Mathematical Programming

1 Introduction

Mathematical programming is an optimization tool that facilitates formalizing and solving complex decision-making problems. Mathematical programs comprise at least an objective function, and often a set of constraints as well. In other words, mathematical programs can be used to model **unconstrained** and **constrained optimization problems**. (As the names suggest, constrained optimization problems involve constraints, while unconstrained ones do not.)

When the objective function of a mathematical program is linear and the constraints are linear inequalities (or equalities), the program is called a **linear program**. Other forms of mathematical programs include **convex (or concave) programs**, where the objective function is convex (or concave) and likewise so are the constraints; **stochastic programs**, in which there is some uncertainty (i.e., randomness) at decision-making time; and **integer programs**, in which the variables are not necessarily continuous (e.g., in $\mathbb{R}$ or $[0, 1]$), but instead some or all of the variables are restricted to integer values.

Mathematical programming applications abound. One of the most interesting seminars Professor Greenwald remembers attending in the Brown CS department concerned an integer programming solution to the NFL (National Football League) scheduling problem. Some constraints were: all teams must play 17 games; all teams must play one game per week, except each team has one bye week (but there should not be too many byes in any one week); all teams must play roughly the same number of home and away games; some games should be broadcast on Monday night football; some games are played on Thanksgiving; stadiums must be available (some teams like the Jets and Giants share a stadium); regarding the pairings, there are certain rules about how many games are played by teams in the same division and how many by teams in different divisions, and there are certain rules about match-ups that depend on last year's standings; etc!

In today's lecture notes, we focus on modeling real-world problems (such as NFL scheduling) as mathematical programs. In practice, mathematical programming solvers like Gurobi and CPLEX are employed to solve these problems. Nonetheless, we spent some time earlier in this unit explaining what goes on inside these solvers "under the hood". Specifically, by now you should know how to

- solve unconstrained, specifically convex, optimization problems
- solve constrained optimization problems with linear constraints

2 Convex Sets

Recall the definition of convexity for functions: a function is convex iff the line segment connecting any two points on a function does not fall below the function. The notion of convexity for sets is similar.

A set $C \subseteq \mathbb{R}^n$ is **convex** iff for all $x, y \in C$ and $\lambda \in [0, 1]$, $\lambda x + (1 - \lambda)y \in C$. In other words, the line segment connecting any two points in the set lies entirely within the set.

[1] This notes were compiled in conjunction with with Professor Amy Greenwald.

Convex Set **Non-Convex Set**

Figure 1: In a convex set, all line segments connecting any two points in the set are fully contained within the set. In a non-convex set, there exists such a line segment that is not fully contained within the set.

Examples of Convex Sets

- Hyperplanes: $\{\boldsymbol{x} \in \mathbb{R}^d \mid \boldsymbol{a}^T\boldsymbol{x} = b\}$
- Half-spaces: $\{\boldsymbol{x} \in \mathbb{R}^d \mid \boldsymbol{a}^T\boldsymbol{x} \leq b\}$
- Polyhedra: intersections of a finite number of half-spaces
- Norm balls: $\{\boldsymbol{x} \in \mathbb{R}^d \mid \|\boldsymbol{x} - \boldsymbol{c}\| \leq r\}$, assuming a center $\boldsymbol{c} \in \mathbb{R}^d$ and a radius $r \in \mathbb{R}$

Operations that Preserve Convexity

- Intersection: $C_1 \cap C_2$ is convex if C_1 and C_2 are convex
- Cartesian product: $C_1 \times C_2$ is convex if C_1 and C_2 are convex
- Affine transformation: $\{A\boldsymbol{x} + \boldsymbol{b} : \boldsymbol{x} \in C\}$ is convex if C is convex

3 Hyperplanes, Half-Spaces, and Convex Polytopes

Recall that a line can be fully characterized by its normal. That is, the normal vector $\boldsymbol{n} = \langle a, b\rangle \in \mathbb{R}^2$ defines the **line** $ax + by + c = 0$, where $a, b, c \in \mathbb{R}$. Note that such a line is 1-dimensional, although it inhabits 2-dimensional space.

More generally, in 3-dimensional space, the normal vector $\boldsymbol{n} = \langle a, b, c\rangle \in \mathbb{R}^3$ defines the 2-dimensional **plane** $ax + by + cz + d = 0$, where $a, b, c, d \in \mathbb{R}$.

And more generally still, we can define a **hyperplane**, the generalization of lines and planes to d dimensions, in terms of its normal $\boldsymbol{n}$ and an offset b as all vectors $\boldsymbol{x}$ that satisfy $\boldsymbol{n} \cdot \boldsymbol{x} = \boldsymbol{n}^T\boldsymbol{x} = c$.

