaseName+'.csv', set='gt', format='set')
    dictSets.load(filename=_path+'/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(_path+'/oUC_Data_Parameter_'        +CaseName+'.csv', index_col=[0  ])
    dfDuration           = pd.read_csv(_path+'/oUC_Data_Duration_'         +CaseName+'.csv', index_col=[0  ])
    dfScenario           = pd.read_csv(_path+'/oUC_Data_Scenario_'         +CaseName+'.csv', index_col=[0  ])
    dfDemand             = pd.read_csv(_path+'/oUC_Data_Demand_'           +CaseName+'.csv', index_col=[0,1])
    dfOperatingReserve   = pd.read_csv(_path+'/oUC_Data_OperatingReserve_' +CaseName+'.csv', index_col=[0,1])
    dfGeneration         = pd.read_csv(_path+'/oUC_Data_Generation_'       +CaseName+'.csv', index_col=[0  ])
    dfVariableMinPower   = pd.read_csv(_path+'/oUC_Data_MinimumGeneration_'+CaseName+'.csv', index_col=[0,1])
    dfVariableMaxPower   = pd.read_csv(_path+'/oUC_Data_MaximumGeneration_'+CaseName+'.csv', index_col=[0,1])
    dfVariableMinStorage = pd.read_csv(_path+'/oUC_Data_MinimumStorage_'   +CaseName+'.csv', index_col=[0,1])
    dfVariableMaxStorage = pd.read_csv(_path+'/oUC_Data_MaximumStorage_'   +CaseName+'.csv', index_col=[0,1])
    dfEnergyInflows      = pd.read_csv(_path+'/oUC_Data_EnergyInflows_'    +CaseName+'.csv', index_col=[0,1])
```

Stochastic Daily Unit Commitment (iv)

```
    # 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)
    dfVariableMinPower.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)

    #%% 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]
    pVariableMinPower   = dfVariableMinPower  [list(mSDUC.gg)] * 1e-3                        # dynamic variable minimum power   [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()
    pVariableMinPower   = pVariableMinPower.rolling (pTimeStep).mean()
    pVariableMaxPower   = pVariableMaxPower.rolling (pTimeStep).mean()
    pVariableMinStorage = pVariableMinStorage.rolling(pTimeStep).mean()
    pVariableMaxStorage = pVariableMaxStorage.rolling(pTimeStep).mean()
    pEnergyInflows      = pEnergyInflows.rolling    (pTimeStep).mean()
```

Stochastic Daily Unit Commitment (v)

```
pDemand.fillna            (0.0, inplace=True)
pOperReserveUp.fillna     (0.0, inplace=True)
pOperReserveDw.fillna     (0.0, inplace=True)
pVariableMinPower.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)

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

#%% 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
```

Stochastic Daily Unit Commitment (vi)

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

#%% 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

if pTimeStep > 1:
    # drop levels with duration 0
    pDuration           = pDuration.loc          [mSDUC.sc*mSDUC.n]
    pDemand             = pDemand.loc            [mSDUC.sc*mSDUC.n]
    pOperReserveUp      = pOperReserveUp.loc     [mSDUC.sc*mSDUC.n]
    pOperReserveDw      = pOperReserveDw.loc     [mSDUC.sc*mSDUC.n]
    pVariableMinPower   = pVariableMinPower.loc  [mSDUC.sc*mSDUC.n]
    pVariableMaxPower   = pVariableMaxPower.loc  [mSDUC.sc*mSDUC.n]
    pVariableMinStorage = pVariableMinStorage.loc[mSDUC.sc*mSDUC.n]
    pVariableMaxStorage = pVariableMaxStorage.loc[mSDUC.sc*mSDUC.n]
    pEnergyInflows      = pEnergyInflows.loc     [mSDUC.sc*mSDUC.n]

# variable minimum and maximum power
pVariableMinPower = pVariableMinPower.replace(0.0, float('nan'))
pVariableMaxPower = pVariableMaxPower.replace(0.0, float('nan'))
pMinPower         = pd.DataFrame([pRatedMinPower]*len(pVariableMaxPower.index), index=pd.MultiIndex.from_tuples(pVariableMaxPower.index), columns=pRatedMinPower.index)
pMaxPower         = pd.DataFrame([pRatedMaxPower]*len(pVariableMaxPower.index), index=pd.MultiIndex.from_tuples(pVariableMaxPower.index), columns=pRatedMaxPower.index)
pMinPower         = pMinPower.reindex       (sorted(pMinPower.columns       ), axis=1)
pMaxPower         = pMaxPower.reindex       (sorted(pMaxPower.columns       ), axis=1)
pVariableMinPower = pVariableMinPower.reindex(sorted(pVariableMinPower.columns), axis=1)
pVariableMaxPower = pVariableMaxPower.reindex(sorted(pVariableMaxPower.columns), axis=1)
pMinPower         = pVariableMinPower.where(pVariableMinPower > pMinPower, other=pMaxPower)
pMaxPower         = pVariableMaxPower.where(pVariableMaxPower < pMaxPower, other=pMaxPower)
pMaxPower2ndBlock = pMaxPower - pMinPower
```

