Getting Started with Gurobi

Abrémod Training

July 1, 2020

Overview

- Linear Programming (LP)
- Solving LPs with Gurobi
  - Interactive Shell (Python)
- LP Modeling Techniques
- Multi-Objective Optimization
- Integer Programming
- Performance Tuning
- Column Generation
- Convex Quadratic Programming
- Stochastic Programming

Abrémod

Abrémod specializes in implementing math programming models to solve business problems. Including business analysis, modeling, and implementation.

- Revenue Management
- Assignment/Scheduling Problems
- Network Optimization

Gurobi

Gurobi is a state-of-the-art solver for mathematical programming. It includes solvers for the following types of models:

- Linear Programming (LP)
- Mixed-Integer Linear Programming (MILP)
- Quadratic Programming (QP)
- Mixed-Integer Quadratic Programming (MIQP)
- Quadratically Constrained Programming (QCP)
- Mixed-Integer Quadratically Constrained Programming (MIQCP)

Here, "program" does not refer to a computer program but rather a schedule.

Gurobi

- The problems Gurobi solves are all special cases of mathematical programs.
- Before we discuss those special cases in detail, we should understand what a math program looks like and define some related terminology.

What is a Mathematical Program?

A math program consists of three components:

- Decision Variables (what you control)
- Constraints (rules you must follow)
- Objective Function (what you want to minimize/maximize)

What is a Mathematical Program?

Definition

$$
\begin{aligned}
\text{minimize/maximize:} \quad & f(x_1, x_2, \ldots, x_n) \\
\text{subject to:} \quad & g_i(x_1, x_2, \ldots, x_n) \begin{Bmatrix} \leq \\ \geq \\ = \end{Bmatrix} b_i, \quad i = 1, \ldots, m \\
& x_j \geq 0, \quad j = 1, \ldots, n
\end{aligned}
$$

Math Programming Terminology

- x_j are the *decision variables*.
- $g_i(x_1, x_2, \ldots, x_n) \left\{ \begin{matrix} \leq \\ \geq \\ = \end{matrix} \right\} b_i$ are *structural constraints*.
- $x_j \geq 0$ are *nonnegativity constraints*.
- $f(x_1, \ldots, x_n)$ is the *objective function*.
- A *feasible solution*, $\hat{x} = (\hat{x}_1, \ldots, \hat{x}_n)$ satisfies all constraints.
- The *feasible region* is the set of all feasible solutions.
- The objective function ranks the feasible solutions.
- The optimal solution x^* satisfies $f(x^*) \leq f(\hat{x})$ for all feasible $\hat{x}$.
  - x^* is feasible itself.

The Linear Program

$$
\begin{aligned}
z^* = \min_x \quad & c_1 x_1 + c_2 x_2 + \cdots + c_n x_n \\
\text{subject to:} \quad & a_{i1} x_1 + a_{i2} x_2 + \cdots + a_{in} x_n \begin{Bmatrix} \le \\ \ge \\ = \end{Bmatrix} b_i, \quad i = 1, \ldots, m \\
& x_1, x_2, \ldots, x_n \ge 0
\end{aligned}
$$

In standard matrix form:

$$
\begin{aligned}
z^* = \min_x \quad & c^T x \\
\text{s.t.} \quad & Ax = b \\
& x \ge 0
\end{aligned}
$$

The Linear Program

$$
\begin{aligned}
z^* = \min_{x} \quad & c_1 x_1 + c_2 x_2 + \cdots + c_n x_n \\
\text{subject to:} \quad & a_{i1} x_1 + a_{i2} x_2 + \cdots + a_{in} x_n \left\{ \begin{matrix} \le \\ \ge \\ = \end{matrix} \right\} b_i, \quad i = 1, \ldots, m \\
& x_1, x_2, \ldots, x_n \ge 0
\end{aligned}
$$

- a_{ij}, c_j, and b_i are data.
- Find x^* satisfying $c_1 x_1^* + \cdots + c_n x_n^* \le c_1 \hat{x}_1 + \cdots + c_n \hat{x}_n$ for all feasible $\hat{x}$.
- A linear program (LP) is a special type of math program with:
  - $f(x_1, \ldots, x_n) = c_1 x_1 + \cdots + c_n x_n$
  - $g_i(x_1, \ldots, x_n) = a_{i1} x_1 + \cdots + a_{in} x_n, \quad i = 1, \ldots, m$

Linear Programming Axioms

$$
\begin{aligned}
z^* = \min_{x} \quad & c_1x_1 + c_2x_2 + \cdots + c_nx_n \\
\text{subject to:} \quad & a_{i1}x_1 + a_{i2}x_2 + \cdots + a_{in}x_n \left\{ \begin{matrix} \leq \\ \geq \\ = \end{matrix} \right\} b_i, \quad i = 1, \ldots, m \\
& x_1, x_2, \ldots, x_n \geq 0
\end{aligned}
$$

- Additivity
- Proportionality
- Divisibility
- Certainty

Linear Programming Transformations

Gurobi will automatically perform the following transformations to get into standard form:

- Maximize by minimizing the negative
  - $\max_x cx \Leftrightarrow \min_x -cx$
- Add a slack or surplus variable to convert inequality into equality
  - $ax \geq b \Leftrightarrow ax - s = b, s \geq 0$
  - $ax \leq b \Leftrightarrow ax + s = b, s \geq 0$
- Write a variable that can be positive or negative as the difference of non-negative variables
  - $x = x^+ - x^-$
  - $x^+, x^- \geq 0$

The Diet Problem

- Suppose you are trying to construct a diet out of a given set of foods, each with a different cost and nutritional composition, and wish to meet some minimum requirements of various nutrients.
- How can you find the combination of foods that meets all the nutrient requirements and minimizes cost?