Given $\boldsymbol{w} \in \mathbb{R}^d$ and $b \in \mathbb{R}$, a hyperplane $\boldsymbol{w}^T\boldsymbol{x} = b$ defines two **half-spaces** as follows:

- $\boldsymbol{w}^T\boldsymbol{x} \leq b$, or equivalently, $\boldsymbol{w}^T\boldsymbol{x} - b \leq 0$
- $\boldsymbol{w}^T\boldsymbol{x} \geq b$, or equivalently, $\boldsymbol{w}^T\boldsymbol{x} - b \geq 0$

That is, a hyperplane is the boundary between the two half-spaces.

The intersection of a finite number of half-spaces is called a **polytope**.

- In 2D, the intersection of half spaces forms a **polygon**
- In 3d, it is called a **polyhedron**
- In n-dimensional space, it is called a **polytope**

Interestingly, all of these geometric objects are convex. Why? Because half-spaces are convex (**Exercise!**), and as we stated without proof earlier, the intersection of convex sets is again convex.

4 Mathematical Programming

The general form of a mathematical program is:

$$\begin{aligned} \min_{\boldsymbol{x}\in\mathbb{R}^n} \quad & f(\boldsymbol{x}) \\ \text{s.t.} \quad & g_i(\boldsymbol{x}) \leq 0, \quad i \in [m] \\ & h_j(\boldsymbol{x}) = 0, \quad j \in [p] \end{aligned}$$

where

- $\boldsymbol{x}$ is the vector of decision variables
- $f(\boldsymbol{x})$ is the objective function
- $g_i(\boldsymbol{x}) \leq 0$ are inequality constraints
- $h_j(\boldsymbol{x}) = 0$ are equality constraints

5 Linear Programming

Linear programming is the most fundamental class of mathematical programming.

Linear program (LP) The standard form of a linear program is:

$$\min_{\boldsymbol{x}\in\mathbb{R}^n} \quad \boldsymbol{c}^T\boldsymbol{x} \tag{1}$$

$$\text{s.t.} \quad A\boldsymbol{x} \leq \boldsymbol{b} \tag{2}$$

$$\boldsymbol{x} \geq 0 \tag{3}$$

where $\boldsymbol{c} \in \mathbb{R}^n$, $A \in \mathbb{R}^{m\times n}$, and $\boldsymbol{b} \in \mathbb{R}^m$.

Despite its simplicity, linear programming is surprisingly effective at representing a wide variety of problems. The best way to see this is to work through some examples!

6 Examples

The remainder of this lecture is concerned with designing mathematical programs. We provide several examples, based on which we hope the students can learn to formalize mathematical programs themselves.

6.1 Manufacturing

A manufacturing business is able to make n different goods with their manufacturing plant (e.g., shirts, pants, etc.). Each of these goods requires different amounts of raw materials (e.g., cotton, buttons, etc.). The firm only has access to certain amounts of each raw material. Each good can be sold for some amount of revenue (which varies with the good type). How can the firm maximize its revenue, while obeying its capacity constraints: i.e., not using more natural resources than are available?

What are the **decision variables**? The firm can decide how much of each product to produce. Let x_i be the quantity of good i that the firm produces. There are n total decision variables, since there are n goods that can be produced.

What is the **objective function**? Each good has an associated profit margin, which we will call c_i for good i. Note, this is not a decision variables. The profit per good is known ahead of time. The firm would like to maximize its profits, which is:

$$\max_x \qquad c_1x_1 + c_2x_2 + \cdots + c_nx_n$$

What are the **constraints**? There is a limited amount of raw materials available and each good requires a certain amount of each natural resource. We will call the amount of natural resource i available b_i. Let the amount of raw material i that good j requires be called $a_{i,j}$. The amount of resource i that is used total is:

$$a_{i,1}x_1 + a_{i,2}x_2 + \cdots + a_{i,n}x_n$$

This will be true for all raw resources available. The total amount used, must be less than the maximum available amount of each resource, which gives us the following set of constraints:

$$a_{i,1}x_1 + a_{i,2}x_2 + \cdots + a_{i,n}x_n \leq b_i \qquad \forall i = 1, \ldots, m$$

There is one final very important constraint that is often forgotten when formulating problems. The firm is not allowed to produce negative amounts of a good. You may think that there would be no reason to produce a negative amount of a good, since it would reduce our total objective (assuming all profits are positive). However, producing a negative product would in fact "produce" more natural resources that could be used to make other products. This is not how production works in the real world, so we should not allow this in our model.

$$x_i \geq 0 \qquad \forall i = 1, \ldots, n$$

Putting these all together, we have our fully formed linear program:

$$\begin{array}{lcl} \max\limits_x & c_1x_1+c_2x_2 + \cdots + c_nx_n & \\ \text{s.t.} & a_{i,1}x_1+a_{i,2}x_2 + \cdots + a_{i,n}x_n \leq b_i & \forall i = 1, \ldots, m \\ & x_j \leq 0 & \forall j = 1, \ldots, n \end{array}$$

