CHEMICAL ENGINEERING 4G03

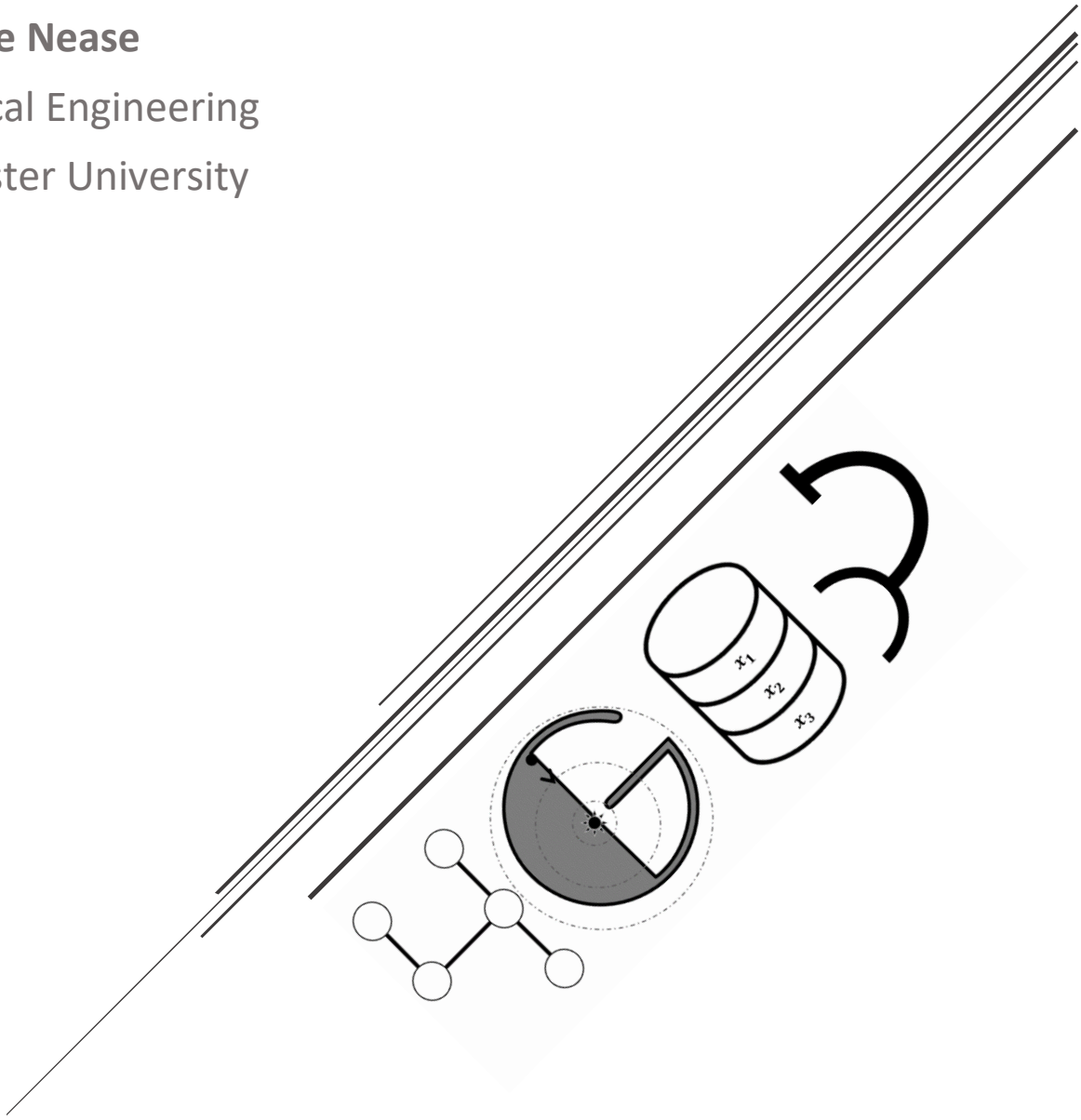
Module 03

Linear Programming (I)

Dr. Jake Nease

Chemical Engineering

McMaster University



Updated January 30, 2023

Outline of Module

This module consists of the following topics. It is important to note that this topic will consist of multiple modules.