Stochastic Daily Unit Commitment (vii)

```
# variable minimum and maximum storage capacity
   pVariableMinStorage = pVariableMinStorage.replace(0.0, float('nan'))
   pVariableMaxStorage = pVariableMaxStorage.replace(0.0, float('nan'))
   pMinStorage         = pd.DataFrame([pRatedMinStorage]*len(pVariableMinStorage.index), index=pd.MultiIndex.from_tuples(pVariableMinStorage.index), columns=pRatedMinStorage.index)
   pMaxStorage         = pd.DataFrame([pRatedMaxStorage]*len(pVariableMaxStorage.index), index=pd.MultiIndex.from_tuples(pVariableMaxStorage.index), columns=pRatedMaxStorage.index)
   pMinStorage         = pMinStorage.reindex        (sorted(pMinStorage.columns        ), axis=1)
   pMaxStorage         = pMaxStorage.reindex        (sorted(pMaxStorage.columns        ), axis=1)
   pVariableMinStorage = pVariableMinStorage.reindex(sorted(pVariableMinStorage.columns), axis=1)
   pVariableMaxStorage = pVariableMaxStorage.reindex(sorted(pVariableMaxStorage.columns), axis=1)
   pMinStorage         = pVariableMinStorage.where(pVariableMinStorage > pMinStorage, other=pMinStorage)
   pMaxStorage         = pVariableMaxStorage.where(pVariableMaxStorage < pMaxStorage, other=pMaxStorage)

   # values < 1e-5 times the maximum system demand are converted to 0
   pEpsilon = pDemand.max()*1e-5
   # 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 downtime 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]
```

Stochastic Daily Unit Commitment (viii)

```
#%% 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] =        1
    if  pStorageType[es] == 'Weekly'  :
        pCycleTimeStep[es] = int(  24/pTimeStep)
    if  pStorageType[es] == 'Monthly' :
        pCycleTimeStep[es] = int( 168/pTimeStep)

# 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')
```

Stochastic Daily Unit Commitment (ix)

```
    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')

    #%% 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')
```

Stochastic Daily Unit Commitment (x)

```python
#%%
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')
```

Stochastic Daily Unit Commitment (xi)

```
#%%
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(_path+'/openSDUC_'+CaseName+'.lp', io_options={'symbolic_solver_labels': True}) # create lp-format file
Solver = SolverFactory(SolverName)                                                          # select solver
if SolverName == 'gurobi':
    Solver.options['LogFile'       ] = _path+'/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,nr in mSDUC.n*mSDUC.nr:
    mSDUC.vCommitment[n,nr].fix(round(mSDUC.vCommitment[n,nr]()))
    mSDUC.vStartUp   [n,nr].fix(round(mSDUC.vStartUp   [n,nr]()))
    mSDUC.vShutDown  [n,nr].fix(round(mSDUC.vShutDown  [n,nr]()))

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())
```