The Diet Problem

Let's consider the following sample inputs where there are 5 types of food and 2 nutrient requirements.

Units of nutrients and cost per ounce

Food type	Iron	Calcium	Cost
1	2	0	20
2	0	1	10
3	3	2	31
4	1	2	11
5	2	1	12

Nutrient requirements: 21 units of iron and 12 units of calcium

Diet Problem Formulation

Food	Iron	Calcium	Cost
1	2	0	20
2	0	1	10
3	3	2	31
4	1	2	11
5	2	1	12

Nutrient requirements:
Iron: 21, Calcium: 12

- Decision Variables
  - $x_j = \#$ of ounces of food type $j = 1, 2, \ldots, 5$
- Objective Function
  - $\min z = 20x_1 + 10x_2 + 31x_3 + 11x_4 + 12x_5$
- Structural Constraints
  - $2x_1 + 0x_2 + 3x_3 + 1x_4 + 2x_5 \geq 21$
  - $0x_1 + 1x_2 + 2x_3 + 2x_4 + 1x_5 \geq 12$
- Nonnegativity constraints
  - $x_j \geq 0, j = 1, 2, \ldots, 5$

Solving with the Gurobi Interactive Shell

- Objects and Methods you will need
  - Model
    - addVar(lb, ub, obj, vtype, name)
    - addConstr(constr, name)
    - update()
    - optimize()
  - Var
    - X
    - RC
  - Constr
    - Pi
    - Slack

Querying the Solution

- Model.write()
  - m.write('diet.lp') outputs the model in human-readable form
  - m.write('diet.mps') outputs a full-precision copy of the model
  - m.write('diet.sol') outputs the solution
- Model object attributes
  - m.Status (was the solver able to find an optimal solution?)
  - m.objVal() (optimal objective value)

Querying the Solution

- Var object attributes
  - x1.X - optimal value
  - x1.RC - reduced cost, change in objective/change in variable bound
- Constr object attributes
  - iron_constraint.Pi - shadow price, change in objective/change in RHS
  - iron_constraint.Slack - difference between LHS and RHS

Updating the Model

- Settable Var object attributes
  - x1.LB - lower bound
  - x1.UB - upper bound
  - x1.Obj - objective coefficient
- Settable Constr object attributes
  - iron_con.RHS - right-hand side constant
- model.chgCoeff(constr, var, newvalue) modifies a coefficient in the constraint matrix

Diet Problem Implemented in Python

```
import gurobipy as grb
m = grb.Model()
x1 = m.addVar(obj=20, name='consumed.1')
x2 = m.addVar(obj=10, name='consumed.2')
x3 = m.addVar(obj=31, name='consumed.3')
x4 = m.addVar(obj=11, name='consumed.4')
x5 = m.addVar(obj=11, name='consumed.5')
m.update()
iron_constr = m.addConstr(2*x1 + 3*x3 + x4 + 2*x5 >= 21, name='nutrient.iron')
calcium_constr = m.addConstr(x2 + 2*x3 + 2*x4 + x5 >= 12, name='nutrient.calcium')
m.update()
m.optimize()
for var in m.getVars():
    print (var.VarName, var.X, var.RC)
```

Python Objects and Methods So Far

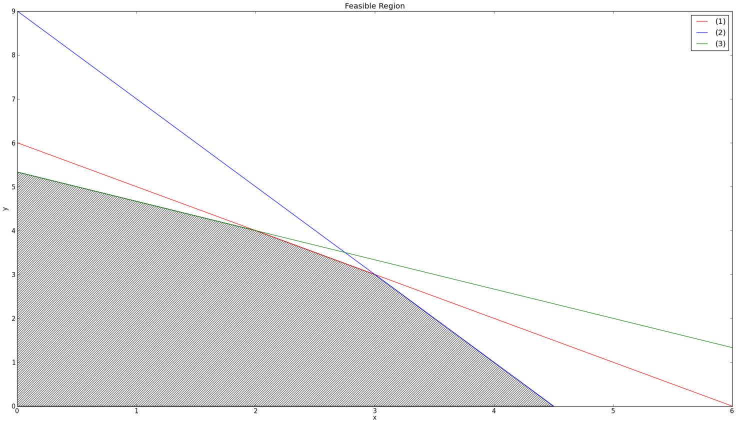
- grb.Model()
- model.addVar(float lb, float ub, float obj, string type, string name, Column column)
- model.addConstr(LinExpr lhs, char sense, LinExpr rhs, String name)
- model.update()
- model.optimize()
- model.getVars(), model.getConstrs()

Exercises

- How would the optimal cost change if we forced $x_1 \geq 0.1$?
- How would the optimal cost change if we required an extra 0.1 units of iron?

