1.2 Running Pyomo on Google Colab

This note notebook shows how to install the basic pyomo package on Google Colab, and then demonstrates the subsequent installation and use of various solvers including

- GLPK
- COIN-OR CBC
- COIN-OR Ipopt
- COIN-OR Bonmin
- COIN-OR Couenne
- COIN-OR Gecode

Basic Installation of Pyomo

We'll do a quiet installation of pyomo using `pip`. This needs to be done once at the start of each Colab session.

In [1]:

```
!pip install -q pyomo
```

```
     100% |████████████████████████████████| 2.1MB 12.6MB/s
     100% |████████████████████████████████| 51kB 19.4MB/s
     100% |████████████████████████████████| 256kB 28.6MB/s
```

The installation of pyomo can be verified by entering a simple model. We'll use the model again in subsequent cells to demonstrate the installation and execution of various solvers.

In [2]:

```
from pyomo.environ import *

# create a model
model = ConcreteModel()

# declare decision variables
model.x = Var(domain=NonNegativeReals)
model.y = Var(domain=NonNegativeReals)

# declare objective
model.profit = Objective(expr = 40*model.x + 30*model.y, sense=maximize)

# declare constraints
model.demand = Constraint(expr = model.x <= 40)
model.laborA = Constraint(expr = model.x + model.y <= 80)
model.laborB = Constraint(expr = 2*model.x + model.y <= 100)

model.pprint()
```

```
2 Var Declarations
    x : Size=1, Index=None
        Key  : Lower : Value : Upper : Fixed : Stale : Domain
        None :     0 :  None :  None : False :  True : NonNegativeReals
    y : Size=1, Index=None
        Key  : Lower : Value : Upper : Fixed : Stale : Domain
        None :     0 :  None :  None : False :  True : NonNegativeReals

1 Objective Declarations
    profit : Size=1, Index=None, Active=True
        Key  : Active : Sense    : Expression
        None :   True : maximize : 40*x + 30*y

3 Constraint Declarations
    demand : Size=1, Index=None, Active=True
        Key  : Lower : Body : Upper : Active
        None :  -Inf :    x :  40.0 :   True
    laborA : Size=1, Index=None, Active=True
        Key  : Lower : Body  : Upper : Active
        None :  -Inf : x + y :  80.0 :   True
    laborB : Size=1, Index=None, Active=True
        Key  : Lower : Body    : Upper : Active
        None :  -Inf : 2*x + y : 100.0 :   True

6 Declarations: x y profit demand laborA laborB
```

GLPK Installation

GLPK is a the open-source **G**NU **L**inear **P**rogramming **K**it available for use under the GNU General Public License 3. GLPK is a single-threaded simplex solver generally suited to small to medium scale linear-integer programming problems. It is written in C with minimal dependencies and is therefore highly portable among computers and operating systems. GLPK is often 'good enough' for many examples. For larger problems users should consider higher-performance solvers, such as COIN-OR CBC, that can take advantage of multi-threaded processors.

In [ ]:

```
!apt-get install -y -qq glpk-utils
```

In [5]:

```
SolverFactory('glpk', executable='/usr/bin/glpsol').solve(model).write()

# display solution
print('\nProfit = ', model.profit())

print('\nDecision Variables')
print('x = ', model.x())
print('y = ', model.y())

print('\nConstraints')
print('Demand  = ', model.demand())
print('Labor A = ', model.laborA())
print('Labor B = ', model.laborB())
```

```
# ==========================================================
# = Solver Results                                         =
# ==========================================================
# ----------------------------------------------------------
#   Problem Information
# ----------------------------------------------------------
Problem:
- Name: unknown
  Lower bound: 2600.0
  Upper bound: 2600.0
  Number of objectives: 1
  Number of constraints: 4
  Number of variables: 3
  Number of nonzeros: 6
  Sense: maximize
# ----------------------------------------------------------
#   Solver Information
# ----------------------------------------------------------
Solver:
- Status: ok
  Termination condition: optimal
  Statistics:
    Branch and bound:
      Number of bounded subproblems: 0
      Number of created subproblems: 0
  Error rc: 0
  Time: 0.020076274871826172
# ----------------------------------------------------------
#   Solution Information
# ----------------------------------------------------------
Solution:
- number of solutions: 0
  number of solutions displayed: 0

Profit =  2600.0

Decision Variables
x =  20.0
y =  60.0

Constraints
Demand  =  20.0
Labor A =  80.0
Labor B =  100.0
```

COIN-OR CBC Installation

COIN-OR CBC is a multi-threaded open-source **C**oin-or **b**ranch and **c**ut mixed-integer linear programming solver written in C++ under the Eclipse Public License (EPL). CBC is generally a good choice for a general purpose MILP solver for medium to large scale problems.

