Lecture 14: Primal and Dual Methods For Large Optimization Problems

ISyE 6673: Financial Optimization

Dr. Arthur Delarue
arthur.delarue@isye.gatech.edu

Stewart School of Industrial and Systems Engineering
Georgia Institute of Technology

October 19, 2022

Size of a linear program

- How should we think about the *size* of a linear program?
- Two numbers:
  - Problem dimension n: number of decision variables
  - Number of constraints m: this is actually the dimension of the dual problem!

- Both numbers affect tractability
  - Number of extreme points scales like $\binom{m}{n} \sim m^{m-n}$ in the worst case

  - The constraint matrix $\mathbf{A}$ is $m \times n$. If m and n are both 100,000, the matrix might have 10 billion entries, which may not fit in memory!

Solving large-scale optimization problems

Today's objective

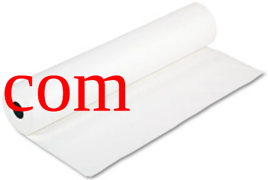
How to solve very large linear programs efficiently!

- Too many variables: column generation
- Too many constraints: cutting planes

Column generation

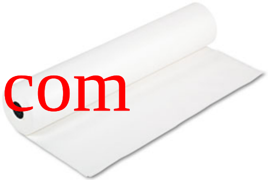
The cutting stock problem

- A classic optimization problem
- You are a paper company, and you produce rolls of paper, each of width W
- Note that it's called "width" but when the roll is fully rolled up it's probably the longest dimension.
- Your m customers don't want rolls of width W. In fact, each customer i wants b_i rolls of width w_i



The cutting stock problem

- A classic optimization problem
- You are a paper company, and you produce rolls of paper, each of width W
- Note that it's called "width" but when the roll is fully rolled up it's probably the longest dimension.
- Your m customers don't want rolls of width W. In fact, each customer i wants b_i rolls of width w_i



How do we meet customer demand?

We have to figure out how to cut the rolls!

Example

- Let's say $W = 10$

Example

- Let's say $W = 10$
- $m = 1$, $b_1 = 6$, $w_1 = 5$: we can cut three rolls of width 10 into two rolls of width 5 each, for a total of 6 rolls of width 5

Example

- Let's say $W = 10$
- $m = 1$, $b_1 = 6$, $w_1 = 5$: we can cut three rolls of width 10 into two rolls of width 5 each, for a total of 6 rolls of width 5
- $m = 1$, $b_1 = 6$, $w_1 = 9$: we can cut six rolls of width 10, each into one roll of width 9 and one roll of width 1. We end up with six rolls of width 9 to satisfy the customer, and six rolls of width 1 are *wasted*

Example

- Let's say $W = 10$
- $m = 1$, $b_1 = 6$, $w_1 = 5$: we can cut three rolls of width 10 into two rolls of width 5 each, for a total of 6 rolls of width 5
- $m = 1$, $b_1 = 6$, $w_1 = 9$: we can cut six rolls of width 10, each into one roll of width 9 and one roll of width 1. We end up with six rolls of width 9 to satisfy the customer, and six rolls of width 1 are *wasted*
- What about when $n > 1$? We need linear programming!

Formulating the cutting stock problem

Question: How many ways are there to cut a stock of width W?

Answer: A LOT!

Formulating the cutting stock problem

Question: How many ways are there to cut a stock of width W?

Answer: A LOT!

Example

Assume only want integer widths, a roll of width 4 can be cut into:

- 4 rolls of width 1
- 2 rolls of width 2
- 2 rolls of width 1, 1 roll of width 2
- 1 roll of width 3 and 1 roll of width 1

- Number of possible "cutting patterns" is exponential in W!
- Even though we only care about widths w_i that are actually requested by consumers, the number of possible cutting patterns could still be really large!

