LP1: Linear Programming Introduction

And

LP2: Geometry

Notes for CS-8803-GA: Introduction to Graduate Algorithms

Georgia Tech (Dr. Eric Vigoda), Fall 2017

as recorded by Brent Wagenseller

Overview

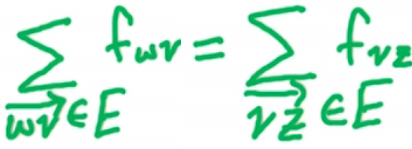
* Linear Programming is a general technique for solving optimization problems
  + Linear Programming can handle a problem that can be formulated as an optimization over a set of variables with a goal (known as the **objective function**) and the constraints can be expressed as linear functions of the variables
  + BRENTS NOTE: I believe linear programming can be summed up as given an equation to optimize and bunch of inequality equations (which set restrictions), find the global maximum point that maximizes the equation.
* We will
  + See Examples
    - Max Flow via LP
    - Simple production
  + See a general formulation
    - This is a standard form for Linear Programs
  + See Simplex algorithm, which can solve LPs
  + Examine LP Duality
  + Apply LP to a MAX-SAT approximation

Max-Flow via Linear Programming

* Recall Max-Flow
  + Input: directed graph G=(V, E) with capacities Ce > 0 for e ∈ E
* Our LP will have
  + m variables (one for every edge); specifically fe for every e ∈ E
  + objective function
    - We are trying to optimize this
    - We are maximizing the flow out of the source vertex:



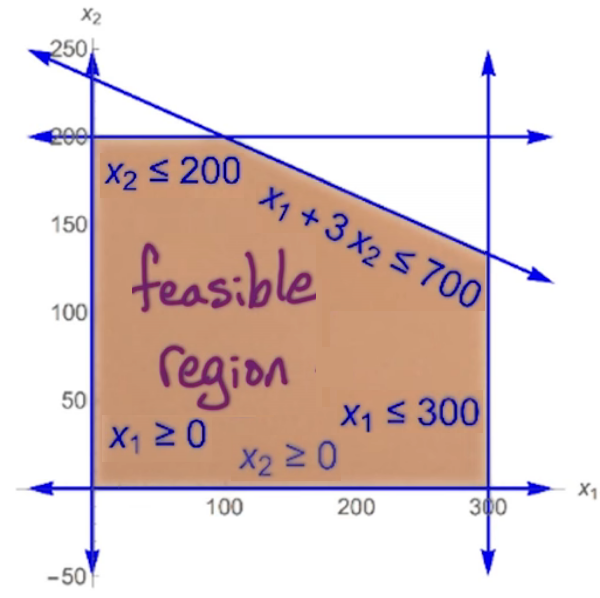
* + - * This is summing over the edges out of source S
    - Subject to:
      * For every e ∈ E, 0 <= fe <= ce
        + The flow can be at most the capacity
      * For every v ∈ V-{s,t}
        + The flow into the vertex must equal the flow out of that vertex



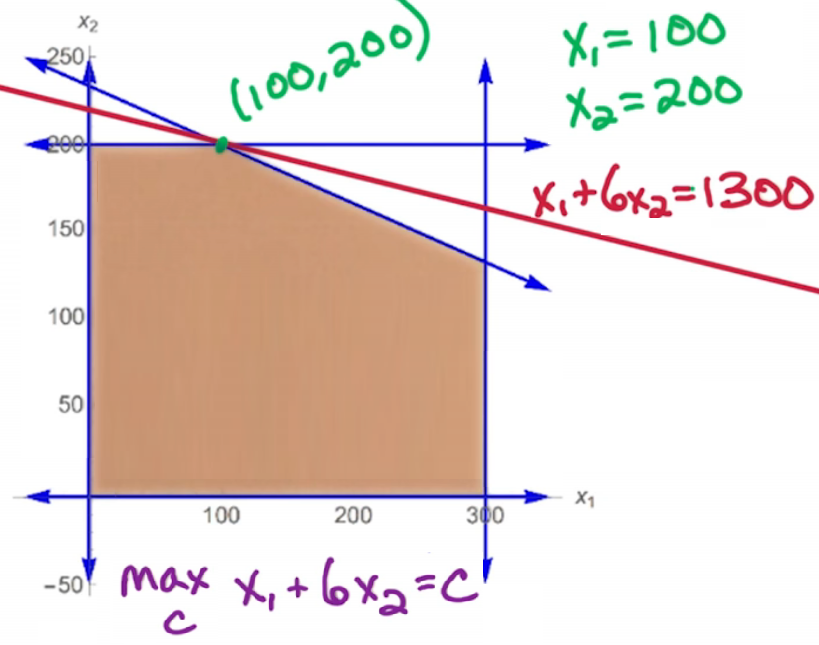
* + - * + The flow is conserved
  + This is a linear program for Max-Flow because the objective function is a max (or a min, but in this case max) of a linear function of the variables; in addition, all of the constraints are linear functions of the variables
    - We will easily be able to solve many variants of the Max-Flow problem with simple modifications to this LP

