Introduction to Linear Optimization in Python

Using PuLP and Pyomo

Optimization Problems

- Model real world situations mathematically
- Add constraints to the problem
- Add an objective
- Find the optimal solution

Linear Optimization

- Restricted to linear functions

$$a_1 x_1 + a_2 x_2 + \ldots + a_n x_n < comparison >< value >$$

- Objective function: $Min \sum_{j=1}^{n} c_j x_j$

- Such that, Constraints:

$$\sum_{j=1}^{n} a_{ij} x_i = b_i \quad , i = 1 \ldots m$$

Basic Terminology

- If x satisfies Ax=b, x >= 0, then x is feasible
- An *Optimal* solution is feasible, and results in the lowest value of the objective function
    - Assuming we are minimizing the objective.

Modeling Problems

- Describe real problems with linear equations
- Apply constraints as linear equations
- Provide an objective function
- Define this all in an LP language
- Solve the model

Fundamental assumptions

About linear programming

- Deterministic Property
  - Values are fixed, not probabilistic or stochastic
  - Use sensitivity analysis to explore uncertainty
- Divisibility
  - Ensured if we let all numbers be real
  - Integer linear problems are a special class of problem
- Linearity
  - Objects and constrains must be linear functions
  - Or, piecewise linear

Ex1: Production Planning

- We manufacture 3 products. P1,P2, and P3
- They sell for $20, $15, and $17 respectively
- Manufactured on 2 machines, M1 and M2
- Each is processed on both machines
- Machines run for 8h, and 9h, and then require maintenance.

Ex1: continued

- Each product has a processing time for each machine

	P1	P2	P3
M1	3	5	4
M2	6	1	3

- Production costs on M1,M2 are $120/hr and $90/hr
    - Or $1.50 and $2/minute
- Product costs are $15, $11.50 and $12.50
- So, profit on products $5, $3.50 and $4.50

Ex1: Continued

- Define: quantity of P1, P2 and P3 as x1, x2, x3
- Objective: Max z = 5*x1 + 3.5*x2 + 4.5*x3
- Since each machine has a limited bandwidth, we have constraints:
  - 3*x1+5*x2+4*x2 <= 540
  - 6*x1+1*x2+3*x3 <= 480
  - x1, x2, x3 >= 0
- Solution: x1=20, x2=0, x3=120

Example 2: Tennis Roster

- 16 players, 8 men, 8 women
- 4 courts, 3 time slots.
- Schedule so men/women only face each other once.
- Schedule to mix men/women partners
- Minimize the strength difference across the court
  - Don’t put 2 4.5’s against 2 3.5’s

Define the problem

```
self.prob = LpProblem("TennisBlock Optimization", LpMinimize)
```

Constraints

```
def add_core_constraints(self, slots,sides,courts,assign_m, assign_f):
      """
      Add the core constrains to make a viable game.

      a) Each man and woman can be in a single place at one time.
      b) Each side of each court in each time slot can have one man and one woman
      """

      # Constraint a
      for player in self.men:
          for slot in slots:
              c = lpSum([assign_m[player][slot][si][c] for c in courts for si in sides]) == 1, "One place at a time:{}
_{}".format(player,slot)
              self.prob += c

      for player in self.women:
          for slot in slots:
              c = lpSum([assign_f[player][slot][si][c] for c in courts for si in sides]) == 1, "One place at a time:{}
_{}".format(player,slot)
              self.prob += c

      # Constraint b
      for court in courts:
          for slot in slots:
              for side in sides:
                  self.prob += lpSum([assign_m[player][slot][side][court] for player in self.men]) == 1, "One man per side:{}_{}
_{}".format(court, slot, side)
      for court in courts:
          for slot in slots:
              for side in sides:
                  self.prob += lpSum([assign_f[player][slot][side][court] for player in self.women]) == 1, "One women per side:{}_{}
_{}".format(court, slot, side)
```

Constraints

```
def add_pair_constraints(self,slots,sides,courts,assign_m, assign_f):
    """
    Add constraints that men/women can't play against the same man/woman
    more than 1 time.

    :param slots:
    :param sides:
    :param courts:
    :param assign_m:
    :param assign_f:
    :return:
    """

    # Add constraint that men/women can't play against each other more than once.
    men_pairs = [tuple(c) for c in permutation(self.men, 2)]
    women_pairs = [tuple(c) for c in permutation(self.women, 2)]

    # Setup constraints for opponents
    # Pairs of men can only face off at most 1 times per night
    # Pairs of women can only face off at most 1 time per night
    for pr in men_pairs:
        self.prob += lpSum([assign_m[p][sl][si][c]
                            for p in pr
                            for c in courts
                            for sl in slots
                            for si in sides]) <= 2, ""

    for pr in women_pairs:
        self.prob += lpSum([assign_f[p][sl][si][c]
                            for p in pr
                            for c in courts
                            for sl in slots
                            for si in sides]) <= 2, ""
```

Objective

```
def add_objective(self):
    """
    Add the objective function to the problem.
    :return:
    """
    courts = self.C
    slots = self.T
    sides = self.S

    am = self.assign_m
    af = self.assign_f

    un = self.untrpdict

    exp = []
    for slot in slots:
        for court in courts:

            side_strength = []
            for side in sides:
                # Example: assign_m[player][slot][si][c]
                rating = lpSum(un[p] * am[p][slot][side][court] for p in self.men)
                rating += lpSum(un[p] * af[p][slot][side][court] for p in
self.women)
                side_strength.append(rating)

            exp.append(side_strength[0]-side_strength[1])

    total_variation = lpSum(exp)
    self.prob += total_variation, 'objective'
```

Solve

- See example code and runtime.
- Still need to refine the objective.
- Can’t do ABS.. so, learning to handle that
  - it’s nonlinear
- It’s been a learning adventure!

References

- Operations Research

  A model based approach

  H.A. Iselt, Carl-Louis Sandblom

- An Introduction to Linear Programming

  Goemans, Michael X, MIT

Questions?