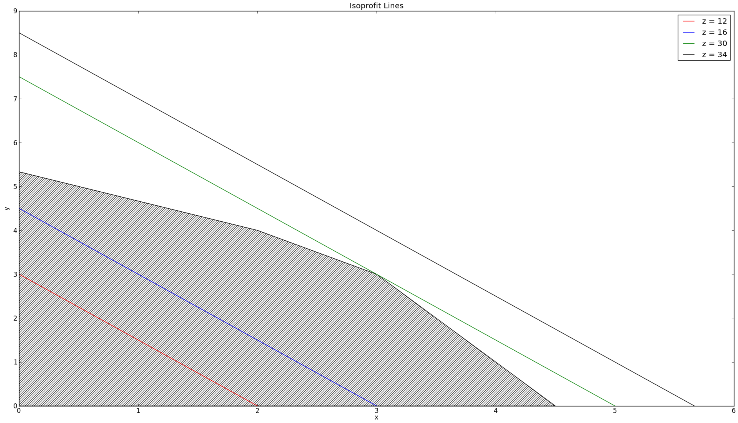
Linear Programming Geometry

$$
\begin{aligned}
\max_{x,y} \quad & z = 6x + 4y \\
\text{s.t.} \quad & x + y \leq 6 && (1) \\
& 2x + y \leq 9 && (2) \\
& 2x + 3y \leq 16 && (3) \\
& x, y \geq 0
\end{aligned}
$$



Linear Programming Geometry

- Plot the objective function $z = 6x + 4y$ for some fixed values of z.
- These are the so-called *isoprofit lines* or *objective function contours*.
- Increasing z results in a parallel shift to the right.



Observations

- Feasible region of a linear program is always a convex polyhedron
- At least one optimal solution occurs at a corner point (a.k.a. extreme point or vertex) of this polyhedron
- Infinitely-many points in the feasible region, but only finitely many corner points

Linear Programming Algebra

First, how to solve linear systems of equations?

$$\begin{aligned} 2x_1 + 1x_2 + 1x_3 &= 4 \\ 4x_1 - 6x_2 + 0x_3 &= 2 \\ -2x_1 + 7x_2 + 2x_3 &= 1 \end{aligned}$$

Systematically perform row operations to form equivalent systems

$$\begin{gathered} \text{Row } 1 \leftarrow \tfrac{1}{2} \text{ Row } 1 \\ \text{Row } 2 \leftarrow \text{Row } 2 \text{ - } 4 \text{ Row } 1 \\ \text{Row } 3 \leftarrow \text{Row } 3 + 2 \text{ Row } 1 \\ \vdots \end{gathered}$$

Until we arrive at an equivalent system with an obvious solution

$$\begin{aligned} 1x_1 + 0x_2 + 0x_3 &= 2 \\ 0x_1 + 1x_2 + 0x_3 &= 1 \\ 0x_1 + 0x_2 + 1x_3 &= -1 \end{aligned}$$

Linear Programming Algebra

- These row operations correspond to multiplying equations by constants and adding the result to other equations.
- "Systematically" performing row operations means picking an equation and solving for a specific variable, then eliminating that variable in all other equations.
- For a square system that has an equal number of variables and equations, it is relatively easy to decide which equation to solve and which variable to solve for.
- If the system has a unique solution, we can solve for the ith variable in the ith equation, swapping the order of equations as needed.

Linear Programming Algebra

Linear programs typically have:

- More variables than equations.
- More than one feasible solution (almost always).
- More than one optimal solution (more often than you might think).

This being said, we can still solve LPs via systematic row operations, but the variable selection step is a little trickier.

Linear Programming Algebra

$$
\begin{aligned}
\max_{x,y} \quad & 6x + 4y = z \\
\text{s.t.} \quad & x + y \leq 6 \\
& 2x + y \leq 9 \\
& 2x + 3y \leq 16 \\
& x, y \geq 0
\end{aligned}
$$

As a system of linear equations:

$$
\begin{aligned}
\max_{x,y,s} \quad & 6x + 4y + 0s_1 + 0s_2 + 0s_3 = z \\
\text{s.t.} \quad & 1x + 1y + 1s_1 + 0s_2 + 0s_3 = 6 \\
& 2x + 1y + 0s_1 + 1s_2 + 0s_3 = 9 \\
& 2x + 3y + 0s_1 + 0s_2 + 1s_3 = 16 \\
& x, y, s_1, s_2, s_3 \geq 0
\end{aligned}
$$

Add the objective as the equation $-z + 6x + 4y = 0$ and write in matrix form:

$$
\left[\begin{array}{cccccc|c}
z & x & y & s_1 & s_2 & s_3 & RHS \\
-1 & 6 & 4 & 0 & 0 & 0 & 0 \\
0 & 1 & 1 & 1 & 0 & 0 & 6 \\
0 & 2 & 1 & 0 & 1 & 0 & 9 \\
0 & 2 & 3 & 0 & 0 & 1 & 16
\end{array}\right]
$$

Linear Programming Algebra

$$\begin{bmatrix} z & x & y & s_1 & s_2 & s_3 & RHS \\ -1 & 6 & 4 & 0 & 0 & 0 & 0 \\ 0 & 1 & 1 & 1 & 0 & 0 & 6 \\ 0 & 2 & 1 & 0 & 1 & 0 & 9 \\ 0 & 2 & 3 & 0 & 0 & 1 & 16 \end{bmatrix}$$