This can be rewritten very compactly using vectors and matrices. Let $\vec{x} = (x_1, x_2, \ldots, x_n)$ be the vector of decision variables and $\vec{c} = (c_1, c_2, \ldots, c_n)$ be the vector of profits. The objective can be rewritten as the dot

product between these two vectors $\vec{c}^T\vec{x}$. Let A be a matrix with $a_{i,j}$ as the entry of A at row i column j and $\vec{b} = (b_1, b_2, \ldots, b_m)$. Our constraints can be rewritten as $A\vec{x} \leq b$. Our full LP is therefore:

$$\begin{aligned} \max_x \quad & \vec{c}^T\vec{x} \\ \text{s.t.} \quad & A\vec{x} \leq \vec{b} \\ & \vec{x} \geq 0 \end{aligned}$$

6.2 Nutrition

The human body needs certain nutrients to survive and there are suggested levels of daily intake for many of these nutrients (e.g., 18mg of iron, 28g fiber, 50g of protein). Different foods will supply each of these nutrients in different amounts. Chicken is high in protein, but low in fiber. Additionally, each of the different foods you might eat has an associated cost. What is the least-cost way to achieve your nutritional goals for a given set of n foods with known nutritional values and cost?

What are the **Decision Variables**? You can decide how much of each food you buy and eat. Let x_i be the amount of food i to purchase. There are n foods in total, so there will be n different decision variables.

What is the **objective function**? You know the cost of each food (or can look it up). Let c_i be the cost of food i. The objective cost is therefore:

$$\min_x \quad c_1x_1 + c_2x_2 + \cdots + c_nx_n$$

What are the **constraints**? We must meet our dietary needs. That is, for each nutritional requirement b_j, we need to eat more than b_j. Let $a_{j,i}$ be the amount of nutrient j provided by food i. If we know how much food we will eat is x_i, then we will get $a_{j,i}x_i$ of that nutrient by eating food i. The constraints can therefore be written:

$$a_{j,1}x_1 + a_{j,2}x_2 + \cdots + a_{j,n}x_n \geq b_j \quad \forall j$$

Finally, we can put it all together:

$$\begin{aligned} \min_x \quad & c_1x_1 + c_2x_2 + \cdots + c_nx_n \\ & a_{j,1}x_1 + a_{j,2}x_2 + \cdots + a_{j,n}x_n \geq b_j \quad \forall j \\ & x_i \geq 0 \quad \forall i \end{aligned}$$

And again, we can rewrite this with matrices and vectors as:

$$\begin{aligned} \min_x \quad & \vec{c}^T\vec{x} \\ & A\vec{x} \geq \vec{b} \\ & \vec{x} \geq 0 \quad \forall i \end{aligned}$$

6.3 Scheduling

A hospital is trying to schedule nurses for the night shift (12am-8am). The nurses are in a union which guarantees them 2 consecutive days off each week (they all work 5 straight days, then 2 days off). The hospital knows how many nurses it needs for each day of the week. What is the minimum number of nurses the hospital can hire while satisfying the demand for each day of the week?

Here, the choice of decision variable is not obvious. The hospital is deciding how many nurses to hire, but that single variable will not help us determine if we can meet the demand. What about deciding how many nurses work each night of the week? The constraints will follow that the number of nurses working each night must be greater than or equal to the demand. But then how to encode the requirement that nurses must work 5 days in a row if we don't know when each nurse starts? We can use x_i to be the number of nurses that *start* their week on day i. Let b_i be the number of nurses required for day i. We can formulate the problem as follows:

$$
\begin{aligned}
\min_x \quad & \sum_{i=1}^{7} x_i \\
\text{s.t.} \quad & x_1 + && x_4 + && x_5 + x_6 + && x_7 = b_1 \\
& x_1 + x_2 + && && x_5 + x_6 + && x_7 = b_2 \\
& x_1 + x_2 + && x_3 + && x_6 + && x_7 = b_3 \\
& x_1 + x_2 + && x_3 + x_4 + && && x_7 = b_4 \\
& x_1 + x_2 + && x_3 + x_4 + && x_5 && = b_5 \\
& x_2 + && x_3 + x_4 + && x_5 + x_6 && = b_6 \\
& && x_3 + x_4 + && x_5 + x_6 + && x_7 = b_7 \\
& && \vec{x} \geq 0
\end{aligned}
$$

Can we rewrite this in matrix form? We don't have any obvious a_i in this problem, but the constants are still there. We could rewrite the first constraint as:

$$1x_1 + 0x_2 + 0x_3 + 1x_4 + 1x_5 + 1x_6 + 1x_7 = b_1$$

Doing so it becomes clear that A will be a binary matrix (made up of 0's and 1's) that match the constraints described above.

