EXAMPLES OF LINEAR OPTIMIZATION

José M. Garrido
Department of Computer Science

January 2016

College of Computing and Software Engineering
Kennesaw State University

1 Linear Optimization Models with Python

Python is a very good language used to model linear optimization problems. Two important Python features facilitate this modeling:

- The syntax of Python is very clean and it lends itself to naturally adapt to expressing (linear) mathematical programming models
- Python has the built-in data structures necessary to build and manipulate models built in.

Python uses a linear optimization solver, such as GLPK, to compute the actual optimization. Therefore, with Python there will always be an underlying efficient linear optimization solver.

Several Python libraries or packages are available for modeling linear optimization problems, some of the most known are:

- Pyomo - Coopr
- Pulp
- PyGLPK
- PyLPSolve
- PyMathProg
- PyCplex

2 Modeling with Pyomo

The Python Optimization Modeling Objects also known as Pyomo is a software package that supports the formulation and analysis of mathematical models for complex optimization applications. A linear optimization model in Pyomo is comprised of modeling components that define different aspects of the model. Pyomo uses index sets, symbolic parameters that are used to specify decision variables, objective functions, and constraints. Two types of models can be specified with Pyomo:

1. A *concrete* model, in which the problem data is embedded in the mathematical model itself.

2. An *abstract* model, in which the problem data is separated from the symbolic (mathematical) model.

A concrete model is generally more convenient for simple and relatively small problems. An abstract model is more appropriate for larger problems, which often have larger data sets.

2.1 Formulating Case Study 1

The following listing includes the model of Case Study 1 (previous chapter) using Pyomo. Note that this is a concrete model and is stored in file `casestud1.py`. The problem data is specified in lines 13–34 using Python lists and dictionaries. The model decision variables are declared in line 60 with Pyomo class *Var*. The variables are indexed by list *Products* and the values are limited to be non negative reals.

The objective function is specified in lines 63–65 with Pyomo class *Objective*. The decision variables and the dictionary *MPrice* are indexed by list *Products*.

This model has two types of constraints, the first type are the material constraints, which are generated for every material type in the model. The second type of constraint are the production constraints, for which only one constraint is generated.

Constraints are generated with Pyomo class *Constraint* and indicated the appropriate expression. Lines 68–73 defines a constraint function with parameter p that is used in line 76 to specify one constraint for each material type. This statement generates the constraints indexed by list *Materials*. The production constraint is specified in line 80.

```
 1 """
 2 Case Study 1 Python Formulation for the Pyomo Modeler
 3 An industrial chemical plant produces substances A and B
 4 The company needs to optimize the amount of A and B to
 5 maximize sales. Concrete model.
 6 J M Garrido, September  2014. Usage: pyomo casestud1.py
 7 """
 8 print
 9 print "Case Study 1: Chemical Plant Production"
10 # Import
11 from coopr.pyomo import *
12
13 # Data for Linear Optimization Problem
14 N = 2   # number of products
15 Products = range(1, N+1) # list of indices for decision var
```

```
16 IndxProd = 1       # index of product with limit
17 ProdLimit = 18.5   # limit of product 1 (pounds)
18 numprod = range(N)
19
20 Price = [12.75, 15.25]  # price per pound for each product
21 MPrice = {Products[i] : Price[i] for i in numprod}
22
23 M = 2  # M: number of types of material
24 Material = range(1, M+1) # list of indices for materials
25 nummat = range(M)
26
27 #Capacity of available material (pounds)
28 CapMat = [21.85, 29.5]
29 AvailMat = {Material[i] : CapMat[i] for i in nummat}
30
31 # requirement of materials for every pound of product
32 MatReq = [[0.25, 0.125],
33           [0.15, 0.350]]
34 RequireMat = {(Products[i], Material[j]) : MatReq[i][j] for j in
       nummat for i in numprod}
35
36 #Print Data
37 print
38 print "Price (per pound) of product: "
39 for i in numprod:
40    print "Product",Products[i], ":", MPrice[Products[i]]
41 print "Product 1 limit: ", ProdLimit
42 print
43 print "Available Material:"
44 for j in nummat:
45    print "Material",Material[j], ":", AvailMat[Material[j]]
46 print
47 print "Requirements of Material "
48 for i in numprod:
49    for j in nummat:
50        print "Product",Products[i], "-", "Material",Material[j],
              ":", MatReq[i][j]
51 print
52
53 #Concrete Model
54 model = ConcreteModel()
55
56 #Decision Variables
57 # The 2 variables x1, x2 are created with a lower limit of zero
58 # x1 is the amount of product 1 to produce
```

```
59 # x2 is the amount of product 2 to produce
60 model.Prod = Var(Products, within=NonNegativeReals)
61
62 # The objective function
63 model.obj = Objective(expr=
64             sum(MPrice[i] * model.Prod[i] for i in Products),
65             sense = maximize)
66
67 # Capacity Constraints
68 def CapacityRule(model, p):
69     """
70     This function has the Pyomo model as the first positional
           parameter,
71     and a material requirement index as a second positional
           parameter
72     """
73     return sum(RequireMat[i,p] * model.Prod[i] for i in Products)
          <= AvailMat[p]
74
75 # Generate one constraint for each material type
76 model.Capacity = Constraint(Material, rule = CapacityRule)
77
78 # Production Constraint
79 # Limit of production for Products[0]
80 model.ProdRestriction = Constraint(expr=model.Prod[IndxProd]
       <= ProdLimit)
```

