Pyomo Workshop

Exercises 1: Pyomo Fundamentals

1.1 Knapsack example: Solve the knapsack problem shown in the tutorial using your IDE (e.g., Spyder) or the command line: `> python knapsack.py`. Which items are acquired in the optimal solution? What is the value of the selected items?

1.2 Knapsack with improved printing: The `knapsack.py` example shown in the tutorial uses `model.pprint()` to see the value of the solution variables. Starting with the code in `knapsack_print_incomplete.py`, complete the missing lines to produce formatted output. Note that the Pyomo `value` function should be used to get the floating point value of Pyomo modeling components (e.g., `print(value(model.x[i]))`. Also print the value of the items selected (the objective), and the total weight. (A solution can be found in `knapsack_print_soln.py`).

1.3 Changing data: If we were to increase the value of the wrench, at what point would it become selected as part of the optimal solution? (A solution can be found in `knapsack_wrench_soln.py`.)

1.4 Loading data from Excel: In the knapsack example shown in the tutorial slides, the data is hardcoded at the top of the file. Instead of hardcoding the data, use Python to load the data from a different source. You can start from the file `knapsack_pandas_excel_incomplete.py`. (A solution that uses pandas to load the data from Excel is shown in `knapsack_pandas_excel_soln.py`.)

1.5 NLP vs MIP: Solve the knapsack problem with IPOPT instead of `glpk`. (Hint: switch `glpk` to `ipopt` in the call `SolverFactory`. Print the solution values for `model.x`. What happened? Why?

Exercises 2: Pyomo Fundamentals

2.1 Knapsack problem with rules: Rules are important for defining indexed constraints, however, they can also be used for single (i.e. scalar) constraints. Starting with `knapsack.py`, reimplement the model using rules for the objective and the constraints. (A solution can be found in `knapsack_rules_soln.py`.)

2.2 Integer formulation of the knapsack problem: Consider again, the knapsack problem. Assume now that we can acquire multiple items of the same type. In this new formulation, x_i is now an integer variable instead of a binary variable. One way to formulate this problem is as follows:

$$
\begin{aligned}
&\max_{q,x} \sum_{i \in A} v_i x_i \\
&\text{s.t.} \sum_{i \in A} w_i x_i \leq W_{\max} \\
&\quad x_i = \sum_{j=0}^{N} j q_{i,j} && \forall\, i \in A \\
&\quad 0 \leq x \leq N \\
&\quad q_{i,j} \in \{0, 1\} && \forall i \in A, j \in \{0..N\}
\end{aligned}
$$

Starting with `knapsack_rules.py`, implement this new formulation and solve. Is the solution surprising? (A solution can be found in `knapsack_integer_soln.py`.)

Exercises 3: Pyomo Fundamentals

3.1 Using the decorator notation for rules: In the slides, we saw an alternative notation for declaring and defining Pyomo components using decorators. Starting with the warehouse location problem in `warehouse_location_decorator_incomplete.py` change the model to use the decorator notation. (A solution for this problem can be found in `warehouse_location_decorator_soln.py`.)

3.2 Changing Parameter values: In the tutorial slides, we saw that a parameter could be specified to be `mutable`. This tells Pyomo that the value of the parameter may change in the future, and allows the user to change the parameter value and resolve the problem without the need to rebuild the entire model each time. We will use this functionality to find a better solution to an earlier exercise. Considering again the knapsack problem, we would like to find when the wrench becomes valuable enough to be a part of the optimal solution. Create a Pyomo `Parameter` for the value of the items, make it mutable, and then write a loop that prints the solution for different wrench values. Start with the file `knapsack_mutable_parameter_incomplete.py`. (A solution for this problem can be found in `knapsack_mutable_parameter_soln.py`.)