Systematically perform row operations

$$\begin{bmatrix} z & x & y & s_1 & s_2 & s_3 & RHS \\ -1 & 0 & 1 & 0 & -3 & 0 & -27 \\ 0 & 0 & 1/2 & 1 & -1/2 & 0 & 3/2 \\ 0 & 1 & 1/2 & 0 & 1/2 & 0 & 9/2 \\ 0 & 0 & 2 & 0 & -1 & 1 & 7 \end{bmatrix}$$

Until the solution is obvious

$$\begin{bmatrix} z & x & y & s_1 & s_2 & s_3 & RHS \\ -1 & 0 & 0 & -2 & -2 & 0 & -30 \\ 0 & 0 & 1 & 2 & -1 & 0 & 3 \\ 0 & 1 & 0 & -1 & 1 & 0 & 3 \\ 0 & 0 & 0 & -4 & 1 & 1 & 1 \end{bmatrix}$$

What is the obvious solution? This is the equivalent LP:

$$\begin{aligned} \max_{x,y,s} \quad & 0x + 0y - 2s_1 - 2s_2 + 0s_3 + 30 = z \\ \text{s.t.} \quad & 0x + 1y + 2s_1 - 1s_2 + 0s_3 = 3 \\ & 1x + 0y - 1s_1 + 1s_2 + 0s_3 = 3 \\ & 0x + 0y - 4s_1 + 1s_2 + 1s_3 = 1 \\ & x, y, s_1, s_2, s_3 \geq 0 \end{aligned}$$

The optimal solution to this transformed LP is $(x, y, s_1, s_2, s_3) = (3, 3, 0, 0, 1)$, $z^* = 30$

Linear Programming Algebra

In more detail

$$\begin{array}{c} \begin{array}{cccccc|c} z & x & y & s_1 & s_2 & s_3 & RHS \end{array} \\ \left[\begin{array}{cccccc|c} -1 & 6 & 4 & 0 & 0 & 0 & 0 \\ 0 & 1 & 1 & 1 & 0 & 0 & 6 \\ 0 & 2 & 1 & 0 & 1 & 0 & 9 \\ 0 & 2 & 3 & 0 & 0 & 1 & 16 \end{array}\right] \end{array}$$

Feasible solution $(x, y, s_1, s_2, s_3) = (0, 0, 6, 9, 16)$, $z = 0$. Increase x since it has a positive coefficient.

$$\begin{array}{c} \begin{array}{cccccc|c} z & x & y & s_1 & s_2 & s_3 & RHS \end{array} \\ \left[\begin{array}{cccccc|c} -1 & 0 & 1 & 0 & -3 & 0 & -27 \\ 0 & 0 & 1/2 & 1 & -1/2 & 0 & 3/2 \\ 0 & 1 & 1/2 & 0 & 1/2 & 0 & 9/2 \\ 0 & 0 & 2 & 0 & -1 & 1 & 7 \end{array}\right] \end{array}$$

Feasible solution $(x, y, s_1, s_2, s_3) = (9/2, 0, 3/2, 0, 7)$, $z = 27$. Increase y since it has a positive coefficient.

$$\begin{array}{c} \begin{array}{cccccc|c} z & x & y & s_1 & s_2 & s_3 & RHS \end{array} \\ \left[\begin{array}{cccccc|c} -1 & 0 & 0 & -2 & -2 & 0 & -30 \\ 0 & 0 & 1 & 2 & -1 & 0 & 3 \\ 0 & 1 & 0 & -1 & 1 & 0 & 3 \\ 0 & 0 & 0 & -4 & 1 & 1 & 1 \end{array}\right] \end{array}$$

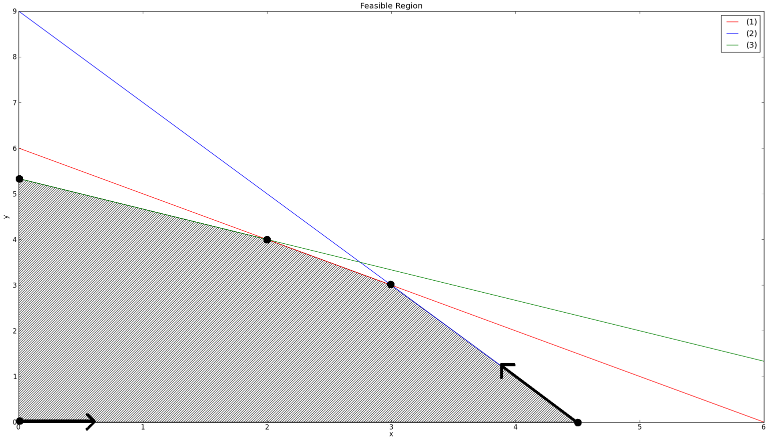
Feasible solution $(x, y, s_1, s_2, s_3) = (3, 3, 0, 0, 1)$, $z = 30$.
Transformed objective has no positive coefficients.
Transformed constraints have an “obvious” solution in which all variables with a negative objective coefficient are zero.
This is a provably optimal solution.