Representing cutting patterns as vectors

- We can write pattern j as a column vector $\mathbf{A}_j$ of height m
  - $\mathbf{A}_j = (a_{1j}, \ldots, a_{mj})$
  - a_{ij} is the number of rolls of width w_i produced by this pattern

Example

- A roll of width $W = 70$ can be cut into 3 rolls of width $w_1 = 15$ and 1 roll of width $w_2 = 17$ (wasting a roll of width 8)
  - We can write this pattern as $(3, 1)$

- Alternatively the same roll can be cut into 1 roll of width $w_1 = 15$ and 3 rolls of width $w_2 = 17$ (wasting a roll of width 4)
  - We can write this pattern as $(1, 3)$

Characterizing a feasible pattern

- A pattern represented by vector A_j is feasible if the sum of the widths don't exceed the width of the roll

$$\sum_{i=1}^{m} a_{ij} w_i \leq W$$

Characterizing a feasible pattern

- A pattern represented by vector A_j is feasible if the sum of the widths don't exceed the width of the roll

$$\sum_{i=1}^{m} a_{ij} w_i \leq W$$

- The "waste" r_j of a pattern is any remaining roll with a width that is not one of the approved customer widths w_i

$$r_j = W - \sum_{i=1}^{m} a_{ij} w_i \geq 0$$

Formulating the cutting stock problem

- Let J designate the number of possible feasible patterns

Formulating the cutting stock problem

- Let J designate the number of possible feasible patterns
- Decision variable x_j is the number of rolls cut with pattern j (integer, but continuous is ok for large demand)

Formulating the cutting stock problem

- Let J designate the number of possible feasible patterns
- Decision variable x_j is the number of rolls cut with pattern j (integer, but continuous is ok for large demand)
- We can formulate the cutting stock problem as follows:

$$
\begin{aligned}
\text{min} \quad & \sum_{j=1}^{J} x_j \\
\text{s.t.} \quad & \sum_{j=1}^{J} a_{ij} x_j \geq w_i & \forall\, 1 \leq i \leq m \\
& x_j \geq 0 & \forall\, j \in J
\end{aligned}
$$

Formulating the cutting stock problem

$$
\begin{aligned}
\min \quad & \sum_{j=1}^{J} x_j & \\
\text{s.t.} \quad & \sum_{j=1}^{J} a_{ij} x_j \geq b_i & \forall\, 1 \leq i \leq m \\
& x_j \geq 0 & \forall\, j \in J
\end{aligned}
$$

- Constraints seek to meet the demand of each customer i

Formulating the cutting stock problem

$$
\begin{aligned}
\min \quad & \sum_{j=1}^{J} x_j \\
\text{s.t.} \quad & \sum_{j=1}^{J} a_{ij} x_j \geq b_i && \forall\, 1 \leq i \leq m \\
& x_j \geq 0 && \forall\, j \in J
\end{aligned}
$$

- Constraints seek to meet the demand of each customer i
- Objective is to minimize the total number of rolls

Formulating the cutting stock problem

$$\min \sum_{j=1}^{J} x_j$$

$$\text{s.t.} \quad \sum_{j=1}^{J} a_{ij} x_j \geq b_i \qquad \forall\, 1 \leq i \leq m$$

$$x_j \geq 0 \qquad \forall\, j \in J$$

- Constraints seek to meet the demand of each customer i
- Objective is to minimize the total number of rolls
- Alternatively we can minimize total waste:

$$\min \sum_{j=1}^{J} r_j x_j$$

Formulation size

- We can write the formulation in partial vector form as

$$\begin{aligned} \min \quad & \sum_{j=1}^{J} x_j \\ \text{s.t.} \quad & \sum_{j=1}^{J} \mathbf{A}_j x_j \geq \mathbf{b} \\ & \mathbf{x} \geq 0 \end{aligned}$$

Formulation size

- We can write the formulation in partial vector form as

$$\begin{aligned} \min \quad & \sum_{j=1}^{J} x_j \\ \text{s.t.} \quad & \sum_{j=1}^{J} \mathbf{A}_j x_j \geq \mathbf{b} \\ & \mathbf{x} \geq 0 \end{aligned}$$

- Notice that the number of variables is huge!