Suggested Readings

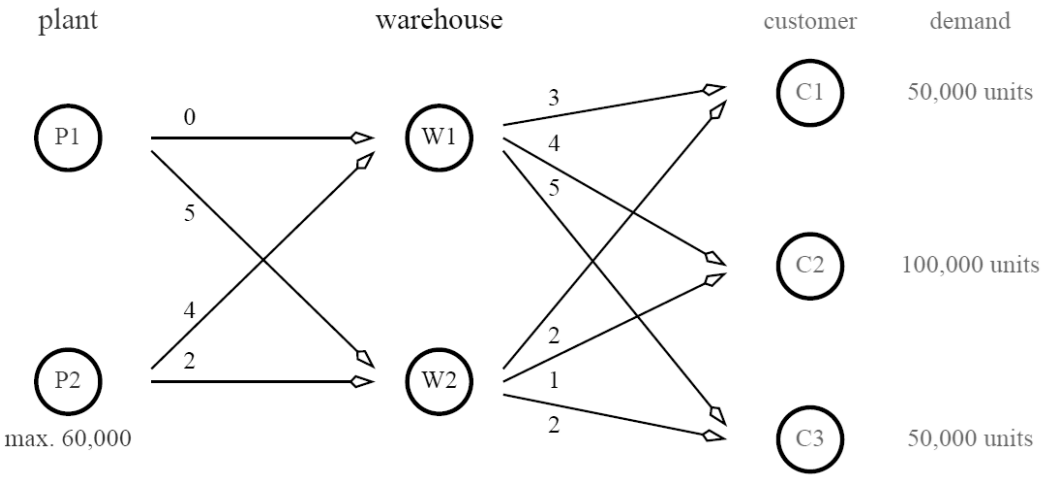
Rardin (1st edition): Chapter 5.1-5.2

Rardin (2nd edition): Chapter 5.1-5.2

Linear Programming: Perspectives

Linear programs are by no means an "introductory" method that sets the table for more complicated scenarios. Although complicated optimization problems *do* exist and can be very applicable, the fact is that linear programming is the **most frequently** used method in the realm of optimization. Linear programs can be very large and impossible to handle by hand, but have huge value in their solutions. As a matter of fact, the first step toward solving a nonlinear program is to attempt to **reformulate** it as a linear program (with appropriate relaxations and assumptions!). Moreover, the first stage in the solution of truly nonlinear programs is to solve a **linear relaxation** as a starting point… So really, linear programs are absolutely everywhere!

For example, consider the following figure, which shows a simple distribution network of two manufacturing plants supplying two warehouses, with each warehouse attempting to fulfill three customer demands (all for the same product; Figure courtesy of Dr. Chachuat). The manufacturing costs for each plant are the same, and the numbers indicated on the arrows represent the shipping costs of going from one location to the other. Our goal is to choose how much of each plant to produce to satisfy demand with the lowest possible shipping costs. Not so easy, is it? Well it is with Linear Programming!



Learning Goals

In this section, we have a few learning goals to lay out, each of which are *just* as important as the others. We are going to tackle these learning goals be attempting to understand the **geometric** interpretation and necessary **matrix calculations** to form a comprehensive knowledge base.

- Attitudes
  - An *optimum* is better than an *answer*.
  - Numbers with no perspective or understanding are useless ("It says $x_3 = 50$").
- Skills
  - Formulation of real-world problems into mathematical systems of equations.
  - Solving optimization problems.
  - Communicating the results to other engineers and those without technical backgrounds.
- Knowledge
  - When and how we can formulate optimization models.
  - Know when we are dealing with "weird events" that can occur in optimization (analysis).