Observations

- Each iteration maintained exactly 3 positive decision variables (one for each of the original structural constraints).
- The set of positive variables is *basis* (and the associated solution is a *basic feasible solution*).
- Each iteration adds a new variable to a basis, and kicks an old variable out.
- The objective improves at each iteration.
- Proof of optimality: transformed objective coefficients are zero for basic variables, non-positive for non-basic variables.
- What would change if we perturbed the original
  - objective function coefficients?
  - right-hand sides?

Connecting the Algebra to the Geometry

We iterated over basic feasible solutions $(0, 0, 6, 9, 16)$, $(9/2, 0, 3/2, 0, 7)$, and $(3, 3, 0, 0, 1)$. Plotting these points in (x, y) space...



Exercises

- Solve the preceding model with Gurobi.
    - Note: Set the model attribute ModelSense to -1 in order to maximize
- Which constraints are tight at the optimal solution?
- Besides the origin, the feasible region has three other extreme points that are suboptimal under the current objective function. How might you change the objective function coefficients so that:
    - the extreme point at $(0, \frac{16}{3})$ is optimal?
    - the extreme point at $(\frac{9}{2}, 0)$ is optimal?
    - all points between $(2, 4)$ and $(3, 3)$ are optimal?

Possible Outcomes of an LP

- Infeasible - feasible region is empty, i.e., $x_1 \geq 0$, $x_1 \leq -1$
- Unbounded - no finite optimum, i.e. $\max\ 15x_1$ subject to $x_1 \geq 0$
- Multiple optima, i.e. $\max\ 3x_1 + 3x_2$ subject to $x_1 + x_2 \leq 1$ and non-negative
- Unique optimal solution (as in previous example)

Linear Programming Solvers

- Finds **an** optimal solution if feasible region is non-empty and objective function is bounded
- Might be multiple optimal solutions
  - Which optimal solution returned is not defined.
- Constraints are "hard"
  - Won't violate constraints [1] even if it helps the objective.
  - Will tell if relaxing a constraint would help the objective.
- Solver makes no apologies for this behavior!
- Includes proof of optimality

[1] by more than the numerical tolerance

Real Linear Programming Solvers

Strengths

- Can solve very large problems in practice
  - Tuned for real-world business problems
  - Routinely solve problems with 10^7 variables and constraints
  - Multiple algorithms (controlled by parameter Method)

Weaknesses

- Don't guarantee a time to a solution
- Might not reach optimality (but will tell you if it doesn't)
- Work with floating point values
  - Might violate constraints by a small tolerance (controlled by parameter FeasibilityTol)
  - Might return a solution that is within some numerical tolerance of optimal (controlled by parameter OptimalityTol)

The better the linear programming solver, the less of an issue you will have with these realities.

Duality

The diet problem, revisited:

- Let's take the perspective of a supplement vendor, who has pills that contain a single unit of iron or calcium that can be used to replace meals.
- This vendor will attempt sell these pills to a dieter, and must determine the appropriate price to offer.
- We'll assume the dieter knows how to solve the diet problem and will replace her optimal diet with pills but only if her cost does not increase.
- How does the vendor determine pill prices that maximize revenue and are competitive with the food types?

Duality

Food	Iron	Calcium	Cost
1	2	0	20
2	0	1	10
3	3	2	31
4	1	2	11
5	2	1	12

Nutrient requirements:
Iron: 21, Calcium: 12

- Let π_i, π_c be the price to be charged for an iron, calcium pill.
- We wish to maximize total revenue of $v = 21\pi_i + 12\pi_c$.
- We must charge prices that are competitive with the prices of the five food types.
  - $2\pi_i \leq 20$
  - $\pi_c \leq 10$
  - $3\pi_i + 2\pi_c \leq 31$
  - $\pi_i + 2\pi_c \leq 11$
  - $2\pi_i + \pi_c \leq 12$

Duality

The diet problem:

$$
\begin{aligned}
z^* = \min_{x} \quad & 20x_1 + 10x_2 + 31x_3 + 11x_4 + 12x_5 \\
\text{s.t.} \quad & 2x_1 + 0x_2 + 3x_3 + 1x_4 + 2x_5 \geq 21 \\
& 0x_1 + 1x_2 + 2x_3 + 2x_4 + 1x_5 \geq 12 \\
& x_j \geq 0, \;\; j = 1, 2, \ldots, 5
\end{aligned}
$$

The “dual” problem:

$$
\begin{aligned}
v^* = \max_{\pi} \quad & 21\pi_i + 12\pi_c \\
\text{s.t.} \quad & 2\pi_i + 0\pi_c \leq 20 \\
& 0\pi_i + 1\pi_c \leq 10 \\
& 3\pi_i + 2\pi_c \leq 31 \\
& 1\pi_i + 2\pi_c \leq 11 \\
& 2\pi_i + 1\pi_c \leq 12 \\
& \pi_i, \pi_c \geq 0
\end{aligned}
$$

Exercises

- Intuitively, is it possible for $v^* > z^*$?
- Solve the dual of the diet problem with Gurobi. (Maintain a copy of the Model object for the original diet problem for comparison purposes.)
- How are z^* and v^* related?
- How are π_i^* and π_c^* related to solution of the original diet problem?
- Multiply the iron constraint in the original diet problem by π_i^*, the calcium constraint by π_c^*, and add the results.
  - What is the resulting inequality?
  - How can this inequality be used to prove optimality?