The command line that runs this linear optimization model with Pyomo and the results are shown in the following listing. The first part displays the values of the input data for verification purposes and some messages about solving the optimization problem. In the second part of the output, the Solution Summary is displayed. The value computed of the decision variables (*Prod*) at the optimal point are 18.6 and 77.67867. The optimal value of the objective function is 1420.47321429.

```
 $ pyomo casestud1.py --summary
[    0.00] Setting up Pyomo environment
[    0.00] Applying Pyomo preprocessing actions

Case Study 1: Chemical Plant Production

Price (per pound) of product:
Product 1 : 12.75
Product 2 : 15.25
```

```
Product 1 limit:  18.5

Available Material:
Material 1 : 21.85
Material 2 : 29.5

Requirements of Material
Product 1 - Material 1 : 0.25
Product 1 - Material 2 : 0.125
Product 2 - Material 1 : 0.15
Product 2 - Material 2 : 0.35

[    0.01] Creating model
[    0.01] Applying solver
[    0.05] Processing results
    Number of solutions: 1
    Solution Information
      Gap: 0.0
      Status: feasible
      Function Value: 1420.47321429
    Solver results file: results.json

============================================================
Solution Summary
============================================================

Model Chemical Plant Production

  Variables:
    Prod : Size=2, Index=Prod_index, Domain=NonNegativeReals
        Key : Lower : Value         : Upper : Initial : Fixed : Stale
          1 :     0 :          18.5 :  None :    None : False : False
          2 :     0 : 77.6785714286 :  None :    None : False : False

  Objectives:
    obj : Size=1, Index=None, Active=True
        Key  : Active : Value
        None :   True : 1420.47321429

  Constraints:
    Capacity : Size=2
        Key : Lower : Body          : Upper
          1 :  None : 16.2767857143 : 21.85
          2 :  None :          29.5 :  29.5
    ProdRestriction : Size=1
```

```
        Key  : Lower : Body : Upper
        None :  None : 18.5 :  18.5

[    0.06] Applying Pyomo postprocessing actions
[    0.06] Pyomo Finished
```

2.2 An Abstract Model Case Study 1

As mentioned previously, in an abstract model the data is separated from the mathematical model. The abstract data structures in the model are declared as parameters and sets and the actual data values are given in a data file. So essentially two files are used when invoking Pyomo, the file with the symbolic model and the file with the data values.

The following list shows the abstract model in file `casestud1abs.py`. In this model, the first important line is the declaration of the abstract model and this appears in line 13.

In line 16, a parameter is declared with name m that correspond to the number of types of products and it is constrained to be a non negative integer. Parameter n is similarly in line 21 and correspond to the number of types of materials.

Parameter m is used in line 19 to declare a set with name *Products* that will be used to index parameters *MPrice* (line 31), *RequireMat* (line 37), and to index the decision variables *Prod* in line 44 .