Definitions of Linear Programming

General Formulation of a Linear Program

Linear Program

An optimization model is known as a **Linear Program** if it has the following features:

1. It is comprised of *only continuous variables*
2. It has a *single linear objective function*
3. It has *only linear equality and/or inequality constraints*

A linear program has the following general formulation notation:

$$\min_{\boldsymbol{x}} \phi = \boldsymbol{c}^T\boldsymbol{x} \quad \leftarrow \text{Objective Function}$$

$$s.t. \quad \leftarrow \text{“Subject to”}$$

$$A_h\boldsymbol{x} = \boldsymbol{b}_h \quad \leftarrow \text{Equality Constraints}$$

$$A_g\boldsymbol{x} \le \boldsymbol{b}_g \quad \leftarrow \text{Inequality Constraints}$$

$$\boldsymbol{x}_{lb} \le \boldsymbol{x} \le \boldsymbol{x}_{ub} \quad \leftarrow \text{Variable Bounds}$$

You should instantly notice that this is different from our “general” formulation that we came up with in Module 01. In this case, the objective function $f(\boldsymbol{x})$ has been replaced by $\boldsymbol{c}^T\boldsymbol{x}$ because we are **specifying** that the objective function is in fact a linear combination of the decision variables $\boldsymbol{x}$, or (ALL of these are equivalent… Think why!):

$$\phi = \boldsymbol{c}^T\boldsymbol{x} = \underbrace{[c_1 \;\; c_2 \ldots \;\; c_n]\begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{bmatrix}}_{One\ notation} = \underbrace{c_1x_1 + c_2x_2 + \cdots + c_nx_n}_{Another\ notation} = \underbrace{\sum_{i=1}^{n} x_ic_i}_{ANOTHER\ldots}$$

For the same reason, we have specified the structure of the constraints as $A_h\boldsymbol{x} = \boldsymbol{b}_h$ and $A_g\boldsymbol{x} \le \boldsymbol{b}_g$ because we no longer need to constrain ourselves to the general forms $\boldsymbol{h}(\boldsymbol{x})$ and $\boldsymbol{g}(\boldsymbol{x})$. Our constraints are linear combinations as well! In summary:

- $x_j \rightarrow j^{th}$ decision variable.
- $c_j \rightarrow j^{th}$ cost coefficient for the j^{th} decision variable.
- $a_{i,j} \rightarrow$ constraint coefficient (think of systems of equations) for variable j in constraint i.
- $b_i \rightarrow$ RHS coefficient for constraint i (we no longer need RHS = 0 since we are linear!).
- A_h and A_g both have n columns (number of variables) and m_h and m_g rows, respectively (number of constraints of each type)

Interior, Boundary, and Extreme Points

If you recall from the previous section, any solution (not to be confused with optimum) of an optimization program must exist within (or at the boundary of) the feasible set. We are able to extend this discussion into a more specific scenario with linear programs in order to define the different classes of feasible solution points.

Point Classifications in a Linear Program

A **feasible solution** (NOT necessarily an optimum) to a linear program is classified as:

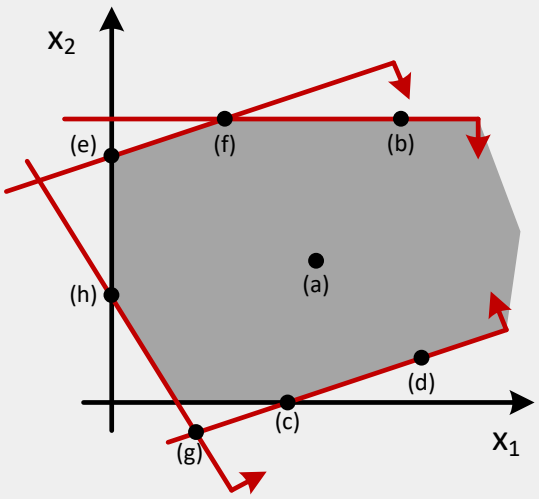
- A **boundary point** if at least one inequality constraint is satisfied as an *equality* (is active) at the given point (for example, the octane number for gasoline blending is 90.5 exactly).
- An **interior point** if no inequalities are "active" (for example, the octane number for gasoline blending is 91.6).
- An **extreme point** (or a **corner point**) if every line segment in the feasible region containing it *also has that point as an endpoint* (for example, the octane number, RVP and volatility for the gasoline blending problem are 90.5, 12.8 and 52%, respectively)