$$
\begin{aligned}
\min_x \quad & \sum_{i=1}^{7} x_i \\
\text{s.t.} \quad & A\vec{x} = \vec{b} \\
& \vec{x} \geq 0
\end{aligned}
$$

What would it mean if the final output of our model said that $x_1 = 4.5$ and $x_2 = 5.5$? How can we hire half a nurse (the union won't allow part time)? We could add a constraint to the problem that $x_i \in \mathbb{Z} \quad \forall i = 1, \ldots, 7$. This integer constraint would turn the problem into an **Integer Linear Program** (ILP). ILPs are NP-Hard to solve in general, but often *relaxing* the problem to an LP by dropping the integer constraint will yield good solutions, which can be turned into feasible solutions by rounding each fractional x_i.

6.4 Optimal Control

How can we plan a path for a robot to move from one location to another? We've previously used A* and search algorithms to plan discrete paths. If we wanted to solve this, we might consider discretizing our space (e.g., making it into a grid and creating a graph from that grid). However, this assumes that the robot can move in any direction at any time, which isn't necessarily true. For instance, cars can't move directly left or right (if they could, parallel parking would be trivial). How then, can we plan paths for robots with more complex motion constraints?

You are in charge of landing a rocket on the moon (a very tricky task). Other engineers have guaranteed that they can keep the rocket above their landing site, but they need help controlling the height of the rocket and bringing it in for a smooth landing (i.e., you only have to worry about the z-direction). The moon has an acceleration due to gravity of $-1.625m/s^2$. The rocket has a maximum acceleration of $5m/s^2$. You will take control of the rocket at a height of 1,000m. Finally, the total amount of acceleration of a rocket is correlated with the amount of fuel you need to bring. If you can land the rocket with less fuel, you'll save money on transport costs. To land, means to bring the rocket to a halt (velocity=0) at ground level (height=0). Your supervisor has informed you that they would like you to land the rocket after exactly 2 minutes of descending. What is the minimum amount of acceleration required to land the rocket?

First, what are the **decision variables**: It should be clear that acceleration will be a decision variable (i.e., when to apply thrust). However, one variable for acceleration is not enough. We will need to constantly make decisions about whether to apply thrust or not. For this problem, we will apply thrust every Δt seconds (e.g., $\Delta t = 0.1$ seconds). At every timestep t, we will decide how much acceleration a_t to deploy. But we still need to connect acceleration and our decision variable to the rest of the problem. The goal of our rocket is to land at the ground, which involves height (z) and velocity (v). These are values that are changing with every timestep t, and are therefore not constant/data values, but variables. They might not look like decision variables in the sense that we do not have direct control of them, but by adding constraints, we will directly tie them to the decisions that we make. We will need a height and velocity variable for every time step, which we will denote z_t and v_t respectively.

What is the **cost function**: This is relatively straightforward. We'd like to minimize the total acceleration, which can be written:

$$\min_a \sum_{i=1}^{n} a_t$$

Where n is the total number of timesteps (i.e., 120 seconds / Δt).

What are the **constraints**: First, we will write out some implicit constraints in in the problem:

$$a_t \geq 0 \quad \forall t$$
$$z_t \geq 0 \quad \forall t$$
$$a_t \leq 5$$

Acceleration should be non-negative (our rocket is always pointing up and can't accelerate towards the moon) and cannot exceed $5m/s^2$. Height should be non-negative (why would a negative height be concerning?).

Then, we will connect velocity, acceleration, and height together:

$$v_t = v_{t-1} + \Delta t \cdot a_t - \Delta t \cdot 1.625$$
$$z_t = z_{t-1} + \Delta t \cdot v_t$$

Velocity is equal to the previous velocity, plus the current acceleration due to thrust and moon gravity. Height is the current height plus the velocity times the time that has passed. Note: Δt is a constant that is chosen while modelling the problem. Higher resolution (lower Δt) will give more precise solutions at the cost of how long the model takes to solve.

Finally, we have to input our initial and ending conditions:

$$z_0 = 1,000$$
$$z_n = 0$$
$$v_n = 0$$

Finally, if you think this is a bit contrived and perhaps you could come up with a better way to solve this using calculus or physics, consider a very similar problem of planning a trajectory for a quadcopter, which has much more complicated dynamics. You have to plan thrust values for 4 different motors, each of which has a different rotational effect on the drone. Additionally, we no longer prefer to minimize acceleration but *snap* (the 4th derivative of position) as it is closely tied to energy consumption for quadcopters. That's where minimum-snap trajectories come in, which a technique to meet the dynamics constraints of quadcopters, while minimmizing snap.