Solving MILP Problem using Pyomo

Dr. Pranay Kumar Saha
Department of Computer Science and Engineering,
Indian Institute of Technology (Indian School of Mines) Dhanbad

November 29, 2024

Python Setup Tutorial for Windows and macOS

1. Setup Python on Windows

Step 1: Download Python

1. Visit the official Python website: `https://www.python.org/downloads/`.
2. Download the latest stable release for Windows.

Step 2: Install Python

1. Run the downloaded `.exe` file.
2. **Important:** Check the box for **"Add Python to PATH"**.
3. Select **"Customize Installation"** and ensure the following are selected:
   - `pip` (for package management).
   - `tcl/tk` and `IDLE` (optional, for GUI support).
   - `py launcher`.
4. Complete the installation by clicking **Install**.

Step 3: Verify Installation

Open the Command Prompt and run:

```
python --version
pip --version
```

Both commands should return version numbers.

Step 4: Set Up a Virtual Environment

1. Create a project directory:

```
mkdir my_project
cd my_project
```

2. Create a virtual environment:

```
python -m venv venv
```

3. Activate the virtual environment:

```
venv\Scripts\activate
```

4. Verify the Python version:

```
python --version
```

Step 5: Install Python Packages

1. Install packages using `pip`:

```
pip install <package_name>
```

2. Example: Install Pyomo:

```
pip install pyomo
```

Step 6: Deactivate the Virtual Environment

To deactivate, run:

```
deactivate
```

2. Setup Python on macOS

Step 1: Check Preinstalled Python

macOS comes with Python preinstalled. To check the version:

```
python3 --version
```

Step 2: Install Python

1. Download the latest stable release from `https://www.python.org/downloads/`.
2. Alternatively, use Homebrew:

```
# Install Homebrew (if not already installed)
/bin/bash -c "$(curl -fsSL https://raw.githubusercontent.com/Homebrew/
    install/HEAD/install.sh)"
# Install Python
brew install python
```

Step 3: Verify Installation

Run the following commands in Terminal:

```
python3 --version
pip3 --version
```

Step 4: Set Up a Virtual Environment

1. Create a project directory:

```
mkdir my_project
cd my_project
```

2. Create a virtual environment:

```
python3 -m venv venv
```

3. Activate the virtual environment:

```
source venv/bin/activate
```

4. Verify the Python version:

```
python --version
```

Step 5: Install Python Packages

1. Install packages using `pip`:

```
pip install <package_name>
```

2. Example: Install Pyomo:

```
pip install pyomo
```

Step 6: Deactivate the Virtual Environment

To deactivate, run:

```
deactivate
```

3. Python Setup Tutorial for Ubuntu

Step 1: Check Preinstalled Python

Ubuntu typically comes with Python preinstalled. Verify the version:

```
python3 --version
pip3 --version
```

If Python or `pip` is not installed or outdated, proceed to the next step.

Step 2: Install Python and pip

Update your package list and install Python:

```
sudo apt update
sudo apt install python3 python3-pip -y
```

Verify the installation:

```
python3 --version
pip3 --version
```

Step 3: Install the `venv` Module

Ensure the virtual environment module is installed:

```
sudo apt install python3-venv -y
```

Step 4: Set Up a Virtual Environment

1. Create a directory for your project:

   ```
   mkdir my_project
   cd my_project
   ```

2. Create a virtual environment:

   ```
   python3 -m venv venv
   ```

3. Activate the virtual environment:

   ```
   source venv/bin/activate
   ```

4. Verify the Python version:

```
python --version
```

Step 5: Install Python Packages

1. Install packages using `pip`:

```
pip install <package_name>
```

2. Example: Install Pyomo:

```
pip install pyomo
```

Step 6: Deactivate the Virtual Environment

When done, deactivate the virtual environment:

```
deactivate
```

Step 7: Remove the Virtual Environment (Optional)

If you need to delete the virtual environment:

```
rm -rf venv
```

Environment Setup for Pyomo and CBC Solver

To set up your environment for Pyomo and the CBC solver, follow these steps:

1. **Install Pyomo using `pip`:**

```
pip install pyomo
```

2. **Install the CBC solver:**

   - If you're using **Anaconda** or **Miniconda**, install via `conda`:

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

   - If you're using **pip**, you need to download CBC separately:
     (a) Download the CBC binaries from the COIN-OR website.
     (b) Add the path of the CBC executable to your system's PATH environment variable.

     OR

   - using pip

```
pip install cbcpy
```

Introduction

Pyomo is a Python-based open-source optimization modeling language that supports defining optimization problems and solving them with a variety of solvers. Below is a list of open-source solvers that are compatible with Pyomo.

List of Open-Source Solvers

1. **GLPK (GNU Linear Programming Kit)**
   - **Capabilities**: Linear Programming (LP), Mixed-Integer Linear Programming (MILP).
   - **Usage**: Ideal for small to medium-sized linear problems.
   - **Installation**: Can be installed via package managers like `apt`, `brew`, or from source.
2. **CBC (Coin-or Branch and Cut)**
   - **Capabilities**: LP, MILP.
   - **Usage**: Suitable for large-scale integer programming problems.
   - **Installation**: Available through package managers and conda-forge (`conda install -c conda-forge coincbc`).
3. **CLP (Coin-or Linear Programming)**
   - **Capabilities**: LP.
   - **Usage**: Efficient for solving large-scale linear programs.
   - **Installation**: Install via conda-forge or from the COIN-OR website.
4. **HiGHS**
   - **Capabilities**: LP, MILP.
   - **Usage**: High-performance solver for large-scale problems.
   - **Installation**: Install via pip (`pip install highs`) or conda.
5. **Ipopt (Interior Point OPTimizer)**
   - **Capabilities**: Nonlinear Programming (NLP).
   - **Usage**: Suitable for large-scale nonlinear optimization.
   - **Installation**: Available via conda-forge (`conda install -c conda-forge ipopt`).
6. **Bonmin**
   - **Capabilities**: Mixed-Integer Nonlinear Programming (MINLP).

- **Usage**: For problems involving both integer and continuous nonlinear variables.
- **Installation**: Requires building from source; dependencies include Ipopt.

7. **Couenne**
   - **Capabilities**: Global optimization for nonconvex MINLP.
   - **Usage**: Solves nonconvex problems to global optimality.
   - **Installation**: Typically built from source along with its dependencies.
8. **GLPSOL**
   - **Capabilities**: LP, MILP.
   - **Usage**: Command-line solver that comes with GLPK.
   - **Installation**: Included with GLPK installation.
9. **OSQP (Operator Splitting Quadratic Program)**
   - **Capabilities**: Quadratic Programming (QP).
   - **Usage**: Efficient for large-scale QP problems.
   - **Installation**: Install via pip (`pip install osqp`).
10. **PENBOPT**
    - **Capabilities**: NLP.
    - **Usage**: For nonlinear optimization with bound constraints.
    - **Installation**: Available from the developer's website.

Using Solvers with Pyomo

1. **Installation**: Ensure the solver is installed on your system and is accessible via the command line. For Python-based solvers or those available via conda/pip, installation is straightforward.
2. **Pyomo Solver Interface**: Pyomo interacts with solvers through `SolverFactory`. For example:

```
from pyomo.environ import SolverFactory
solver = SolverFactory('glpk')  # Replace 'glpk' with the solver you are using
```

3. **Solver Options**: You can pass specific options to the solver using the `options` dictionary:

```
results = solver.solve(model, options={'tol': 1e-8})
```

Additional Resources

- **Pyomo Documentation**: Pyomo Solvers.
- **COIN-OR Projects**: Many solvers like CBC, CLP, Ipopt, Bonmin, and Couenne are part of the COIN-OR Project.
- **Conda Forge**: An excellent source for installing many of these solvers using conda.

Summary

By leveraging these open-source solvers, you can effectively solve a wide range of optimization problems using Pyomo without the need for commercial software. Make sure to choose a solver that aligns with the specific requirements of your optimization model.

Problem Statement

A company produces two products, **Product 1** and **Product 2**. Each product requires resources:

- **Product 1** requires 2 units of resource A and 1 unit of resource B.
- **Product 2** requires 1 unit of resource A and 2 units of resource B.

The company has:

- **8 units of resource A**.
- **6 units of resource B**.

The profit per unit is:

- **$3** for Product 1.
- **$5** for Product 2.

The objective is to determine how many units of each product to produce to maximize profit.

Pyomo Implementation

The following Python code demonstrates how to model and solve this problem using Pyomo:

```python
from pyomo.environ import ConcreteModel, Var, Objective, Constraint, SolverFactory

# Create a Pyomo model
model = ConcreteModel()

# Decision variables: x1 and x2 (quantities of Product 1 and Product 2)
model.x1 = Var(domain=float, bounds=(0, None)) # Product 1 quantity
model.x2 = Var(domain=float, bounds=(0, None)) # Product 2 quantity

# Objective function: Maximize profit
model.profit = Objective(expr=3 * model.x1 + 5 * model.x2, sense=-1) # Maximize

# Constraints
model.constraint1 = Constraint(expr=2 * model.x1 + 1 * model.x2 <= 8) # Resource A
model.constraint2 = Constraint(expr=1 * model.x1 + 2 * model.x2 <= 6) # Resource B

# Solve the model using GLPK
solver = SolverFactory('cbc') # Ensure CBC solver is installed
result = solver.solve(model)

# Display results
print("Status:", result.solver.status)
print("Termination condition:", result.solver.termination_condition)
print(f"Optimal value of x1 (Product 1): {model.x1.value}")
print(f"Optimal value of x2 (Product 2): {model.x2.value}")
print(f"Maximum profit: {model.profit.expr()}")
```

Explanation

1. **Model Setup**:
   - `ConcreteModel`: Defines a Pyomo model instance.
   - `Var`: Decision variables (`x1` and `x2`) are the quantities of products to produce.
2. **Objective Function**:
   - `Objective`: Maximizes profit based on the profit per product.
3. **Constraints**:

- `Constraint`: Ensures that resource usage does not exceed availability.

4. **Solver**:
   - `SolverFactory`: Specifies the solver (e.g., GLPK).
5. **Result**:
   - Outputs the optimal solution and the maximum profit.

Expected Output

If you run the script, you might see an output like this (depending on the solver used):

```
Status: ok
Termination condition: optimal
Optimal value of x1 (Product 1): 2.0
Optimal value of x2 (Product 2): 2.0
Maximum profit: 16.0
```

This means producing 2 units of Product 1 and 2 units of Product 2 yields the maximum profit of $16.

Exercise Questions

Question 1: Adjust Resource Availability

Suppose the company increases the availability of resources to:

- **Resource A**: 10 units,
- **Resource B**: 8 units.

Modify the Pyomo model to reflect these changes and find the new optimal solution:

1. Values of x_1 (Product 1) and x_2 (Product 2).
2. Maximum profit.

How do the results change compared to the original scenario?

Question 2: Add a New Product

The company decides to introduce a new product, **Product 3**, which requires:

- **3 units of Resource A**,
- **2 units of Resource B**.

Question 3

The profit per unit of **Product 3** is **$4**. Update the Pyomo model to include this new product and determine:

1. The optimal quantities of all three products (x_1, x_2, x_3).
2. The maximum profit.

A manufacturing company produces three types of products: **Product A**, **Product B**, and **Product C**. Each product requires different amounts of three key resources: **Raw Material 1**, **Raw Material 2**, and **Labor Hours**. The availability and usage of these resources are described below:

- **Resource Availability**:
  - Raw Material 1: 120 units.
  - Raw Material 2: 100 units.
  - Labor Hours: 200 hours.
- **Resource Usage per Unit of Product**:

	Product A	**Product B**	**Product C**
Raw Material 1	2 units	3 units	4 units
Raw Material 2	1 unit	2 units	3 units
Labor Hours	5 hours	4 hours	6 hours

- **Profit per Unit**:
  - Product A: $20.
  - Product B: $25.
  - Product C: $30.

Define the decision variables:

$$x_1 = \text{Number of units of Product A to produce,}$$

$$x_2 = \text{Number of units of Product B to produce,}$$

$$x_3 = \text{Number of units of Product C to produce.}$$

Tasks

1. Write the objective function to maximize the total profit.
2. Write down the constraints for the available resources.
3. Formulate the complete linear programming problem (objective and constraints).
4. Solve the problem using Pyomo or another optimization tool to find:
   - The optimal quantities of x_1, x_2, and x_3.
   - The maximum profit.
5. If the availability of Raw Material 2 increases to 150 units, how does the solution change?

Question 4

Consider an optimization problem where you must allocate resources to four activities: x_1, x_2, x_3, and x_4. The objective is to maximize the total profit subject to certain resource constraints.

Objective Function

Maximize:

$$Z = 5x_1 + 8x_2 + 6x_3 + 7x_4$$

Constraints

The activities are constrained by the following equations:

$$2x_1 + 4x_2 + 3x_3 + 5x_4 \leq 50 \quad \text{(Resource 1 constraint)}$$

$$3x_1 + 2x_2 + 5x_3 + 4x_4 \leq 60 \quad \text{(Resource 2 constraint)}$$

$$4x_1 + 3x_2 + 2x_3 + 6x_4 \leq 70 \quad \text{(Resource 3 constraint)}$$

Additionally, the decision variables must satisfy:

$$x_1, x_2, x_3, x_4 \geq 0 \quad \text{(Non-negativity constraint)}.$$

Tasks

1. Formulate the optimization problem as a linear programming problem.
2. Solve for the optimal values of x_1, x_2, x_3, and x_4.
3. Calculate the maximum profit Z.