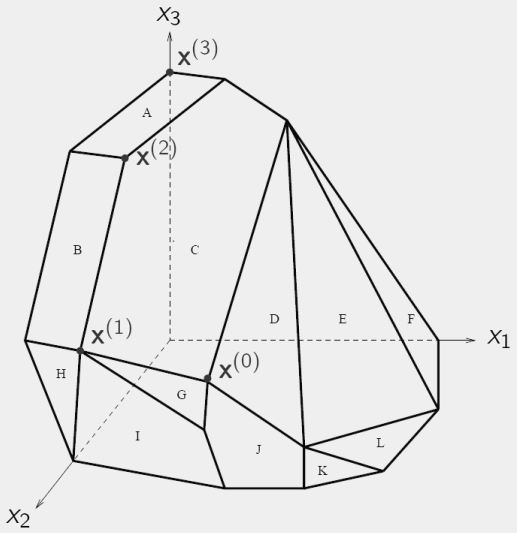
When considering extreme points, remember the following **important remarks**:

- Extreme points are called so because they are "jagged" extrema and thus "stick out" of the feasible region (they are pointy/cornered due to the convexity of linear constraints).
- Every extreme point is determined by the intersection of **at least n linearly independent constraints** that are active ONLY at that solution. For example, in three dimensions ($n = 3$), the intersection of three (or possibly more!) constraints define each extreme point.

Class Workshop – Classifying Feasible/Extreme Points

1. Determine if each of the points in the figure below to the left represent interior, boundary, and/or extreme points for the LP feasible region.
2. Identify all sets of 3 constraints that determine the extreme points in the figure below to the right.





Workshop Solution – Classifying Feasible/Extreme Points

(a)	(e)	$\boldsymbol{x}^{(0)}$
(b)	(f)	$\boldsymbol{x}^{(1)}$
(c)	(g)	$\boldsymbol{x}^{(2)}$
(d)	(h)	$\boldsymbol{x}^{(3)}$

The classification of boundary points versus interior points might seem rather arbitrary at first, but it actually leads us to a **powerful and useful set of results**.

Optimal Points in Linear Programs

Fundamental Result 1

EVERY solution to an LP with a non-constant objective function will be at a *boundary point* of its feasible region.

Fundamental Result 2

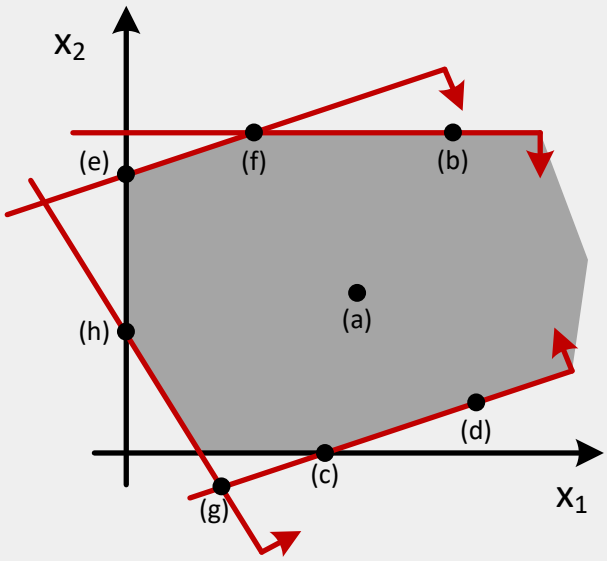
If an LP has a unique optimum, it *MUST* occur at an *EXTREME POINT* of the feasible region. Moreover, if an LP has multiple (equivalent) optima, *one of them MUST* occur at an extreme point of the feasible region.