In [ ]:

```
!apt-get install -y -qq coinor-cbc
```

In [7]:

```
SolverFactory('cbc', executable='/usr/bin/cbc').solve(model).write()

# display solution
print('\nProfit = ', model.profit())

print('\nDecision Variables')
print('x = ', model.x())
print('y = ', model.y())

print('\nConstraints')
print('Demand  = ', model.demand())
print('Labor A = ', model.laborA())
print('Labor B = ', model.laborB())
```

```
# ==========================================================
# = Solver Results                                         =
# ==========================================================
# ----------------------------------------------------------
#   Problem Information
# ----------------------------------------------------------
Problem:
- Name: unknown
  Lower bound: -2600.0
  Upper bound: -2600.0
  Number of objectives: 1
  Number of constraints: 4
  Number of variables: 3
  Number of nonzeros: 6
  Sense: minimize
# ----------------------------------------------------------
#   Solver Information
# ----------------------------------------------------------
Solver:
- Status: ok
  User time: -1.0
  Termination condition: optimal
  Error rc: 0
  Time: 0.019547700881958008
# ----------------------------------------------------------
#   Solution Information
# ----------------------------------------------------------
Solution:
- number of solutions: 0
  number of solutions displayed: 0

Profit =  2600.0

Decision Variables
x =  20.0
y =  60.0

Constraints
Demand  =  20.0
Labor A =  80.0
Labor B =  100.0
```

COIN-OR Ipopt Installation

COIN-OR Ipopt is an open-source **I**nterior **P**oint **Opt**imizer for large-scale nonlinear optimization available under the Eclipse Public License (EPL). It is well-suited to solving nonlinear programming problems without integer or binary constraints.

In [ ]:

```
!wget -N -q "https://ampl.com/dl/open/ipopt/ipopt-linux64.zip"
!unzip -o -q ipopt-linux64
```

In [9]:

```
SolverFactory('ipopt', executable='/content/ipopt').solve(model).write()

# display solution
print('\nProfit = ', model.profit())

print('\nDecision Variables')
print('x = ', model.x())
print('y = ', model.y())

print('\nConstraints')
print('Demand  = ', model.demand())
print('Labor A = ', model.laborA())
print('Labor B = ', model.laborB())
```

```
# ==========================================================
# = Solver Results                                         =
# ==========================================================
# ----------------------------------------------------------
#   Problem Information
# ----------------------------------------------------------
Problem:
- Lower bound: -inf
  Upper bound: inf
  Number of objectives: 1
  Number of constraints: 3
  Number of variables: 2
  Sense: unknown
# ----------------------------------------------------------
#   Solver Information
# ----------------------------------------------------------
Solver:
- Status: ok
  Message: Ipopt 3.12.8\x3a Optimal Solution Found
  Termination condition: optimal
  Id: 0
  Error rc: 0
  Time: 0.022800445556640625
# ----------------------------------------------------------
#   Solution Information
# ----------------------------------------------------------
Solution:
- number of solutions: 0
  number of solutions displayed: 0

Profit =  2600.000025994988

Decision Variables
x =  20.0000001998747
y =  60.0000006

Constraints
Demand  =  20.0000001998747
Labor A =  80.0000007998747
Labor B =  100.0000009997494
```

COIN-OR Bonmin Installation

COIN-OR Bonmin is a **b**asic **o**pen-source solver for **n**onlinear **m**ixed-**in**teger programming problems (MINLP). It utilizes CBC and Ipopt for solving relaxed subproblems.

In [ ]:

```
!wget -N -q "https://ampl.com/dl/open/bonmin/bonmin-linux64.zip"
!unzip -o -q bonmin-linux64
```

In [11]:

```
SolverFactory('bonmin', executable='/content/bonmin').solve(model).write()

# display solution
print('\nProfit = ', model.profit())

print('\nDecision Variables')
print('x = ', model.x())
print('y = ', model.y())

print('\nConstraints')
print('Demand  = ', model.demand())
print('Labor A = ', model.laborA())
print('Labor B = ', model.laborB())
```

```
# ==========================================================
# = Solver Results                                         =
# ==========================================================
# ----------------------------------------------------------
#   Problem Information
# ----------------------------------------------------------
Problem:
- Lower bound: -inf
  Upper bound: inf
  Number of objectives: 1
  Number of constraints: 0
  Number of variables: 2
  Sense: unknown
# ----------------------------------------------------------
#   Solver Information
# ----------------------------------------------------------
Solver:
- Status: ok
  Message: bonmin\x3a Optimal
  Termination condition: optimal
  Id: 3
  Error rc: 0
  Time: 0.025995254516601562
# ----------------------------------------------------------
#   Solution Information
# ----------------------------------------------------------
Solution:
- number of solutions: 0
  number of solutions displayed: 0

Profit =  2600.0000259999797

Decision Variables
x =  20.000000199999512
y =  60.00000059999998

Constraints
Demand  =  20.000000199999512
Labor A =  80.0000007999995
Labor B =  100.000000999999
```

COIN-OR Couenne Installation

COIN-OR Couenne](https://www.coin-or.org/Couenne/) is attempts to find global optima for mixed-integer nonlinear programming problems (MINLP).