Simple 2D Example of LP

* Based on a basic production problem
* Setup
  + Consider a company that makes A&B products
    - Each unit of A makes $1 profit
    - Each unit of B makes $6 profit
  + Goal: Determine how many of each are needed to be produced to maximize profit
  + Constraints
    - Demand
      * <= 300 units of A a day
      * <= 200 units of B a day
    - Supply
      * Workers can work <= 700 hours per day
      * Each unit of A takes 1 hour
      * Each unit of B takes 3 hours
* Formulating as a Linear Program
  + Variables we will use in our LP
    - Since we are trying to determine the number of units of A and B we will be making, use
      * x1 = number of units of A to produce/day
      * x2 = number of units of B to produce/day
    - Constraints – Based off the constraints above
      * Demand
        + 0 <= x1 <= 300
        + 0 <= x2 <= 200
      * Supply
        + x1 + 3 x2 <= 700
  + Our goal is a maximization problem; so form this in terms of A and B
    - Max (x1 + 6 x2)
      * This is what we will be maximizing
  + Re-starting formulation
    - Ware are maximizing: x1 + 6x2
    - Subject to constraints
      * x1 <=300
      * x2 <= 200
      * x1 >= 0
      * x2 >= 0
      * x1 + 3x2 <= 700
    - Looking at this geometrically, each of the 5 constraints is a half-plane
      * If it was equality it would be a line, but we are doing one side of the line
      * We are in two dimensions because we are solving for 2 variables (A and B, or x1 and x2)
      * We want to satisfy all 5 constraints so we must look at the intersection of all 5 half-planes
      * The intersection of these 5 half-planes will give us the set of feasible x, which is a set of x which will satisfy all constraints
      * We will look in the feasible set to find the x which will maximize the objective function
* Geometric View



* + The above are all the half-planes that define the **feasible region**
    - all the points in this region satisfy all 5 constraints
    - We are looking to maximize this
* Finding the Optimal
  + The goal is to maximize (x1 + 6 x2)
  + We can do this by overlaying the **linear function** line **x1 + 6 x2 = c** over the feasible region
    - Specifically, we are interested where this line just touches the topmost point:



* + - This line intersects the last point that is in the feasible region
      * This happens to be x1 = 100 and x2 = 200
      * For these values, c = 1300

Key Issues

* We are optimizing over the entire region, so the optimum may be non-integer
  + To handle this, we usually round
* LP ∈ P
  + Linear Programming optimizes over the entire set
  + When we optimize over the set, the problem lies within polynomial time
* ILP is NP-Complete
  + ILP is is ‘best **Integer Linear Programming**’ when we look for the best integer values
  + BRENTS NOTE: I believe this is finding the best integers of x1 and x2, but we will see this later in the lecture
* The optimal point lies within a vertex (aka corner) of the polygon
  + Vertex not to be confused with vertices in graphs
  + There are 5 potential vertices in the example above (although only one is correct)
  + There may be other points that are also optimal that are not a vertex, BUT its always guaranteed that a vertex will be at least as good as any other point
  + Since the optimal line is a line, its possible multiple correct answers will exist on the line, but we are guaranteed a corner to be one of them
* Feasible region is convex
  + **Convexity** means that if we pick any two points in the feasible region and draw a line between the points, the entire line/edge is contained in the set
    - The convexity is due to the fact that we are using half-planes to create the boundaries of the feasible region
    - A key point of convexity is its always optimal at a vertex
      * Its possible other points are optimal, but we are guaranteed that one optimal point is a vertex
      * The concept that the optimal vertex lies at the point in a polygon – in addition to if a vertex is better than its neighbors – this must mean the optimal is the global optimal
        + This leads to the Simplex Algorithm which we will detail later