Fundamental Result 3

EVERY local optimum for an LP is also a global optimum because *ALL LPs are CONVEX*

Class Workshop – Identifying Optimal Points

For the following linear program, identify which of the labeled points are potential optima, potential unique optima, or neither:



Workshop Solution – Classifying Feasible/Extreme Points

(a)	(d)	(g)
(b)	(e)	(h)
(c)	(f)	

The final piece of the puzzle that we require before being able to solve these types of problems are the definitions of **edges** and **adjacent points**. Once we get this out of the way, we will be able to derive a solution method that exploits the geometric nature of all linear programs. So… Why wait! They are defined right here:

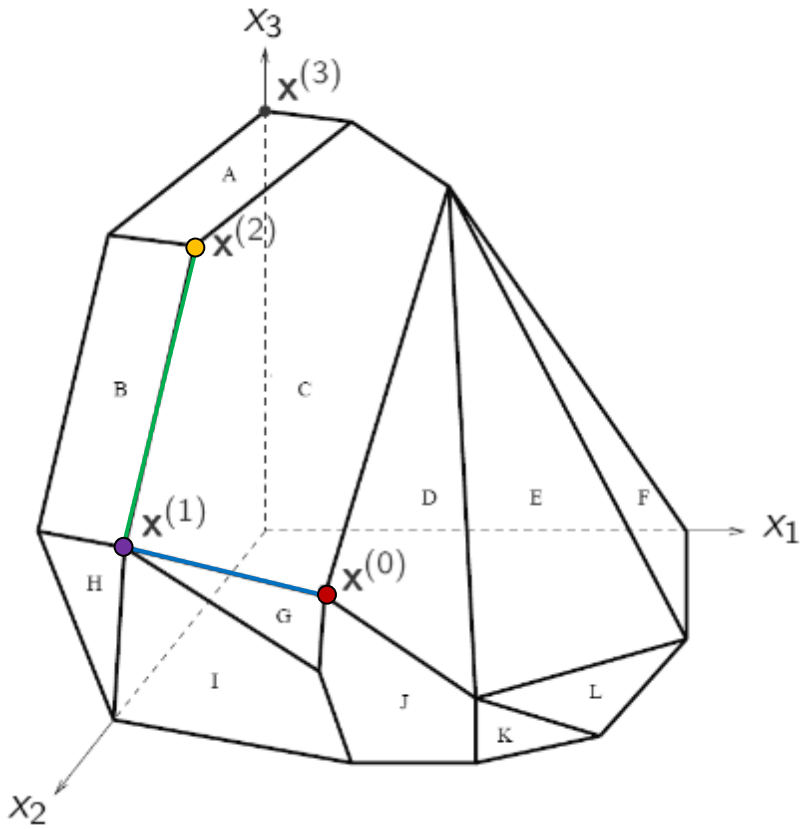
Adjacent Extreme Points and Edges

Two extreme points of an LP feasible region are **adjacent** if they are determined by an active set of constraints that differ by only one element (variable). An **edge** of the feasible region is a 1-D set of feasible points along a line determined by the collection of active constraints.

- Every edge is determined as the intersection of **exactly $n-1$ independent, active constraints**.
- Adjacent extreme points are **joined** by the edge that has all of the active points those extreme points have in common (think why?).