Formulation size

- We can write the formulation in partial vector form as

$$\begin{aligned} \min \quad & \sum_{j=1}^{J} x_j \\ \text{s.t.} \quad & \sum_{j=1}^{J} \mathbf{A}_j x_j \geq \mathbf{b} \\ & \mathbf{x} \geq 0 \end{aligned}$$

- Notice that the number of variables is huge!
- Another way to say it is that the matrix $\mathbf{A}$ has too many columns!

Formulation size

- We can write the formulation in partial vector form as

$$\begin{aligned} \min \quad & \sum_{j=1}^{J} x_j \\ \text{s.t.} \quad & \sum_{j=1}^{J} \mathbf{A}_j x_j \geq \mathbf{b} \\ & \mathbf{x} \geq 0 \end{aligned}$$

- Notice that the number of variables is huge!
- Another way to say it is that the matrix $\mathbf{A}$ has too many columns!
- An optimal BFS will have at most m nonzero variables, corresponding to a basis matrix with at most m columns of $\mathbf{A}$

Formulation size

- We can write the formulation in partial vector form as

$$\begin{aligned} \min \quad & \sum_{j=1}^{J} x_j \\ \text{s.t.} \quad & \sum_{j=1}^{J} \mathbf{A}_j x_j \geq \mathbf{b} \\ & \mathbf{x} \geq 0 \end{aligned}$$

- Notice that the number of variables is huge!
- Another way to say it is that the matrix $\mathbf{A}$ has too many columns!

- An optimal BFS will have at most m nonzero variables, corresponding to a basis matrix with at most m columns of $\mathbf{A}$
- Why should we have a formulation with $J \gg m$ variables if we will only need m nonzero variables at optimality?

Idea: Restrict the set of patterns

- We can restrict ourselves to a subset of patterns $\mathcal{J} \subseteq \{1, \ldots, J\}$

$$\begin{aligned} \min \quad & \sum_{j \in \mathcal{J}} x_j \\ \text{s.t.} \quad & \sum_{j \in \mathcal{J}} \mathbf{A}_j x_j \geq \mathbf{b} \\ & \mathbf{x} \geq 0 \end{aligned}$$

Idea: Restrict the set of patterns

- We can restrict ourselves to a subset of patterns $\mathcal{J} \subseteq \{1, \dots, J\}$

$$\begin{aligned} \text{min} \quad & \sum_{j \in \mathcal{J}} x_j \\ \text{s.t.} \quad & \sum_{j \in \mathcal{J}} \mathbf{A}_j x_j \geq \mathbf{b} \\ & \mathbf{x} \geq 0 \end{aligned}$$

- If $\mathcal{J}$ contains the m optimal patterns then the optimal solution of the restricted problem is optimal for the full problem

Idea: Restrict the set of patterns

- We can restrict ourselves to a subset of patterns $\mathcal{J} \subseteq \{1, \ldots, J\}$

$$\begin{aligned} \min \quad & \sum_{j \in \mathcal{J}} x_j \\ \text{s.t.} \quad & \sum_{j \in \mathcal{J}} \mathbf{A}_j x_j \geq \mathbf{b} \\ & \mathbf{x} \geq 0 \end{aligned}$$

- If $\mathcal{J}$ contains the m optimal patterns then the optimal solution of the restricted problem is optimal for the full problem
- If $\mathcal{J}$ is missing at least one optimal pattern, the optimal solution of the restricted problem is suboptimal in the full problem

Solving the restricted problem

- The restricted problem is a lot more tractable (fewer decision variables) but it may produce suboptimal solutions
- Can we even tell if the solution we obtain is optimal in the full problem?

Recall the simplex method!

A basic feasible solution is optimal if all of the associated reduced costs are non-negative!

- How can we check if the reduced costs of the nonbasic variables are non-negative?