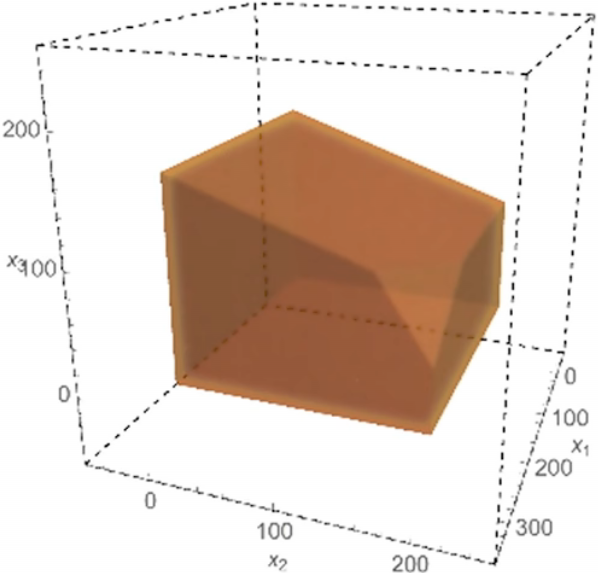
The short story is it’s a local greedy approach

We start at some vertex in the polygon and we check its neighbors

We keep cycling through these vertices until we find one that is better than all its neighbors

3D Linear Programming Example

* Setup
  + Consider a company that makes A, B, and C products
    - Each unit of A makes $1 profit
    - Each unit of B makes $6 profit
    - Each unit of C makes $10 profit
  + Goal: Determine how many of each are needed to be produced to maximize profit
  + Constraints
    - Demand
      * <= 300 units of A a day
      * <= 200 units of B a day
      * C has unlimited demand
    - Supply
      * Workers can work <= 1000 hours per day
      * Each unit of A takes 1 hour
      * Each unit of B takes 3 hours
      * Each unit of C takes 2 hours
    - Packaging
      * We can use <= 500 packaging units / day
      * A takes 0 packaging
      * B takes 1 unit of packaging
      * C takes 3 units of packaging
* Formulating as a Linear Program
  + Variables we will use in our LP
    - Since we are trying to determine the number of units of A, B, and C we will be making, use
      * x1 = number of units of A to produce/day
      * x2 = number of units of B to produce/day
      * x3 = number of units of C to produce/day
    - Constraints – Based off the constraints above
      * Demand
        + 0 <= x1 <= 300
        + 0 <= x2 <= 200
      * Supply
        + x1 + 3x2 + 6x3 <= 1000
      * Packaging
        + x2 + 3x3 <= 500
      * Nonnegative constraints
        + x1, x2, x3 >= 0
  + Our goal is a maximization problem; so form this in terms of A, B, and C
    - Max (x1 + 6x2 + 10x3)
      * This is what we will be maximizing
  + Re-starting formulation
    - Ware are maximizing: x1 + 6x2 + 10x3
    - Subject to constraints
      * x1 <=300
      * x2 <= 200
      * x1, x2, x3 >= 0
      * x1 + 3x2 + 2x3 <= 1000
      * x2 + 3x3 <= 500
  + If any of these were using ‘equals’ we would be using a hyperplane; since they are all inequalities, we are dealing with one side of a hyperplane
* Graphically



* + Notice this is still a convex set; if it was not convex, linear programming could not solve it!

Standard form for Linear Programs

* Variables
  + n variables x1, x2, …, xn
* Objective Function
  + Maximize a linear function of the n variables:

Max c1x1 + c2x2 + … + cnxn

* We are maximizing subject to the m constraints using b as the binding:

a11x1 + a12x2 +… + a1nxn <= b1

a21x1 + a22x2 +… + a2nxn <= b2

…

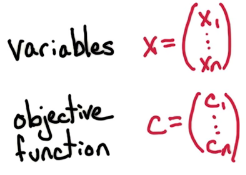
am1x1 + am2x2 +… + amnxn <= bm

x1, x2, …., xn >= 0

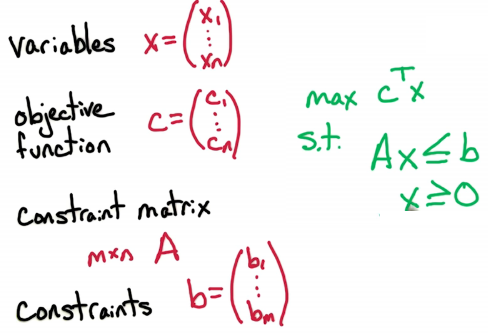
* + Note the **non-negative constraints** at the end
    - BRENTS NOTE: These seem to be ‘understood’ and not always explicitly mentioned…but I am not sure.
  + Note the odd numbering….its a concatenation of ConstraintVariable
  + The above can be streamlined using Linear Algebra

Linear Algebra Review

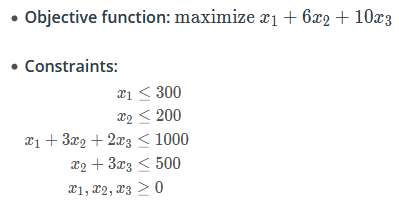
* We can make this into a linear algebra problem
* We make the variables and objective function (I believe the coefficients, so for our 3D example that’s (1, 6, 10)) into a 1\*n matrix (so a **variable matrix** and an **objective function matrix**):



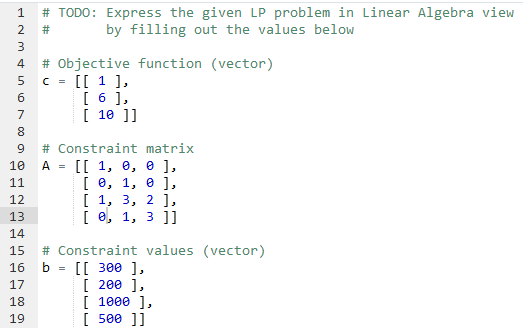
* + Our objective function is to maximize CTX
    - MAX CTX
    - T = transpose in this situation
  + The **constraint matrix** is an m+n matrix (we will call this matrix A, and to see how its constructed look just under the top of this page at the lists of m\*n variables in the list of equations)
  + We also need an additional **constraints matrix** (not to be confused with the constraint matrix, although it seems to be confusing)
    - This is a 1Xm matrix that represents the ‘b’ values above (the right hand side values in the equation lists)
* Recap of matrices:



* + The objective function MAX CTX is subject to the constraints AX <= b as well as x >= 0
    - Keep in mind this is all matrix math above
* Example using 3D problem above:
  + Problem Setup:



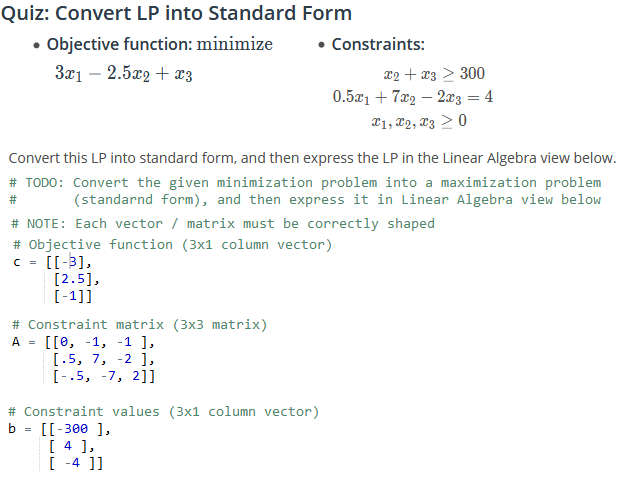
* + Matrix Setup:



* + - Notice that the ‘demand’ constraints are on two separate lines; this is because, unlike the other constraints, demand is independent for each product