The geometric interpretation of active versus extreme points can be visually interpreted by looking at the faceted 3-D feasible region in one of our previous workshops. In the figure below, we have the following scenario:

- There are $n = 3$ variables (x_1, x_2, x_3)
- $\boldsymbol{x}^{(0)}$ and $\boldsymbol{x}^{(1)}$ are **adjacent extreme points**. They are connected by the **edge** defined by constraints G and C. Also note that G and C are $(n-1) = 2$ constraints, as required to define an edge.
- $\boldsymbol{x}^{(1)}$ and $\boldsymbol{x}^{(2)}$ are **adjacent extreme points**. They are connected by the **edge** defined by constraints C and B. Also note that C and B are $(n-1) = 2$ constraints, as required to define an edge.



Congratulations! You now have all the terminology and tools you need to develop an algorithm to solve linear programs! So why don't we just go right on ahead and do that!

Introduction to Linear Solution Algorithms

Brute Force Method

OK, so we have a very convenient interpretation of linear programs. We know that we can find the solution to **any LP** as one of its extreme points, AND we know that the extreme points are classified as the intersection of at least n linearly independent constraints. OK, so how about this idea:

LP Algorithm Idea (1)
The optimum to any LP can be found by determining the set of *all* extreme points, evaluating the objective function at each of them, and keeping the extreme point with the **best objective value**.

This is actually not a terrible idea! However, for the sake of argument let's list some **advantages** and **disadvantages** of this method:

- **AD –** Guaranteed to find the global optimum.
- **AD** – Simple to implement.
- **DISAD** – Not very elegant (if you're into that sort of thing).
- **DISAD** – Might have to visit *A LOT* of different points to make sure we have the answer.

Now, when I say we have to visit *a lot* of points. How many is that, exactly? Recall that not every constraint combination ends up giving us an extreme point, since a good deal of them intersect outside of our feasible region. However, when considering a solution algorithm we must always plan for the worst-case scenario. As it turns out, the **maximum number of extreme points** $\mathcal{N}_E$ for a given LP with n variables and m independent constraints is defined as:

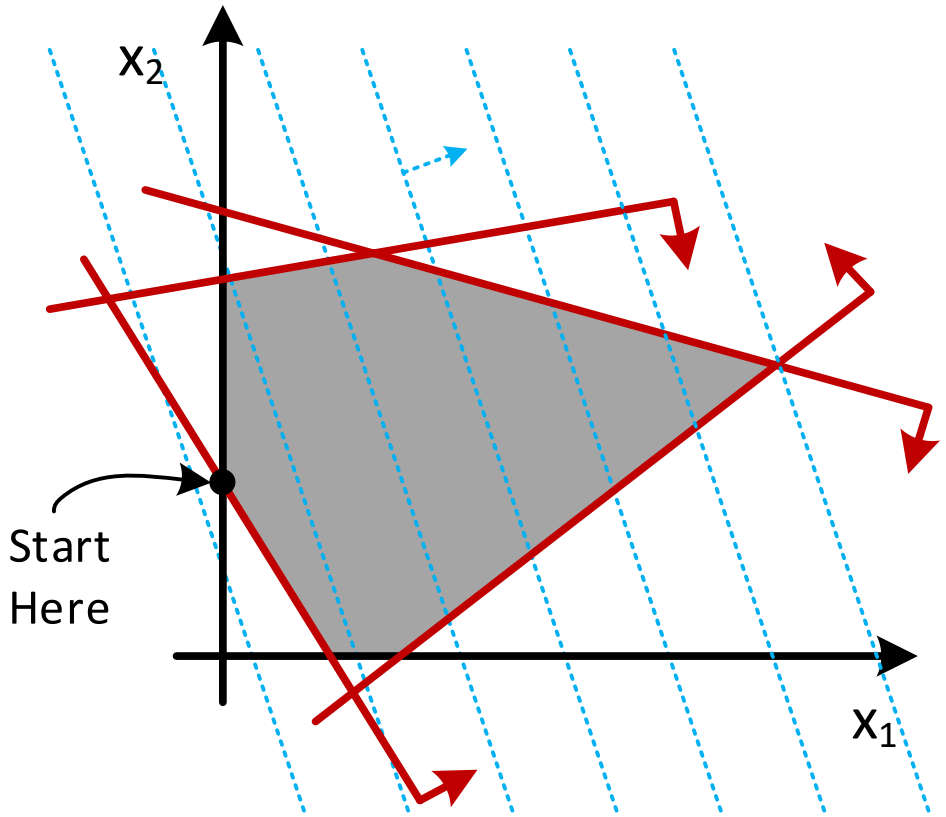
$$\mathcal{N}_E = \frac{n!}{(n-m)!\,m!}$$