Computing Shadow Prices

Gurobi computes π for us even when we solve the primal. Recall that in the optimal solution to the diet problem example, only x_4 and x_5 were non-zero. Letting b_i and b_c be nutrient requirements, we have

$$\begin{aligned} x_4 + 2x_5 &= b_i \\ 2x_4 + x_5 &= b_c \end{aligned}$$

We can solve for x_4 and x_5 as

$$\begin{aligned} x_4 &= -1/3b_i + 2/3b_c \\ x_5 &= 2/3b_i - 1/3b_c \end{aligned}$$

Plugging into the objective, we get

$$\begin{aligned} z &= 11x_4 + 12x_5 \\ &= 11(-1/3b_i + 2/3b_c) + 12(2/3b_i - 1/3b_c) \\ &= 13/3b_i + 10/3b_c \end{aligned}$$

So, $\pi_i = 13/3$ and $\pi_c = 10/3$.

Computing Reduced Costs

- How are the reduced costs related to the shadow prices?
- Consider food 3, which costs 31 per ounce and provides 3 units of iron and 2 units of calcium.
- Iron is priced at $13/3$ per unit, calcium at $10/3$ per unit.
- If we discount the cost of food 3 by the value of the nutrients that it provides, we get $31 - 3 * (13/3) - 2 * (10/3) = 34/3$, which is exactly the reduced cost.
- What is the reduced cost for food 4?
- How cheap would food type 3 need to be in order for it to be in our diet?
- Suppose we introduce a new food that costs 20 per ounce and provides 2 units of iron and 3 units of calcium. Should we include this new food in our diet? Do we need to reoptimize?

Notation

Linear Programming involves exclusively addition and multiplication by constants. The symbols $\in$ and $\notin$ is read as "in" and "not in". If $I = \{1, 2, 3\}$.

$$\sum_{i \in I} a_i x_i$$

$$a_1 x_1 + a_2 x_2 + a_3 x_3$$

$$x_i \leq b_i \quad \forall i \in I$$

$$\begin{aligned} x_1 &\leq b_1 \\ x_2 &\leq b_2 \\ x_3 &\leq b_3 \end{aligned}$$

Knapsack Problem

- Set of items $I = \{1, \ldots, n\}$.
- Each item has value v_i and weight w_i.
- Have a knapsack that with capacity b.
- Can take part of any item.
- What is the highest valued collection of items (and partial items) that can go into the knapsack?

$$
\begin{aligned}
\max_x \quad & \sum_{i \in I} v_i x_i \\
\text{s.t.} \quad & \sum_{i \in I} w_i x_i \leq b \\
& 0 \leq x_i \leq 1, \quad i \in I
\end{aligned}
$$

- Use a LinExpr to build up the sum in the constraint.

Python Implementation

```
import gurobipy as grb
weights = [70, 73, 77, 80, 82, 87, 90, 94, 98, 106, 110, 113, 115, 118, 120]
values = [135, 139, 149, 150, 156, 163, 173, 184, 192, 201, 210, 214, 221, 229, 240
capacity = 750
m = grb.Model()
item_selected = []
for i in range(len(values)):
    item_selected.append(m.addVar(ub=1, obj=values[i], name='item_selected.' + str(
m.update()
total_weight = grb.quicksum(weights[i]*item_selected[i] for i in range(len(values))
weight_con = m.addConstr(total_weight <= capacity, name='total_weight')
m.update()
m.ModelSense = grb.GRB.MAXIMIZE
m.optimize()
for var in item_selected:
    print (var.VarName, var.X)
```

The Diet Problem Generalized

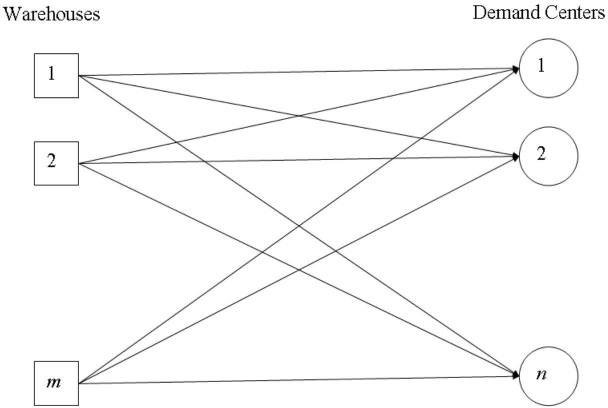
- Sets and Indices
  - $i \in I$: nutrients
  - $j \in J$: food types
- Data
  - c_j: per ounce cost of food type j
  - a_{ij}: quantity of nutrient i per ounce of food type j
  - l_i, u_i: min, max daily requirements for nutrient i
- Decision Variables
  - x_j: the number of ounces to consume of food type j.
- Formulation?
- Let π_i be the shadow price for nutrient i. What is the reduced cost of food type j, in terms of π?