Different Situations In Linear Programming

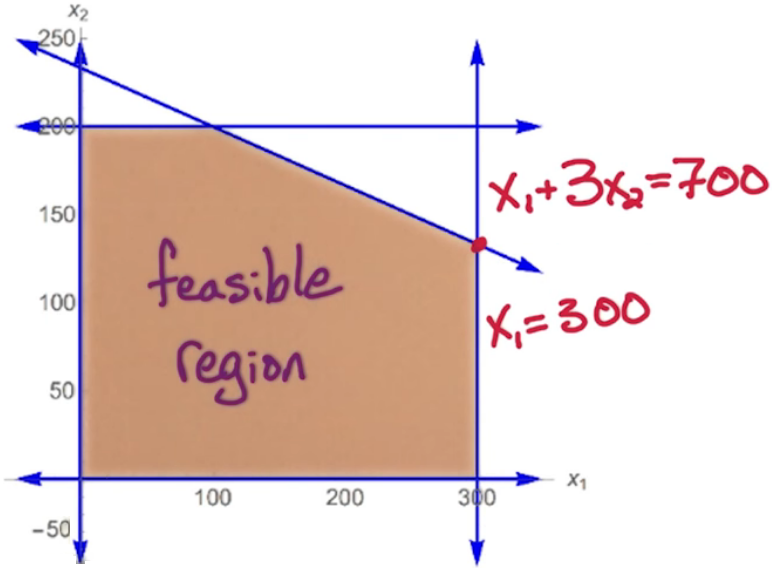
* To minimize a set instead of maximizing:
  + Multiply the resulting matrix CTX by -1
    - Min CTX ⇔ Max - CTX
      * Can be done by -CTX
* We may have to handle constraints that say ‘at least b’:
  + a11x1 + a12x2 +… + a1nxn >= b1
  + To do this, we can again multiply everything by -1:
    - -a11x1 - a12x2 -… - a1nxn <= - b1
* What if we have an equality constraint like:
  + a11x1 + a12x2 +… + a1nxn = b1
  + To do this we can use two constraints and say ‘at most’ and ‘at least’ (which uses the ‘at least b’ above):
    - a11x1 + a12x2 +… + a1nxn <= b1
    - -a11x1 - a12x2 -… - a1nxn <= - b1
  + What if we have a strict inequality (for ex. < 100)?
    - SORRY, strict inequalities (such as < or >) cannot happen in linear programming!
  + What do we do if we have an unconstrained x (where x can be positive or negative)?
    - In this case we don’t have the non-negativity constraint x >= 0
    - To do this we create two new variables: X+ and X-
      * The positive and negative magnitudes of X
    - We constrain X+ to be non-negative: X+ >= 0
    - We constrain X- to be non-negative: X- >= 0
      * BRENTS NOTE: I don’t quite understand this
    - We replace X with X+ - X-:
      * X = (X+) – (X-)
    - Do this and we can consider all variables to be non-negative
* Quiz



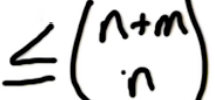
* + NOTE: PAY ATTENTION to the wording. LOOK FOR a min (instead of a max), look for >= instead of <=, and look for = instead of <=. And know < OR > means linear programming will not work!

General Linear Programming Notes

* In general,
  + we have n variables in the objective function (so we will have n dimensions)
  + we have n+m constraints
    - The ‘n’ here is for the non-negativity constraints
      * BRENTS NOTE: These seem to be usually understood unless otherwise specified
  + Feasible Region
    - Each constraint corresponds to a half-space in n dimensions
    - The feasible region are the points which satisfy all n+m constraints
    - This means we will lie in all n+m half-spaces
    - The **feasible region** is the intersection of n+m halfspaces
      * This corresponds to a **convex polyhedron**, where the vertices are the corners of this shape
* How to find vertices in a feasible region



* + If we satisfy both constraints (listed) with equality, we can find the vertices/corners of the convex polyhedron
    - We also have to make sure that point satisfies the other inequalities as well
  + We have to check that n constraints are satisfied with equality, but we have to make sure the remaining m constraints are satisfied with <=
  + At most, the number of vertices is (n+m) choose n:



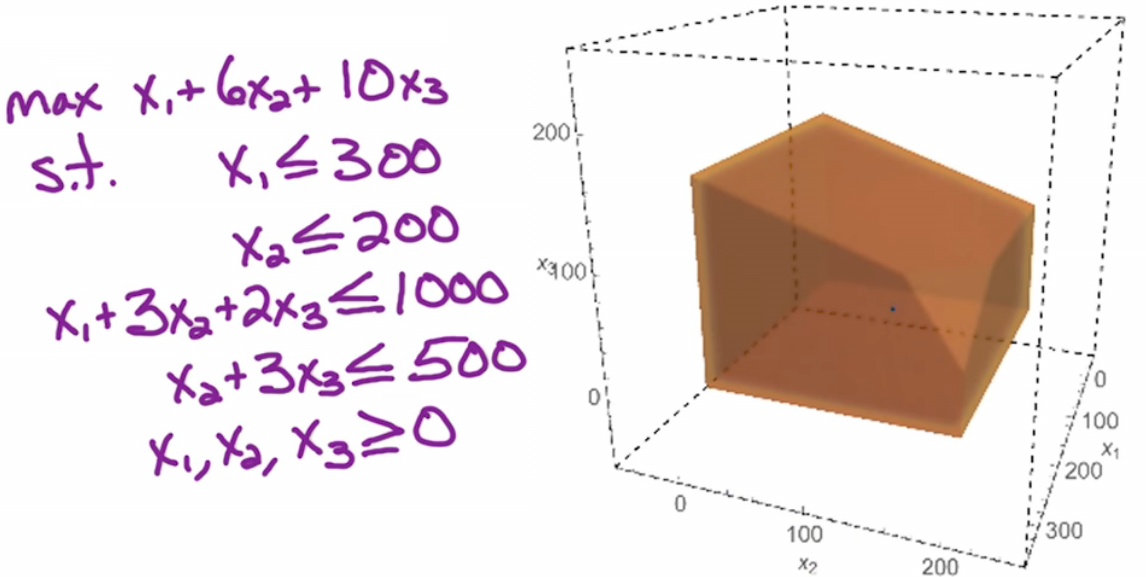
* + - In the worst case, this is exponential in n so there could be a huge list of vertices
  + For a particular vertex, there are at most nm neighbors
* Types of Linear Programming algorithms
  + Polynomial-time algorithms
    - Ellipsoid algorithms
      * These are of more theoretical interests at this point in time
    - Interior point methods
      * These are currently used in a variety of settings
  + Simplex Algorithm
    - Worst-case exponential time
      * Despite this, it is widely used
    - The Simplex Algorithm is used because
      * there is guaranteed to be an optimal
      * It works very fast on huge linear programs

Simplex Algorithm

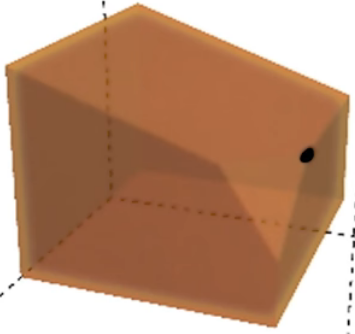
* Overview of Simplex Algorithm
  + Start at x = 0
    - This is the vertex of the feasible region, as it satisfies the n nonnegativity constraints of equality
    - We still must check that this satisfy the other m constraints
    - If this does not satisfy any of the m constraints, we know this is not in the feasible region and therefore the feasible region is empty (so the given LP is not feasible)
      * No x satisfies all of the constraints; halt the algorithm
  + Do a local search
    - Look for a neighboring vertex with a higher objective value
      * This means the value of the objective function is higher
      * This must be strictly higher
  + Move to the neighbor with a higher objective value
    - What to do if there are multiple vertices with a higher objective value
      * This is part of the heuristics of the algorithm and there are many way to do it
        + We can choose one at random
        + We can take the neighbor with the highest objective value
  + Repeat ‘local search’ / ‘move to neighbor’ until no higher vertex is found
    - Recall that the feasible region is a convex polyhedron
      * Meaning the value can go down, but it cannot go back up
    - If all the neighbors are smaller, this means the whole feasible region is smaller, so the current point is optimal
  + Output current point (once optimal found in the previous repeat step)
    - This is guaranteed to be the global optimal

Example of the Simplex Algorithm

* We will use our 3D LP introduced earlier in the lecture:



* Start at point (0, 0, 0)
  + This point is the back bottom corner point
  + We check this to make sure its feasible (it is)
  + The value of the objective function at this point is 0
  + This is done by satisfying the 3 constraints (x1, x2, x3 >= 0) with an equality (=0) and all other constraints with an inequality
* We look at a neighbor (front bottom left corner)
  + This is the point (300, 0, 0)
  + This constraint is satisfied by an equality (x1 = 300) and t2 other equalities (he last two nonnegativity constraints (x2, x3))
  + All other constraints are satisfied with an inequality
  + Since plugging (300, 0, 0) yields a higher objective function value (max (x3 + 6 x2 + 10x3)) we move to this point
* We move to a neighbor (front bottom right corner)
  + This is the point (300, 200, 0)
  + This constraint is satisfied by an equality (x1 = 300), another equality (x2 = 200) and the last nonnegativity constraint (x3)
  + All other constraints are satisfied with an inequality
  + Since plugging (300, 200, 0) yields a higher objective function value (max (x3 + 6 x2 + 10x3)) we move to this point
* We move to a neighbor (point highlighted below)

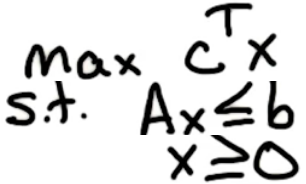


* + This is the point (200, 200, 100)
  + This constraint is satisfied by an equality (x2 = 300), and is then bound by the (now equalities, since they are maxed)
    - x2 + 3x3 <= 500
      * This is actually now an equality since x2 = 200, x3 can be no larger than 100
    - x1 + 3x2 + 2x3 <= 1000
      * This is actually now an equality since x2 = 200 and x3 = 100, x1 can be no larger than 200
  + All other constraints are satisfied with an inequality
  + Since plugging (200, 200, 100) yields a higher objective function value (max (x3 + 6 x2 + 10x3)) we move to this point
* We check all neighbors and since (200, 200, 100) yields the highest return on the objective function, we have found our answer: (200, 200, 100)

LP2

Linear Programming Geometry

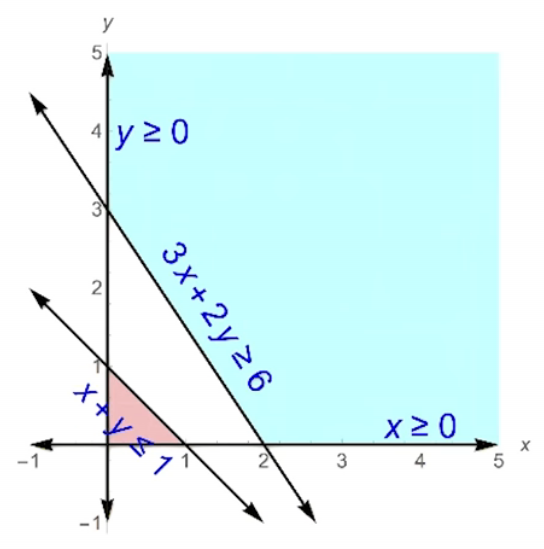
* Consider a linear program in standard form



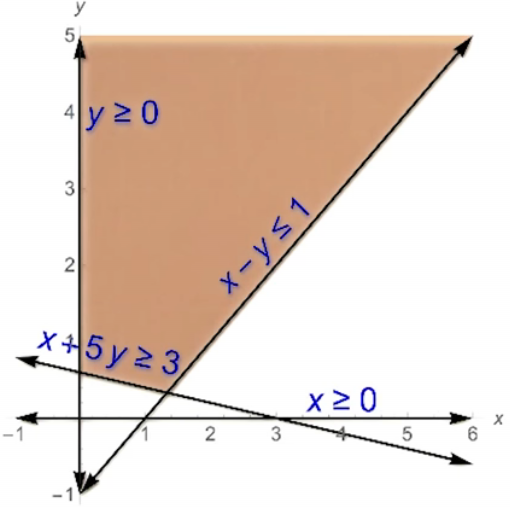
* + There are n variables
    - ‘X’ above is a column vector of size n
  + There are ‘m’ constraints
    - Each one forms a half-space in n dimensions
  + The feasible region is the set of X’s that satisfy the constraints
    - This is a convex set
* Recall the Simplex Algorithm does a walk on vertices (corners) of the convex polyhedron
* Since the feasible region is convex, the optimum are achieved at vertices in the feasible region
  + This is not always true, and part of this lecture will talk about this