So, if we consider a rather **small** problem of $n = 20$ variables and $m = 10$ independent constraints, we have a grand total of $\mathcal{N}_E = 184{,}756$ extreme points to worry about. Depending on the computation cost of a single extreme point, this could be done either very quickly or take quite some time! And, of course, as we extend beyond 20 variables and 10 constraints, things escalate very, very quickly. We can do **better**.

A Better Method

Consider the situation where we have a linear program like the one shown in the figure below. It is our task to start at the starting point and find the optimum (which we can see in this example… Where is it?) as efficiently as possible.

Class Workshop – Developing an LP Algorithm
Consider the linear program given in the figure below. Develop an algorithm that combines basic numerical methods with our new **geometric insight** for LPs that will find the optimum. Be sure to generalize your approach so that it extends to higher dimensional problems that we can't visualize. Then, draw/highlight how you would *expect* your algorithm to proceed for the given 2-D problem.



Workshop Solution – Developing an LP Algorithm

The algorithm that **you** just came up with is the basis for what is known as the ***Simplex Search***. It is my favourite numerical method of all time because of how incredibly simple and elegant it is. Hopefully you will feel the same about is (less than three, hopefully) as I do once we figure it all out.

OK, so we know what we *want* to do, but now we face another issue: **HOW** do we actually only run along the edges to extreme points? How do we know what constraint path to run along if the problem cannot be visualized? Luckily for us, there is a very good method for this, but it requires that we put the problem in Standard Form.

Linear Programs in Standard Form

Standard form will allow us to use matrices (yayyy!) as a part of our Simplex algorithm. Linear Programs in standard form have the following characteristics:

- Only *equality* constraints (except for non-negativity).
- Only *positive* variables.
- The objective function and all constraints have like-terms collected so that each variable only appears on the LHS of any equation *once* and a single constant term (possibly zero) on the RHS.

Standard Linear Program

An optimization model is known as a **Standard Linear Program** if it obeys:

$$\min_{\boldsymbol{x}} \phi = \boldsymbol{c}^T \boldsymbol{x} \quad \leftarrow \text{Objective Function}$$

$$s.t. \quad \leftarrow \text{"Subject to"}$$

$$A\boldsymbol{x} = \boldsymbol{b} \quad \leftarrow \text{ONLY Equality Constraints}$$

$$\boldsymbol{x} \geq 0 \quad \leftarrow \text{Variable Bounds}$$

We want to convert our system to standard form because **systems of linear equations (even under-defined systems) are easy to work with**. Moreover, *all linear programs can be converted to standard form*. We can convert to standard form by performing the following:

- Convert any ≤ and ≥ inequalities to equalities.
- Convert all non-positive variables to non-negative.
- Convert any unrestricted (URS) variables to non-negative.