Stochastic Daily Unit Commitment (xii)

```
    #%% inverse index generator to technology and to company
    pTechnologyToGen = pGenToTechnology.reset_index().set_index('Technology').set_axis(['Generator'], axis=1, copy=False)[['Generator']]
    pTechnologyToGen = pTechnologyToGen.loc[pTechnologyToGen['Generator'].isin(mSDUC.g)]
    pTechnology2Gen  = pTechnologyToGen.reset_index().set_index(['Technology', 'Generator'])

    mSDUC.t2g = Set(initialize=pTechnology2Gen.index, ordered=False, doc='technology to generator')

    #%% outputting the generation operation

    OutputResults = pd.Series(data=[mSDUC.vCommitment[n,nr]() for n,nr in mSDUC.n*mSDUC.nr], index=pd.MultiIndex.from_tuples(list(mSDUC.n*mSDUC.nr)))
    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(_path+'/oUC_Result_GenerationCommitment_'+CaseName+'.csv', sep=',')
    OutputResults = pd.Series(data=[mSDUC.vStartUp   [n,nr]() for n,nr in mSDUC.n*mSDUC.nr], index=pd.MultiIndex.from_tuples(list(mSDUC.n*mSDUC.nr)))
    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(_path+'/oUC_Result_GenerationStartUp_'    +CaseName+'.csv', sep=',')
    OutputResults = pd.Series(data=[mSDUC.vShutDown  [n,nr]() for n,nr in mSDUC.n*mSDUC.nr], index=pd.MultiIndex.from_tuples(list(mSDUC.n*mSDUC.nr)))
    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(_path+'/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(_path+'/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(_path+'/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(_path+'/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(_path+'/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(_path+'/oUC_Result_ENS_'+CaseName+'.csv', sep=',')

    OutputResults = pd.Series(data=[(mSDUC.vTotalOutput[sc,n,g].ub-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(_path+'/oUC_Result_RESCurtailment_'+CaseName+'.csv', sep=',')
```

Stochastic Daily Unit Commitment (xiii)

```
    #%% 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(_path+'/oUC_Plot_RESCurtailment_'+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.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(_path+'/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(_path+'/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(_path+'/oUC_Result_ESSChargeOutput_'+CaseName+'.csv', sep=',')

        OutputResults = pd.Series(data=[sum(OutputResults[sc,n,es] for es in mSDUC.es if (gt,es) in mSDUC.t2g) for sc,n,gt in mSDUC.sc*mSDUC.n*mSDUC.gt if sum(1 for es in mSDUC.es if (gt,es) in mSDUC.t2g)], index=pd.MultiIndex.from_tuples([(sc,n,gt) for
sc,n,gt in mSDUC.sc*mSDUC.n*mSDUC.gt if sum(1 for es in mSDUC.es if (gt,es) in mSDUC.t2g)]))
        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(_path+'/oUC_Result_TechnologyCharge_'+CaseName+'.csv', sep=',')

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

        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(_path+'/oUC_Result_ESSChargeEnergy_'+CaseName+'.csv', sep=',')

        OutputResults = pd.Series(data=[sum(OutputResults[sc,n,es] for es in mSDUC.es if (gt,es) in mSDUC.t2g) for sc,n,gt in mSDUC.sc*mSDUC.n*mSDUC.gt if sum(1 for es in mSDUC.es if (gt,es) in mSDUC.t2g)], index=pd.MultiIndex.from_tuples([(sc,n,gt) for
sc,n,gt in mSDUC.sc*mSDUC.n*mSDUC.gt if sum(1 for es in mSDUC.es if (gt,es) in mSDUC.t2g)]))
        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(_path+'/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.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(_path+'/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.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(_path+'/oUC_Result_ESSSpillage_'+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 = pd.Series(data=[sum(OutputResults[sc,n,g] for g  in mSDUC.g  if (gt,g ) in mSDUC.t2g)      for sc,n,gt in mSDUC.sc*mSDUC.n*mSDUC.gt if sum(1 for g in mSDUC.g if (gt,g) in mSDUC.t2g)], index=pd.MultiIndex.from_tuples([(sc,n,gt) for sc,n,gt
in mSDUC.sc*mSDUC.n*mSDUC.gt if sum(1 for g in mSDUC.g if (gt,g) in mSDUC.t2g)]))
    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(_path+'/oUC_Result_TechnologyOutput_'+CaseName+'.csv',
sep=',')

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

Stochastic Daily Unit Commitment (xiv)

```
    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(), baseline='zero', 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(_path+'/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=[sum(OutputResults[sc,n,g] for g in mSDUC.g if (gt,g) in mSDUC.t2g) for sc,n,gt in mSDUC.sc*mSDUC.n*mSDUC.gt if sum(1 for g in mSDUC.g if (gt,g) in mSDUC.t2g)], index=pd.MultiIndex.from_tuples([(sc,n,gt) for sc,n,gt in
mSDUC.sc*mSDUC.n*mSDUC.gt if sum(1 for g in mSDUC.g if (gt,g) in mSDUC.t2g)]))
    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(_path+'/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(_path+'/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(_path+'/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')
    print('\n #### Academic research license - for non-commercial use only #### \n')