Parameter n is similarly used to declare a set with name *Material* in line 23 that will be used to index parameters *AvailMat* in line 34, *RequireMat* in line 37, and the material capacity constraints in line 60.

```
 1 """
 2 Case Study 1 Python Formulation for the Pyomo Modeler
 3 An industrial chemical plant produces substances A and B
 4 The company needs to optimize the amount of A and B to
 5 maximize sales. Abstract model.
 6 J M Garrido, September  2014
 7 usage: pyomo casestud1abs.py casestud1.dat --summary
 8 """
 9 # Linear Optimization Problem
10 print
11 print "Case Study 1: Chemical Plant Production"
12 from coopr.pyomo import *
13
14 model = AbstractModel()
15
```

```
16 model.m = Param(within=NonNegativeIntegers) # Number products
17
18 # Set of indices of Types of items produced
19 model.Products = RangeSet(1, model.m)
20
21 model.n = Param(within=NonNegativeIntegers) # Number materials
22 # Set of indices of Types of Substances required in production
23 model.Material = RangeSet(1, model.n)
24
25 # Limit of product 1
26 model.ProdLimit = Param(within=PositiveReals)
27 # Index for type of individual material with limit
28 model.K = Param(within=NonNegativeIntegers)
29 model.IndxProd = RangeSet(model.K, model.K)
30
31 model.MPrice = Param(model.Products, within=PositiveReals)
32
33 #Available material
34 model.AvailMat = Param(model.Material, within=PositiveReals)
35
36 # requirement of materials for every pound of product
37 model.RequireMat = Param(model.Products, model.Material)
38
39 #Decision Variables
40 # The 2 variables x1, x2 are created with a lower limit of zero
41 # x1 is the amount of product 1 to produce
42 # x2 is the amount of product 2 to produce
43
44 model.Prod = Var(model.Products, domain=NonNegativeReals)
45
46 # The objective function
47 def objective_expr(model):
48     return summation(model.MPrice, model.Prod)
49 model.obj = Objective(rule=objective_expr, sense = maximize)
50
51 # Material Capacity Constraints
52 def CapacityF(model,p):
53     """
54     This function for material constraint, the pyomo model as
55  the first positional parameter, and a material requirement index
56  as the second positional parameter.
57     """
58     return sum(
     model.RequireMat[i,p] * model.Prod[i] for i in model.Products)
         <= model.AvailMat[p]
```

```
59 # Generate one constraint for each material
60 model.Capacity = Constraint(model.Material, rule=CapacityF)
61
62 # Production Constraint - Limit of production for Product 1
63 def ProductLimit(model, i):
64     return (model.Prod[i] <= model.ProdLimit)
65 model.ProdRestriction = Constraint(model.IndxProd,
        rule=ProductLimit)
```

The data for the linear optimization model of Case Study 1 is stored in file `casestud1.dat` and is shown in the following listing.

```
# Data file: casestud1.dat. Case Study 1: Chemical Plant Production

param m := 2;   # Number of types of products
param n := 2;   # Number of types of materials

# Limit of production of (first) product
param ProdLimit  := 18.5 ;
# Index of (first) product with production limit
param K := 1 ;

# Market Price of every pound of product
param MPrice := 1 12.75
                2 15.25
   ;

# Available material, amount of every material
param AvailMat := 1 21.85
                  2 29.5
  ;

# Amount of material required for producing each pound of product
# One product per row (product, material, material_amount)
param RequireMat := 1 1 0.25 1 2 0.125
                    2 1 0.15 2 2 0.350
     ;
```

The other three case studies are formulated as concrete models with Pyomo and are stored in files: `casestud2.py`, `casestud3.py`, and `casestud4.py`. In addition to these case studies, several sample models are included in the `pyomo` directory.

3 Modeling with Pulp

Pulp is another good modeler for linear optimization models and is written in Python. The general setup of the problem model is similar to Pyomo. It is convenient to use Python lists and dictionaries for the problem data.

The following listing contains the model specification for Case Study 1 and is stored in file `casestud1b.py`. A simplified model of this problem is stored in file `casestud1.py` and several additional models are stored in the directory `pulp_models`.

```
 1 """
 2 Case Study 1 Python Formulation for the PuLP Modeller
 3 An industrial chemical plant produces substances A and B
 4 J M Garrido, September  2014
 5 Usage: python casestud1b.py
 6 """
 7
 8 # Import PuLP modeler functions
 9 from pulp import *
10
11 N = 2         # number of product types
12 Products = range(1, N+1)   # list of products
13 IndxProd = 1      # product with production limit
14 ProdLimit = 18.5
15 numprod = range(N) # index list of product types
16
17 Price = [12.75, 15.25] # price per pound for each product
18 MPrice = {Products[i] : Price[i] for i in numprod}
19
20 M = 2    # number of material types
21 Materials = range(1, M+1) # list material types
22 nummat = range(M)         # index list of materials
23
24 #Capacity of available material (pounds)
25 CapMat = [21.85, 29.5]
26 AvailMat = {Materials[i] : CapMat[i] for i in nummat}
27
28 # requirement of materials for every pound of product
29 MatReq = [[0.25, 0.125],
30           [0.15, 0.350]]
31 RequireMat = {(Products[i], Materials[j]) : MatReq[i][j] for j
       in nummat for i in numprod}
32
33 #Print Data
```

```
34 print
35 print "Price (per pound) of product: "
36 for i in numprod:
37    print "Product",Products[i], ":", MPrice[Products[i]]
38 print "Product 1 limit: ", ProdLimit
39 print
40 print "Available Material:"
41 for j in nummat:
42    print "Material",Materials[j], ":", AvailMat[Materials[j]]
43 print
44 print "Requirements of Material "
45 for i in numprod:
46    for j in nummat:
47        print "Product",Products[i], "-", "Material",Materials[j],
              ":", MatReq[i][j]
48 print
49
50 # Create the model to contain the problem data
51 model = LpProblem("Case study 1", LpMaximize)
52
53 # Decision variables
54 Prod = LpVariable.dicts("ProdVar", Products, 0, None)
55
56 # The objective function
57 model += lpSum([MPrice[i]*Prod[i] for i in Products]),"Total Sales"
58
59 # The constraints of available material
60 for p in Materials:
61     model += lpSum([RequireMat[i,p] * Prod[i] for i in Products])
           <= AvailMat[p], "Maximum of Material %d"%p
62
63 # Production Constraint
64 # Limit of production for Products[IndxProd]
65 model += Prod[IndxProd] <= ProdLimit, "Production limit"
66
67 # Write the problem data to an .lp file
68 model.writeLP("casestud1b.lp")
69
70 # Solve the optimization problem using the specified PuLP Solver
71 model.solve(GLPK())
72
73 # Print the status of the solution
74 print "Status:", LpStatus[model.status]
75
76 # Print each of the variables with it's resolved optimum value
```

```
77 for v in model.variables():
78     print v.name, "=", v.varValue
79
80 # Print the optimised value of the objective function
81 print "Optimal sales", value(model.objective)
```