Converting Inequalities to Equalities

A **less-than inequality** (≤) may be converted to an equality by the addition of a *slack variable*. A **slack variable (s^-)** is an **invented** variable that is non-negative. It does not affect the actual solution to the problem:

$$a_1 x_1 + \cdots + a_n x_n \leq b$$

$$a_1 x_1 + \cdots + a_n x_n + s^- = b$$

In this case, any combination of $x_1 \ldots x_n$ that would have given us a value less than b in the above constraint can have the variable s^- ADDED to it so that it now *equals* b. A greater-than inequality is handled in the same way, except now we subtract a **surplus variable (s^+)**:

$$a_1 x_1 + \cdots + a_n x_n \geq b$$

$$a_1 x_1 + \cdots + a_n x_n - s^+ = b$$

You can see that if we would have had a combination of variables resulting in a number greater than b, subtracting the (non-negative) surplus will give us exactly b. NOW… Ask yourself: Why can we be sure that each inequality we change to an equality does not run the risk of giving us an over-specified system?

Converting Non-Positive and URS Variables to Non-Negative

Perhaps even simpler than converting inequalities to equalities, we can convert non-positive variables (say – conversion of a reactant in a chemical reaction) to positive ones **via a simple substitution**:

$$x \leq 0 \Rightarrow y = -x$$

$$\therefore y \geq 0$$

There are actually two ways of ridding ourselves of unrestricted (URS) variables. The first is to perform a substitution in which we *eliminate* the unrestricted variable by re-writing it in terms of the others. That is not very much fun. Especially if the system of equations is quite large. Let's not do that. INSTEAD, we can replace an URS variable with **the difference between two fictitious positive variables**:

$$x \in \mathbb{R} \Rightarrow x = y_1 - y_2$$

$$y_1, y_2 \geq 0$$

Now if I wanted x in the equation above to be negative, it corresponds to a solution where $y_2 > y_1$, and vice-versa. Now, you may be asking yourself a couple of questions, like:

1. There are now infinitely many combinations of y_1 and y_2 that would give the same x. How do we know we are choosing the right ones?
2. Does the addition of slack, surplus and substitution variables have any impact on our solution?

The answers for which are:

1. This is true but remember that the *only* real variable here is x, which means that even if the other variables are random numbers, it is the combination of those numbers that I truly care about!
2. It has a*bsolutely no impact* on the solution to the problem, but any variables changed by substitution will have to be re-computed at the end to ensure that they may be interpreted correctly. For example, a solution that reports $y_1 = 4$ and $y_2 = 6$ is *actually* saying that $x = y_1 - y_2 = -2$.

Class Workshop – Converting Problems to Standard Form

For each of the following linear programs:

1. Place them in standard form
2. Identify the number of variables (n), number of constraints (m), and the matrices $A, \boldsymbol{b}, \boldsymbol{c}$.

Problem 1

$$\begin{aligned} \min_{x} \phi &= 3x_1 + 6x_2 \\ \text{s.t.} & \\ x_1 + x_2 &\geq 4 \\ 6x_1 - 3x_2 - 25 &= 0 \\ x_2 + 8x_1 &\geq 15 \\ x_2 + 8x_1 &\leq 100 \\ x_1 &\geq 0 \\ x_2 &\leq 0 \end{aligned}$$

Problem 2

$$\begin{aligned} \max_{x} \phi &= 6(x_1 + 2x_2) - 8x_3 \\ \text{s.t.} & \\ 2(10 - x_1) + x_2 &\geq 3 - 5(9 - x_3) \\ 2x_1 + x_3 - 7 &\leq 5 \\ x_1, x_2 &\geq 0 \\ x_3 &= URS \end{aligned}$$

Workshop Solution – Converting Problems to Standard Form

Conclusions

We now know the name and general routine of the method used (almost exclusively!) to solve linear programs. Moreover, we have identified the procedure behind converting a problem into standard form, which will be essential in order to implement the Simplex Search. SO! We can identify, formulate, and set up literally any linear programming optimization problem at this point. What remains now? Well, only two things! We have to **solve it**, which will be covered (the algorithm, graphically and numerically) in the next section, and we have to **interpret our results**, which will be covered in the section after that. Once that is all said and done, AND once we have had some GAMS practice in tutorials, you will be able to confidently handle any LP.

~~ END OF MODULE ~~

\|T|/