if __name__ == '__main__':
    main()
```

JuanitoJaimitoJorgito

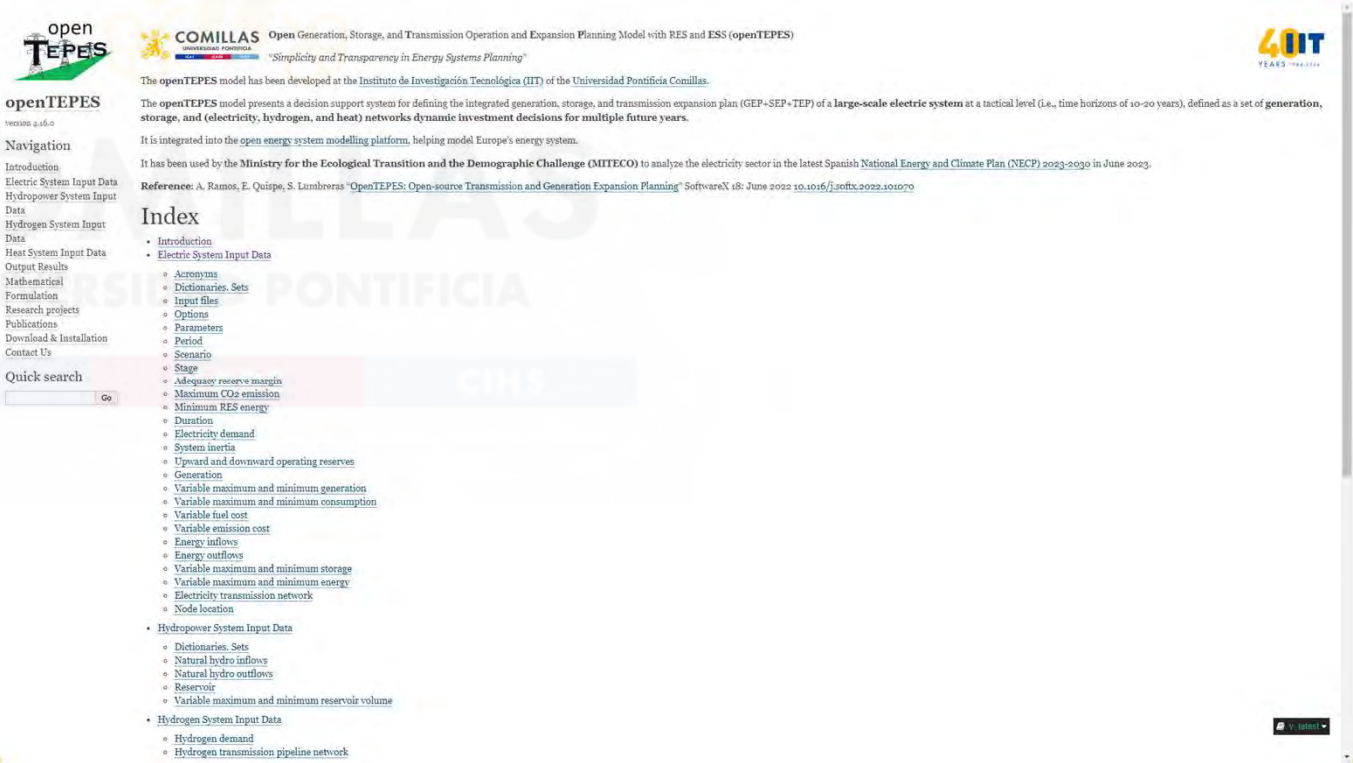
https://gitlab001.iit.comillas.edu/pdeotaola/Ejemplo_Optimizacion_Python-Pyomo

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

openTEPES

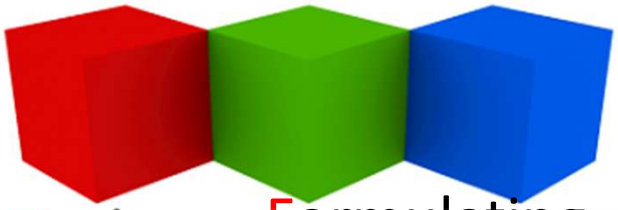
https://opentepes.readthedocs.io/en/latest/index.html
https://github.com/IIT-EnergySystemModels/openTEPES

- 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.



Formulating, writing and solving optimization models

Thank you for your attention

Prof. Andres Ramos

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

Andres.Ramos@comillas.edu

arght@mit.edu