3.3 Integer cuts: Often, it can be important to find not only the "best" solution, but a number of solutions that are equally optimal, or close to optimal. For discrete optimization problems, this can be done using something known as an integer cut. Consider again the knapsack problem where the choice of which items to select is a discrete variable $x_i \ \forall \ i \in A$. Let $x_i^\star$ be a particular set of x values we want to remove from the feasible solution space. We define an integer cut using two sets. The first set S_0 contains the indices for those variables whose current solution is 0, and the second set S_1 consists of indices for those variables whose current solution is 1. Given these two sets, an integer cut constraint that would prevent such a solution from appearing again is defined by,

$$\sum_{i \in S_0} x[i] + \sum_{i \in \in S_1} (1 - x[i]) \geq 1.$$

Starting with `knapsack_rules.py`, write a loop that solves the problem 5 times, adding an integer cut to remove the previous solution, and

printing the value of the objective function and the solution at each iteration of the loop. (A solution for this problem can be found in `knapsack_integer_cut_soln.py`

3.4 Putting it all together with the lot sizing example: (Hart et al., 2017)

We will now write a complete model from scratch using a well-known multi-period optimization problem for optimal lot-sizing adapted from Hagen et al. (2001) shown below.

$$\min \sum_{t \in T} c_t y_t + h_t^+ I_t^+ + h_t^- I_t^- \tag{1}$$

$$\text{s.t.} \quad I_t = I_{t-1} + X_t - d_t \qquad \forall t \in T \tag{2}$$

$$I_t = I_t^+ - I_t^- \qquad \forall t \in T \tag{3}$$

$$X_t \leq P y_t \qquad \forall t \in T \tag{4}$$

$$X_t,\ I_t^+,\ I_t^- \geq 0 \qquad \forall t \in T \tag{5}$$

$$y_t \in \{0, 1\} \qquad \forall t \in T \tag{6}$$

Our goal is to find the optimal production X_t given known demands d_t, fixed cost c_t associated with active production in a particular time period, an inventory holding cost h_t^+ and a shortage cost h_t^- (cost of keeping a backlog) of orders. The variable y_t (binary) determines if we produce in time t or not, and I_t^+ represents inventory that we are storing across time period t, while I_t^- represents the magnitude of the backlog. Note that equation (4) is a constraint that only allows production in time period t if the indicator variable y_t=1.

Write a Pyomo model for this problem and solve it using `glpk` using the data provided below. You can start with the file `lot_sizing_incomplete.py`. (A solution is provided in `lot_sizing_soln.py`.)

Parameter	Description	Value
c	fixed cost of production	4.6
I_0^+	initial value of positive inventory	5.0
I_0^-	initial value of backlogged orders	0.0
h^+	cost (per unit) of holding inventory	0.7
h^-	shortage cost (per unit)	1.2
P	maximum production amount (big-M value)	5
d	demand	$[5, 7, 6.2, 3.1, 1.7]$

Exercises: Nonlinear

1.1 Alternative Initialization: Effective initialization can be critical for solving nonlinear problems, since they can have several local solutions and numerical difficulties. Solve the Rosenbrock example using different initial values for the x variables. Write a loop that varies the initial value from 2.0 to 6.0, solves the problem, and prints the solution for each iteration of the loop. (A solution for this problem can be found in `rosenbrock_init_soln.py`.)

1.2 Evaluation errors: Consider the following problem with initial values x=5, y=5.

$$\min_{x,y} \; f(x,y) = (x-1.01)^2 + y^2$$
$$\text{s.t.} \quad y = \sqrt{x - 1.0}$$

(a) Starting with `evaluation_error_incomplete.py`, formulate this Pyomo model and solve using IPOPT. You should get a list of errors from the solver. Add the IPOPT solver option `solver.options['halt_on_ampl_error']='yes'` to find the problem. (Hint: error output might be ordered strangely, look up in the console output.) What did you discover? How might you fix this? (A solution for this can be found in `evaluation_error_soln.py`.)

(b) Add bounds $x \geq 1$ to fix this problem. Resolve the problem. Comment on the number of iterations and the quality of solution. (Note: The problem still occurs because $x \geq 1$ is not enforced exactly, and small numerical values still cause the error.) (A solution for this can be found in `evaluation_error_bounds_soln.py`.)

(c) Think about other solutions for this problem. (e.g., $x \geq 1.001$). (A solution for this can be found in `evaluation_error_bounds2_soln.py`.)