Python Implementation

Add ranged constraints to model via model.addRange(expr, lower, upper, name)

```
def solve_diet_problem(nutrient_densities, costs, nutrient_requirements):
    m = grb.Model()
    ounces_consumed = {food_type: m.addVar(obj=cost, name='ounces_consumed.' + str(food_type))
                       for food_type, cost in costs.iteritems()}
    m.update()
    nutrient_constraints = {}
    food_types = costs.keys()
    for nutrient, (min_requirement, max_requirement) in nutrient_requirements.iteritems():
        nutrient_consumed = grb.quicksum(nutrient_densities[food_type, nutrient]*ounces_consumed[food_type]
                                         for food_type in food_types)
        constr = m.addRange(nutrient_consumed, min_requirement, max_requirement,
                            'nutrient.' + str(nutrient))
        nutrient_constraints[nutrient] = constr
    m.optimize()
    if m.status == GRB.OPTIMAL:
        return {food_type: var.X for food_type, var in ounces_consumed.iteritems()}
    raise Exception("Model was infeasible.")
```

Transportation Problem



Input:
Warehouse capacity u_i (widgets)
Customer demand d_j (widgets)
Shipping cost c_{ij} (\$/widget)

Transportation Problem

- Sets and Indices
  - $i \in I$: Warehouses
  - $j \in J$: Customers
- Data
  - u_i: capacity for warehouse i (widgets)
  - d_j: demand at demand center j (widgets)
  - c_{ij}: shipping cost from warehouse i to customer j (\$/widget)
- Decision Variables
  - x_{ij}: number of widgets to ship from warehouse i to customer j

LP Formulation

$$
\begin{aligned}
\min_{x} \quad & \sum_{i \in I} \sum_{j \in J} c_{ij} x_{ij} \quad \text{(minimize shipping costs)} \\
\text{s.t.} \quad & \sum_{i \in I} x_{ij} = d_j, \quad j \in J \quad \text{(satisfy demand)} \\
& \sum_{j \in J} x_{ij} \leq u_i, \quad i \in I \quad \text{(don't exceed capacity)} \\
& x_{ij} \geq 0, \quad i \in I, \ j \in J \quad \text{(ship nonnegative quantities)}
\end{aligned}
$$

Python Implementation

Use quicksum to build up the summations in the constraints.
Assume *to_ship* is a 2d array of vars and has already been populated.
Demand constraints $\sum_{i \in I} x_{ij} = d_j, \;\; j \in J$ are built via:

```
def get_demand_constrs(model, to_ship, demands):
    return [model.addConstr(grb.quicksum(to_ship[warehouse, customer]
                            for warehouse in warehouses) == demand,
                            name='demand.' + str(customer))
           for customer, demand in enumerate(demands)]
```

Capacity constraints $\sum_{j \in J} x_{ij} \leq u_i, \;\; i \in I$ are built via:

```
def get_capacity_constrs(model, to_ship, capacities):
    return [model.addConstr(grb.quicksum(to_ship[warehouse, customer]
                            for customer in customers) <= capacity,
                            name='capacity.' + str(warehouse))
            for warehouse, capacity in enumerate(capacities)]
```

Diagnosing Infeasibility

- After optimization, always check the Model attribute Status to determine whether the model was solved to optimality.
- If model is infeasible, Status will be GRB.Status.Infeasible
- model.computeIIS() computes an irreducible inconsistent subsystem.
- Pass model.write() a filename with suffix .ilp to write the IIS to a file.
- Var attributes IISLB, IISUB indicate which variable bounds participate in the IIS.
- Constr attributes IISConstr indicate which constraints participate in the IIS.

Exercise

- Implement the transportation model with Gurobi.
- Under what conditions would the problem become infeasible?
- Create an infeasible instance, compute an IIS, and write it to a file.

Elasticizing Constraints

To extend the transportation problem allow demand to go unsatisfied at a per-unit penalty of ρ replace the demand constraint with

$$\sum_{i \in I} x_{ij} = d_j - y_j, \;\; j \in J,$$

where $y_j \geq 0$, and add $\rho \sum_{j \in J} y_j$ to the objective.

Piecewise Linear Penalties

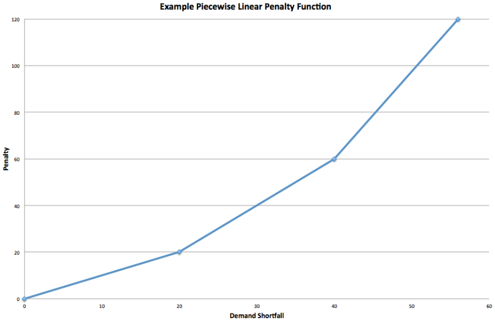
Penalize the first 20% of demand shortfall at a rate ρ, and any additional demand shortfall at a rate 1.5ρ.

$$\begin{aligned}\sum_{i\in I} x_{ij} = d_j - y_j^1 - y_j^2, \;\; j \in J \\ 0 \le y_j^1 \le 0.2 d_j, \;\; j \in J \\ 0 \le y_j^2, \;\; j \in J,\end{aligned}$$

and add $\rho \sum_{j\in J}(y_j^1 + 1.5 y_j^2)$ to the objective.

Piecewise Linear Penalties

- Starting with 6.0, Gurobi provides a method model.setPWLObj.



- The above penalty function can be created in one call:
  - model.setPWLObj(var, [0, 20, 40, 56], [0, 20, 60, 120])
- No auxiliary variables are required.
- Note: If the objective function is not convex, the resulting model will be an Integer Program.