LP Optimum

* The optimum of Linear Programming is achieved at a vertex of the feasible region EXCEPT if its:
  + Infeasible
  + Unbounded
* Exploring Infeasibility
  + **Infeasible** means the feasible region is empty
  + Example: Max(5x – 7y)
    - Subject to (s.t.)
      * x+y<= 1
      * 3x + 2y >= 6
      * x, y >= 0
    - Graphed:



* + - * Notice that the two halfspaces defined by the two constraints do not intersect
      * The feasible region is empty
    - Notice infeasibility has nothing to do with the objective function (in this case that’s max 5x – 7y)
* Exploring Unbounded
  + Unbounded means the objective function is arbitrarily large
  + Example
    - Setup
      * Objective function: MAX x + y
      * Subject to (st)
        + x – y <= 1
        + x + 5y >= 3
        + x, y >= 0
    - Graph of the feasible region:



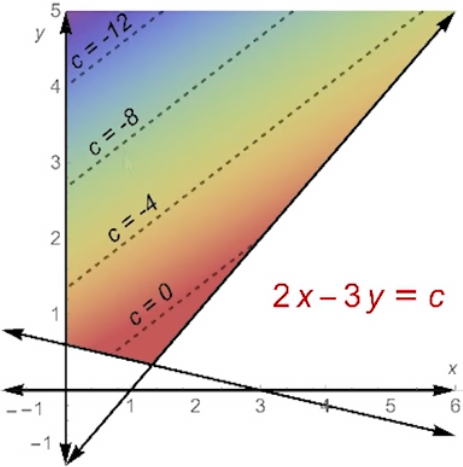
* + - The picture truncates it, but the feasible region is unbounded as y increases
    - Here, the objective function is unbounded because the optimal value, based on how its calculated in the objective function, has no limit and is thus undefined
  + Note that just because one ‘side’ may not be bounded does NOT mean its objective function is unbounded
    - Example
      * Setup
        + Objective function: MAX 2x - 3y
        + Subject to (st)

x – y <= 1

x + 5y >= 3

x, y >= 0

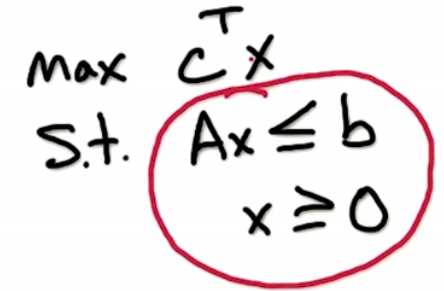
* + - * Graph of the feasible region:



* + - * + Notice that the optimal answer actually lies at the bottom vertex, so this is solvable even though the top portion is unbounded
* Whether an objective function is feasible or infeasible completely depends on the constraints alone.
* Whether an objective function is bounded or unbounded depends on the constraints AND the objective function.

Checking For Infeasibility

* How can we check whether a Linear Program is feasible or infeasible, and if it is feasible, how can we check to see if its bounded?
  + In order to figure out if an LP is bounded or unbounded we will need to look at the Dual LP (which will be discussed later)
* Given a LP in standard form, we would like to figure out if there are some X’s that satisfy the constraints



* + Note that this has nothing to do with the objective function
    - We are simply seeing IF an answer exists in the feasible region via investigating the constraints
      * We are seeing if the feasible region is empty or not
  + Steps
    - Create a new variable z
      * This is a scalar variable – NOT a vector
    - Consider one constraint
      * For example, a1x1 + … + anxn <= b
        + We also apply the non-negativity constraints as well, so (x1, …, xn >= 0) is assumed
    - Add in z to the constraint
      * For example, a1x1 + … + anxn + z <= b
        + Note z is NOT constrained to be non-negative
      * This constraint makes this check trivial, as z could be negative infinity (although practically it will not be)
    - If its possible to get z>= 0 and the constraint is still satisfied, we can drop z and the original constraint is still satisfied
  + The idea here is in order to figure out of constraints are feasible we run an LP with the constraints plus z, where z is unbounded; IF there is a z >=0, the original LP has a feasible answer!
    - This is done by using the objective function (MAX z); if this returns z >= 0 the original constraints are feasible
      * Using this temp objective function will determine if the original constraints are feasible for the original objective function
    - Determining if the LP is feasible via using z in this fashion ALSO gives us the bonus of having a starting point for the Simplex Algorithm, as we have a set of viable points that may be maximum found during this process!