Basic variables and reduced costs

- Say we solve the restricted problem with the simplex method and obtain an optimal BFS, there are three kinds of variables:

Basic variables and reduced costs

- Say we solve the restricted problem with the simplex method and obtain an optimal BFS, there are three kinds of variables:
  - Basic variables $j \in \{B(1), \ldots, B(m)\} \subseteq \mathcal{J}$

Basic variables and reduced costs

- Say we solve the restricted problem with the simplex method and obtain an optimal BFS, there are three kinds of variables:
  - Basic variables $j \in \{B(1), \ldots, B(m)\} \subseteq \mathcal{J}$
  - Non-basic variables $j \in \mathcal{J} \backslash \{B(1), \ldots, B(m)\}$

Basic variables and reduced costs

- Say we solve the restricted problem with the simplex method and obtain an optimal BFS, there are three kinds of variables:
  - Basic variables $j \in \{B(1), \ldots, B(m)\} \subseteq \mathcal{J}$
  - Non-basic variables $j \in \mathcal{J} \setminus \{B(1), \ldots, B(m)\}$
  - Variables that are not even defined in the restricted problem ($j \notin \mathcal{J}$); these are also non-basic by default

Basic variables and reduced costs

- Say we solve the restricted problem with the simplex method and obtain an optimal BFS, there are three kinds of variables:
  - Basic variables $j \in \{B(1), \ldots, B(m)\} \subseteq \mathcal{J}$
  - Non-basic variables $j \in \mathcal{J} \setminus \{B(1), \ldots, B(m)\}$
  - Variables that are not even defined in the restricted problem ($j \notin \mathcal{J}$); these are also non-basic by default
- Computing reduced costs:

Basic variables and reduced costs

- Say we solve the restricted problem with the simplex method and obtain an optimal BFS, there are three kinds of variables:
  - Basic variables $j \in \{B(1), \ldots, B(m)\} \subseteq \mathcal{J}$
  - Non-basic variables $j \in \mathcal{J} \setminus \{B(1), \ldots, B(m)\}$
  - Variables that are not even defined in the restricted problem ($j \notin \mathcal{J}$); these are also non-basic by default
- Computing reduced costs:
  - The basic variables have reduced cost 0 by definition

Basic variables and reduced costs

- Say we solve the restricted problem with the simplex method and obtain an optimal BFS, there are three kinds of variables:
  - Basic variables $j \in \{B(1), \ldots, B(m)\} \subseteq \mathcal{J}$
  - Non-basic variables $j \in \mathcal{J} \setminus \{B(1), \ldots, B(m)\}$
  - Variables that are not even defined in the restricted problem ($j \notin \mathcal{J}$); these are also non-basic by default

- Computing reduced costs:
  - The basic variables have reduced cost 0 by definition
  - The non-basic variables have non-negative reduced cost because the solution is optimal in the restricted problem

Basic variables and reduced costs

- Say we solve the restricted problem with the simplex method and obtain an optimal BFS, there are three kinds of variables:
  - Basic variables $j \in \{B(1), \ldots, B(m)\} \subseteq \mathcal{J}$
  - Non-basic variables $j \in \mathcal{J} \backslash \{B(1), \ldots, B(m)\}$
  - Variables that are not even defined in the restricted problem ($j \notin \mathcal{J}$); these are also non-basic by default

- Computing reduced costs:
  - The basic variables have reduced cost 0 by definition
  - The non-basic variables have non-negative reduced cost because the solution is optimal in the restricted problem
  - The undefined variables have **unknown** reduced costs

Basic variables and reduced costs

- Say we solve the restricted problem with the simplex method and obtain an optimal BFS, there are three kinds of variables:
  - Basic variables $j \in \{B(1), \ldots, B(m)\} \subseteq \mathcal{J}$
  - Non-basic variables $j \in \mathcal{J} \backslash \{B(1), \ldots, B(m)\}$
  - Variables that are not even defined in the restricted problem ($j \notin \mathcal{J}$); these are also non-basic by default
- Computing reduced costs:
  - The basic variables have reduced cost 0 by definition
  - The non-basic variables have non-negative reduced cost because the solution is optimal in the restricted problem
  - The undefined variables have **unknown** reduced costs
- However, we know the formula for reduced costs!