The data is set with Python lists and dictionaries on lines 11–31 and is exactly the same as in the model using Pyomo. Lines 33–48 displays the problem data for verification purposes.

The Pulp model for the problem is declared in line 51 using Pulp class *LPProblem* with argument *LPMaximize*. The decision variables are created in line 54 as a dictionary indexed by list *Products*. The lower bound for the value of the variables is zero and there is no upper bound (specified as *None*). By default, the variables created are of type real and type integer is specified with the argument *LpInteger*.

The objective function is specified in line 57 the summation is indexed by list *Products*. The materials constraints specified in lines 60–61 and are indexed by list *Materials*. The single production constraint is specified with variable *Prod* indexed by the scalar parameter *IndxProd*.

The model is solved with GLPK specified solver in line 71. The status of the solution is displayed in line 74. The computed optimal values of the decision variables are displayed in lines 77–78 and the optimal value of the objective function is displayed in line 81.

The following listing shows the output produced after running the model. The command line used is the first line shown in the listing.

```
$python casestud1b.py
/usr/local/lib/python2.7/dist-packages/pulp_or-1.4.6-py2.7.
  egg/pulp

Price (per pound) of product:
Product 1 : 12.75
Product 2 : 15.25
Product 1 limit:  18.5

Available Material:
Material 1 : 21.85
Material 2 : 29.5

Requirements of Material
Product 1 - Material 1 : 0.25
Product 1 - Material 2 : 0.125
Product 2 - Material 1 : 0.15
```

```
Product 2 - Material 2 : 0.35

GLPSOL: GLPK LP/MIP Solver, v4.54
Parameter(s) specified in the command line:
 --cpxlp /tmp/3556-pulp.lp -o /tmp/3556-pulp.sol
Reading problem data from '/tmp/3556-pulp.lp'...
3 rows, 2 columns, 5 non-zeros
11 lines were read
GLPK Simplex Optimizer, v4.54
3 rows, 2 columns, 5 non-zeros
Preprocessing...
2 rows, 2 columns, 4 non-zeros
Scaling...
 A: min|aij| =  1.250e-01  max|aij| =  3.500e-01  ratio =
   2.800e+00
Problem data seem to be well scaled
Constructing initial basis...
Size of triangular part is 2
*     0: obj =   0.000000000e+00  infeas =  0.000e+00 (0)
*     2: obj =   1.420473214e+03  infeas =  0.000e+00 (0)
OPTIMAL LP SOLUTION FOUND
Time used:   0.0 secs
Memory used: 0.0 Mb (37657 bytes)
Writing basic solution to '/tmp/3556-pulp.sol'...
Status: Optimal
ProdVar_1 = 18.5
ProdVar_2 = 77.6786
Optimal sales 1420.47365
```

4 Software Linear Optimization Solvers

There are many software linear optimization solvers, some free open source and others commercial solvers. This section discusses two software tools that are cross-platform (Linux, MacOS, and MS Windows), solve linear optimization problems using the Simplex algorithm and variations of it. These are LP_solve and GLPK, and problem formulation and executions are shown with several examples. Several Python linear optimization modelers use GLPK as the default solver.

5 Short List of Optimization Solvers

Commercial linear optimization solvers are available some only for MS Windows and others for multiple platforms. These are generally faster than the free open-source

solvers and some of the most widely used are:

- Gurobi
- Cplex
- Xpress
- Lindo

The most widely used of the open-source and cross-platform software linear optimization solvers are:

- The Gnu Linear Programming Kit also known as GLPK
- LPsolve
- SCIP
- CLP (Coin-or)