1.3 Alternative Formulations: Consider the following problem with initial values x=5, y=5.

$$\min_{x,y} \; f(x,y) = (x-1.01)^2 + y^2$$
$$\text{s.t.} \quad \frac{x-1}{y} = 1$$

Note that the solution to this problem is x=1.005 and y=0.005. There are several ways that the problem above can be reformulated. Some examples are shown below. Which ones do you expect to be better? Why? Starting with `formulation_incomplete.py` finish the Pyomo model for each of the formulations and solve with IPOPT. Note the number of iterations and quality of solutions. What can you learn about problem formulation from these examples?

(a) (A solution can be found in `formulation_1_soln.py`.)

$$\min_{x,y} \; f(x,y) = (x-1.01)^2 + y^2$$
$$\text{s.t.} \quad \frac{x-1}{y} = 1$$

(b) (A solution for this can be found in `formulation_2_soln.py`.)

$$\min_{x,y} \; f(x,y) = (x-1.01)^2 + y^2$$
$$\text{s.t.} \quad \frac{x}{y+1} = 1$$

(c) (A solution for this can be found in `formulation_3_soln.py`.)

$$\min_{x,y} \; f(x,y) = (x-1.01)^2 + y^2$$
$$\text{s.t.} \quad y = x - 1$$

(d) Bounds and initialization can be very helpful when solving nonlinear optimization problems. Starting with `formulation_incomplete.py` resolve the original problem, but add bounds, $y \geq 0$. Note the number of iterations and quality of solution, and compare with what you found in 1.2 (a). (A solution for this can be found in `formulation_4_soln.py`.)

1.4 Reactor design problem (Hart et al., 2017; Bequette, 2003): In this example, we will consider a chemical reactor designed to produce product B from reactant A using a reaction scheme known as the Van de Vusse reaction:

$$A \xrightarrow{k_1} B \xrightarrow{k_2} C$$
$$2A \xrightarrow{k_3} D$$

Under appropriate assumptions, F is the volumetric flowrate through the tank. The concentation of component A in the feed is c_{Af}, and the concentrations in the reactor are equivalent to the concentrations of each component flowing out of the reactor, given by c_A, c_B, c_C, and c_D.

If the reactor is too small, we will not produce sufficient quantity of B, and if the reactor is too large, much of B will be further reacted to form the undesired product C. Therefore, our goal is to solve for the reactor volume that maximizes the outlet concentration for product B.

The steady-state mole balances for each of the four components are given by,

$$\begin{aligned}
0 &= \frac{F}{V}c_{Af} - \frac{F}{V}c_A - k_1 c_A - 2k_3 c_A^2 \\
0 &= -\frac{F}{V}c_B + k_1 c_A - k_2 c_B \\
0 &= -\frac{F}{V}c_C + k_2 c_B \\
0 &= -\frac{F}{V}c_D + k_3 c_A^2
\end{aligned}$$

The known parameters for the system are,

$$c_{Af} = 10\ \frac{\text{gmol}}{\text{m}^3} \quad k_1{=}\frac{5}{6}\ \text{min}^{-1} \quad k_2{=}\frac{5}{3}\ \text{min}^{-1} \quad k_3{=}\frac{1}{6000}\ \frac{\text{m}^3}{\text{mol min}}.$$

Formulate and solve this optimization problem using Pyomo. Since the volumetric flowrate F always appears as the numerator over the reactor volume V, it is common to consider this ratio as a single variable, called the space-velocity SV. (A solution to this problem can be found in `reactor_design_soln.py`.)

References

Hart, W. E., Laird, C. D., Watson, J. P., Woodruff, D. L., Hackebeil, G. A., Nicholson, B. L., and Siirola, J. D. Pyomo: Optimization Modeling in Python (Second Edition), Vol (67), Springer Verlag, 2017.

Kjetil K. Haugen, Arne Lkketangen, and David L. Woodruff. Progressive hedging as a meta-heuristic applied to stochastic lot-sizing. European Journal of Operational Research, 132(1):116 122, 2001

B.W. Bequette. Process control: modeling, design, and simulation. Prentice Hall, 2003.