In [ ]:

```
!wget -N -q "https://ampl.com/dl/open/couenne/couenne-linux64.zip"
!unzip -o -q couenne-linux64
```

In [13]:

```
SolverFactory('couenne', executable='/content/couenne').solve(model).write()

# display solution
print('\nProfit = ', model.profit())

print('\nDecision Variables')
print('x = ', model.x())
print('y = ', model.y())

print('\nConstraints')
print('Demand  = ', model.demand())
print('Labor A = ', model.laborA())
print('Labor B = ', model.laborB())
```

```
# ==========================================================
# = Solver Results                                         =
# ==========================================================
# ----------------------------------------------------------
#   Problem Information
# ----------------------------------------------------------
Problem:
- Lower bound: -inf
  Upper bound: inf
  Number of objectives: 1
  Number of constraints: 0
  Number of variables: 2
  Sense: unknown
# ----------------------------------------------------------
#   Solver Information
# ----------------------------------------------------------
Solver:
- Status: ok
  Message: couenne\x3a Optimal
  Termination condition: optimal
  Id: 3
  Error rc: 0
  Time: 0.023235797882080078
# ----------------------------------------------------------
#   Solution Information
# ----------------------------------------------------------
Solution:
- number of solutions: 0
  number of solutions displayed: 0

Profit =  2600.00002599998

Decision Variables
x =  20.000000199999512
y =  60.00000059999999

Constraints
Demand  =  20.000000199999512
Labor A =  80.0000007999995
Labor B =  100.00000099999902
```

Gecode Installation

In [ ]:

```
!wget -N -q "https://ampl.com/dl/open/gecode/gecode-linux64.zip"
!unzip -o -q gecode-linux64
```

Gecode solves constraint programming problems and does not support continuous variables. We therefore create a second model using exclusively discrete variables.

In [15]:

```
from pyomo.environ import *

# create a model
discrete_model = ConcreteModel()

# declare decision variables
discrete_model.x = Var(domain=NonNegativeIntegers)
discrete_model.y = Var(domain=NonNegativeIntegers)

# declare objective
discrete_model.profit = Objective(expr = 40*discrete_model.x + 30*discrete_model.y, sen
se=maximize)

# declare constraints
discrete_model.demand = Constraint(expr = discrete_model.x <= 40)
discrete_model.laborA = Constraint(expr = discrete_model.x + discrete_model.y <= 80)
discrete_model.laborB = Constraint(expr = 2*discrete_model.x + discrete_model.y <= 100)

discrete_model.pprint()
```

```
2 Var Declarations
    x : Size=1, Index=None
        Key  : Lower : Value : Upper : Fixed : Stale : Domain
        None :     0 :  None :  None : False :  True : NonNegativeIntegers
    y : Size=1, Index=None
        Key  : Lower : Value : Upper : Fixed : Stale : Domain
        None :     0 :  None :  None : False :  True : NonNegativeIntegers

1 Objective Declarations
    profit : Size=1, Index=None, Active=True
        Key  : Active : Sense    : Expression
        None :   True : maximize : 40*x + 30*y

3 Constraint Declarations
    demand : Size=1, Index=None, Active=True
        Key  : Lower : Body : Upper : Active
        None :  -Inf :    x :  40.0 :   True
    laborA : Size=1, Index=None, Active=True
        Key  : Lower : Body  : Upper : Active
        None :  -Inf : x + y :  80.0 :   True
    laborB : Size=1, Index=None, Active=True
        Key  : Lower : Body    : Upper : Active
        None :  -Inf : 2*x + y : 100.0 :   True

6 Declarations: x y profit demand laborA laborB
```

In [16]:

```
SolverFactory('gecode', executable='/content/gecode').solve(discrete_model).write()

# display solution
print('\nProfit = ', discrete_model.profit())

print('\nDecision Variables')
print('x = ', discrete_model.x())
print('y = ', discrete_model.y())

print('\nConstraints')
print('Demand  = ', discrete_model.demand())
print('Labor A = ', discrete_model.laborA())
print('Labor B = ', discrete_model.laborB())
```

```
# ==========================================================
# = Solver Results                                         =
# ==========================================================
# ----------------------------------------------------------
#   Problem Information
# ----------------------------------------------------------
Problem:
- Lower bound: -inf
  Upper bound: inf
  Number of objectives: 1
  Number of constraints: 0
  Number of variables: 2
  Sense: unknown
# ----------------------------------------------------------
#   Solver Information
# ----------------------------------------------------------
Solver:
- Status: ok
  Message: gecode 4.4.0\x3a optimal solution; 201 nodes, 0 fails, objective 2600
  Termination condition: optimal
  Id: 0
  Error rc: 0
  Time: 0.019869565963745117
# ----------------------------------------------------------
#   Solution Information
# ----------------------------------------------------------
Solution:
- number of solutions: 0
  number of solutions displayed: 0

Profit =  2600.0

Decision Variables
x =  20.0
y =  60.0

Constraints
Demand  =  20.0
Labor A =  80.0
Labor B =  100.0
```

In [ ]: