Modeling and solving mathematical optimization problems with Python

SciPy India 2015

Industrial Engineering and Operations Research
Indian Institute of Technology Bombay

Outline

What is Optimization?

Optimization is a problem of decision making in which we need to choose between various alternatives under certain conditions.

Mathematical Modeling

- Modeling is a fundamental process in many aspects of scientific research, engineering, and business.
- Modeling involves the formulation of a simplified representation of a system or real-world object.
- Allow structured representation of knowledge about the original system.
- Optimization models are mathematical models that include functions that represent goals or objectives for the system being modelled with given condition.

General form of a mathematical model

$$\begin{aligned} & \min \text{ or } \max f(x_1, ..., x_n) && \text{(Objective function)} \\ \text{subject to,} \quad & g(x_1, ..., x_n) \geq 0 && \text{(functional constraints)} \\ & x_1, ..., x_n \in S && \text{(set constraints)} \end{aligned}$$

$x_1, ..., x_n$ are called decision variables

In another words, the goal is to find $x_1, ..., x_n$ such that

- They satisfy the constraints.
- If no such value exist for $x_1, ..., x_n$, the problem is infeasible.
- They achieve min or max objective function value (may be unbounded)

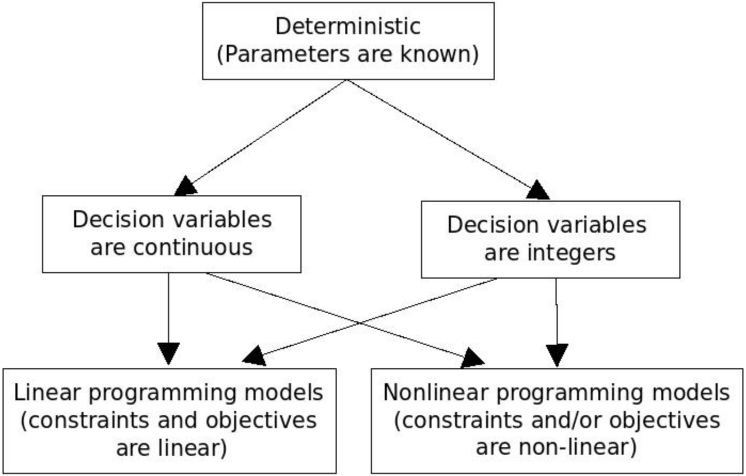

Figure 1 : Types of Deterministic Optimization Models

Applications of optimization

- Scheduling of Buses/trains
- Transportation network design
- Supply chain optimization
- Optimum circuit design of PCB
- Design optimization of various mechanical components
- Process optimization in chemical industry
- Designing of the sharing network in internet
- Search engine optimization
- Shop floor layout planning
- Production planning and scheduling
- Hospital management systems etc.

Popular optimization solvers

- CPLEX
- Gurobi
- GLPK
- CLP, CBC, IPOPT (part of COIN-OR)
- LINDO and Lingo etc.

Python interface for optimization

- Pyomo $\rightarrow$ used for LP models.
- PuLP $\rightarrow$ used for LP models.

- A Python-based modeling tool for optimization models.
- Goal is to provide a platform for expressing optimization models that supports the central ideas of modern AMLs within a framework
- Promotes flexibility, extensibility, portability, and maintainability.
- Pyomo modeling objects are embedded within Python gives rich set of supporting libraries.
- Pyomo can call solvers such as GLPK, Coin-OR, CPLEX and Gurobi to solve linear, integer and mixed integer models

Pyomo

Installing Pyomo

- First install Python pip by typing: *sudo apt-get install python-pip*
- Install Pyomo by typing: *sudo pip install pyomo*

Note: To use Pyomo you need to install the solver separately.

Example

$$\max 1000x_1 + 2000x_2 + 3000x_3 \tag{1}$$

$$s.t.:$$

$$x_1 + 2x_2 + 3x_3 \leq 10$$

$$x_2 + 2x_3 \leq 5$$

$$x_1, x_2, x_3 \geq 0$$

Pyomo Code

```
from __future__ import division
from pyomo.environ import *
from pyomo.opt import SolverFactory
model = AbstractModel()
```

Define Variable

model.x_1 = Var(domain=NonNegativeReals)

model.x_2 = Var(domain=NonNegativeReals)

model.x_3 = Var(domain=NonNegativeReals)

Define the Objective function

def obj_expression(model):

    return 1000 * model.x_1 + 2000 * model.x_2 + 3000 * model.x_3

model.OBJ = Objective(rule = obj_expression, sense = maximize)

Define the constraints

def constraint01_rule(model):

    return x_1 + 2 * model.x_2 + 3 * model.x_3 $\leq$ 10

model.Constraint01 = Constraint(rule=constraint01_rule)

def constraint02_rule(model):

    return model.x_2 + 2 * model.x_3 $\leq$ 5

model.Constraint02 = Constraint(rule=constraint02_rule)

Karnataka Engineering Company Problem

- The problem statement is given in KEC.pdf file.
- Data for solving this problem is given in kecModelData.dat file

Pyomo Code

```
from pyomo.environ import *
from pyomo.opt import SolverFactory
from pyomo.opt import SolverStatus, TerminationCondition
model = AbstractModel()
```

Declare Set

```
model.SupplyRegion = Set()
model.DemandRegion = Set()
```

Define Parameter

model.distances = Param(model.SupplyRegion, model.DemandRegion)

model.lowcapacity = Param(model.SupplyRegion)

model.highcapacity = Param(model.SupplyRegion)

model.costperkm = Param()

model.fixedcosts = Param(model.SupplyRegion)

model.demand = Param(model.DemandRegion)

model.productioncosts = Param(model.SupplyRegion)

model.lowcostperkm = Param()

model.highcostperkm = Param()

model.productMoved = Param()

Define Variable

model.open1 = Var(model.SupplyRegion, domain = Binary)

model.qtyship = Var(model.SupplyRegion, model.DemandRegion, domain = NonNegativeIntegers, initialize = 0)

model.shipcosts = Var(model.SupplyRegion, model.DemandRegion, domain = NonNegativeIntegers, initialize = 0)

model.y = Var(model.SupplyRegion, model.DemandRegion, domain = Binary)

model.z = Var(model.SupplyRegion, model.DemandRegion, domain = Binary)

model.v = Var(model.SupplyRegion, model.DemandRegion, domain = Binary)

Define Objective function

```
def obj_expression(model):
    return sum(model.fixedcosts[i] * model.open1[i] for i in model.SupplyRegion) +
   sum(model.shipcosts[i,j] * model.distances[i,j] for i in model.SupplyRegion
   for j in model.DemandRegion) + sum(model.productioncosts[i] *
   model.qtyship[i,j] for i in model.SupplyRegion for j in model.DemandRegion)
model.OBJ = Objective(rule=obj_expression, sense = minimize)
```

Define Constraints

```
def constraint01_rule(model, j):
    return sum(model.qtyship[i,j] for i in model.SupplyRegion)== model.demand[j]
model.Constraint01 = Constraint(model.DemandRegion, rule=constraint01_rule)

def constraint02_rule(model, i):
    return sum(model.qtyship[i,j] for j in model.DemandRegion) ≤
    model.highcapacity[i] * model.open1[i]
model.Constraint02 = Constraint(model.SupplyRegion, rule=constraint02_rule)

def constraint03_rule(model, i):
    return sum(model.qtyship[i,j] for j in model.DemandRegion) ≥
    model.lowcapacity[i] * model.open1[i]
model.Constraint03 = Constraint(model.SupplyRegion, rule=constraint03_rule)
```

Define Constraints

```
def constraint04_rule(model, i, j):
    return model.qtyship[i,j] ≤ model.productMoved + 40 * model.z[i,j]
model.Constraint04 = Constraint(model.SupplyRegion, model.DemandRegion,
rule=constraint04_rule)

def constraint05_rule(model, i, j):
    return model.qtyship[i,j] ≥ model.productMoved - 40 * model.y[i,j]
model.Constraint05 = Constraint(model.SupplyRegion, model.DemandRegion,
rule=constraint05_rule)

def constraint06_rule(model, i, j):
    return model.y[i,j] + model.z[i,j] == 1
model.Constraint06 = Constraint(model.SupplyRegion, model.DemandRegion,
rule=constraint06_rule)
```

Define Constraints

```
def constraint07_rule(model, i, j):
    return model.shipcosts[i,j] ≥ model.highcostperkm * model.qtyship[i,j] - 1000 * model.z[i,j]
model.Constraint07 = Constraint(model.SupplyRegion, model.DemandRegion, rule=constraint07_rule)

def constraint08_rule(model, i, j):
    return model.shipcosts[i,j] ≥ model.lowcostperkm * model.qtyship[i,j] - 1000 * model.y[i,j]
model.Constraint08 = Constraint(model.SupplyRegion, model.DemandRegion, rule=constraint08_rule)
```

Nonlinear Programming

- Pyomo makes use of the interface provided by the AMPL Solver Library to provide efficient expression evaluation and automatic differentiation.
- Use of the AMPL Solver Library means that any AMPL-enabled solver should be usable as a solver within the Pyomo framework.

General Nonlinear programming formulation:

$$
\begin{aligned}
\min_x f(x) \\
s.t.\;\; c(x) = 0 \\
d^L \leq d(x) \leq d^U \\
x^L \leq x \leq x^U
\end{aligned} \tag{2}
$$

- Pyomo has been tested with local and global solvers that typically assume that these functions are continuous and smooth, with continuous first (and possibly second) derivatives.

Rosenbrock function

- It is a famous unconstrained nonlinear optimization problem.

$$\min_{x,y} f(x,y) = (1-x)^2 + 100(y-x^2)^2 \tag{3}$$

Pyomo Model

- first the necessary packages are imported, and then a model object is created.

*from pyomo import **

model = AbstractModel()

Define Variable

- The model creates two variables x and y and initializes each of them to a value of 1.5

model.x = Var(initialize = 1.5)   model.y = Var(initialize = 1.5)

Define Objective function

```
def rosenbrock(model):
    return (1.0-model.x)**2 + 100.0*(model.y - model.x**2)**2
model.obj = Objective(rule=rosenbrock, sense=minimize)
```

- Run the following to solve the problem.

```
pyomo –solver=ipopt –summary Rosenbrock.py
```