Minimize the Maximum Demand Shortfall

- Recall: $\sum_{i \in I} x_{ij} = d_j - y_j, \;\; j \in J$
- y_j is the demand shortfall at demand center j.
- Suppose we want to control $z = \max\{y_1, y_2, \ldots, y_n\}$.
- Let $z \geq y_j, \;\; j \in J$.
- Penalize z in the objective, or put an upper bound on z.
- Only works if we are trying to minimize z, otherwise we require integer variables.
- Extends to any problem involving minimization (maximization) of the maximum (minimum) of several linear functions.
- $y = |x|$ can be linearized as $y \geq x$, $y \geq -x$ (assuming minimization of y).

Multiple Objectives

$$
\begin{aligned}
\min_{x} \quad & \sum_{i \in I} \sum_{j \in J} c_{ij} x_{ij} + \rho \sum_{j \in J} y_j \\
\text{s.t.} \quad & \sum_{i \in I} x_{ij} = d_j - y_j, \quad j \in J \\
& \sum_{j \in J} x_{ij} \leq u_i, \quad i \in I \\
& x_{ij} \geq 0, \quad i \in I,\ j \in J \\
& y_j \geq 0, \quad j \in J
\end{aligned}
$$

- Objectives:
  - Minimize transportation cost $\sum_{i \in I} \sum_{j \in J} c_{ij} x_{ij}$
  - Minimize demand shortfall $\sum_{j \in J} y_j$
- How do we choose ρ?

Multiple Objectives

- Generate an *efficient frontier* of solutions.
- Let $z = \sum_{j \in J} y_j$.
- Add ρz to the objective.
- Optimize with $\rho = 0$.
- Query Var attribute SAObjUp.
- Reoptimize with $\rho = (1 + \epsilon)$SAObjUp.
- LPs are cheap to reoptimize!

Exercise

Generate the efficient frontier of the transportation problem.

Primary and Secondary Objectives

- Optimize primary objective first.
- Constrain primary objective to be within some tolerance of optimal.
- Optimize secondary objective.

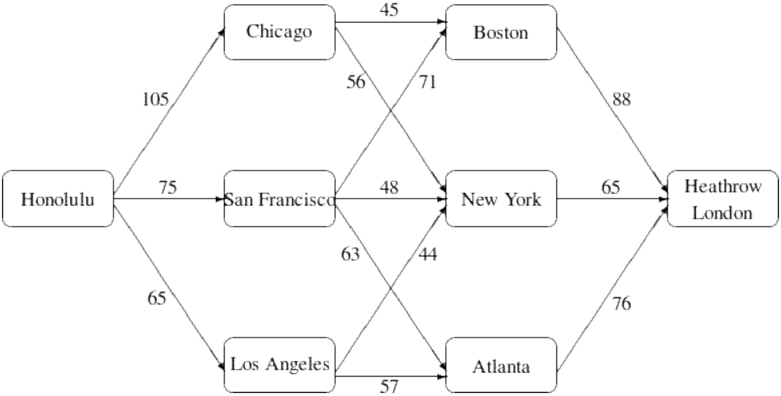
```
primary = grb.LinExpr()
secondary = grb.LinExpr()
// build up objective functions
m.setObjective(primary)
m.optimize()
objValue = model.objVal
m.addConstr(primary, ‘L’, (1 + EPS)*objValue, "PrimaryObj")
m.setObjective(secondary)
m.optimize()
```

Best Modeling Practices

- Allow objectives to become constraints and vice versa
- Multiple optima are the norm, not the exception
  - Exploit this with tie-breakers (secondary, tertiary objectives)
- Elasticize constraints

Shortest Path

- Let N be a set of cities.
- Let A be a set of arcs between cities.
- Let d_{ij} be the distance between city $i \in N$ and city $j \in N$.
- What is the shortest distance between a given origin city $s \in N$ and destination city $t \in N$?



Shortest Path

- Let π_j be the length of the shortest path from the origin to city j.
- Can write $\pi_j = \min_{(i,j)\in A} d_{ij} + \pi_i$ for every $j \in N$.
- Linearize, and maximize π_t to compute length of shortest path.
- Tight constraints indicate arcs on the shortest path.

$$
\begin{aligned}
\max_{\pi} \quad & \pi_t \\
\text{s.t.} \quad & \pi_j \le \pi_i + d_{ij}, \quad (i,j) \in A \\
& \pi_s = 0
\end{aligned}
$$

Shortest Path

Let $x_{ij} = 1$ if arc (i, j) is traversed.

$$\begin{aligned}
\min_{x} \quad & \sum_{(i,j)\in A} d_{ij} x_{ij} \\
\text{s.t.} \quad & \sum_{j\in RS(i)} x_{ji} - \sum_{j\in FS(i)} x_{ij} = b_i, \quad i \in N \\
& 0 \le x_{ij} \le 1, \quad (i, j) \in A,
\end{aligned}$$

where $FS(i) = \{j|(i, j) \in A\}$, $RS(i) = \{j|(j, i) \in A\}$, $b_s = -1$, $b_t = 1$, and $b_i = 0$ for $i \neq s, t$.

Exercise

Implement both shortest path formulations. How are the two formulations related?