Computing reduced costs

Definition reminder: reduced cost

For a non-basic variable with index $j \notin \mathcal{J}$,

$$\bar{c}_j = c_j - \mathbf{c}_B^\top \mathbf{B}^{-1} \mathbf{A}_j$$

Computing reduced costs

Definition reminder: reduced cost

For a non-basic variable with index $j \notin \mathcal{J}$,

$$\bar{c}_j = c_j - \mathbf{c}_B^\top \mathbf{B}^{-1} \mathbf{A}_j$$

- We can use this definition to compute the reduced cost of patterns that were not included in the restricted problem.

- **Problem:** Checking that every single reduced cost is negative might still take a long time if we do them one by one!

Finding the most negative reduced cost

- Consider the following optimization problem:

$$z^* = \min_{j \notin \mathcal{J}} \bar{c}_j$$

Finding the most negative reduced cost

- Consider the following optimization problem:

$$z^* = \min_{j \notin \mathcal{J}} \bar{c}_j$$

- If $z^* \geq 0$, then all the reduced costs are non-negative and the solution is optimal in the full problem

Finding the most negative reduced cost

- Consider the following optimization problem:

$$z^* = \min_{j \notin \mathcal{J}} \bar{c}_j$$

- If $z^* \geq 0$, then all the reduced costs are non-negative and the solution is optimal in the full problem
- If $z^* < 0$, then we've found a column/pattern with negative reduced cost!

Finding the most negative reduced cost

- Consider the following optimization problem:

$$z^* = \min_{j \notin \mathcal{J}} \bar{c}_j$$

- If $z^* \geq 0$, then all the reduced costs are non-negative and the solution is optimal in the full problem
- If $z^* < 0$, then we've found a column/pattern with negative reduced cost!

Problem

How do we solve this *subproblem*?

Solving the subproblem

- Recall that we can describe a feasible pattern $\mathbf{a}$ as follows:

$$\sum_{i=1}^{m} a_i w_i \leq W$$

- We can simplify the expression of the reduced cost of pattern $\mathbf{a}$ since every pattern has cost 1

$$\bar{c} = 1 - \mathbf{c}_B^\top \mathbf{B}^{-1}\mathbf{a} = 1 - \mathbf{p}^\top \mathbf{a} = 1 - \sum_{i=1}^{m} p_i a_i.$$

where p_i is the dual optimal solution in the restricted problem

- Minimizing $\bar{c}$ is equivalent to maximizing $\sum_{i=1} p_i a_i$

Solving the subproblem

- So we can write the subproblem as follows:

$$
\begin{aligned}
z^* = \max \quad & \sum_{i=1}^{m} p_i a_i \\
\text{s.t.} \quad & \sum_{i=1}^{m} a_i w_i \leq W \\
& a_i \in \mathbb{Z}^+ \qquad \forall\, i \in [m]
\end{aligned}
$$

Solving the subproblem

- So we can write the subproblem as follows:

$$
\begin{aligned}
z^* = \max \quad & \sum_{i=1}^{m} p_i a_i \\
\text{s.t.} \quad & \sum_{i=1}^{m} a_i w_i \leq W \\
& a_i \in \mathbb{Z}^+ && \forall\, i \in [m]
\end{aligned}
$$

- $z^* > 1 \Rightarrow$ found pattern with negative reduced cost

Solving the subproblem

- So we can write the subproblem as follows:

$$
\begin{aligned}
z^* = \max \quad & \sum_{i=1}^{m} p_i a_i \\
\text{s.t.} \quad & \sum_{i=1}^{m} a_i w_i \leq W \\
& a_i \in \mathbb{Z}^+ \qquad\qquad \forall\, i \in [m]
\end{aligned}
$$

- $z^* > 1 \Rightarrow$ found pattern with negative reduced cost
- We need to keep the integer variables because a_i is the number of rolls of width w_i to include in the pattern: not easily rounded

Solving the subproblem

- So we can write the subproblem as follows:

$$
\begin{aligned}
z^* = \max \quad & \sum_{i=1}^{m} p_i a_i \\
\text{s.t.} \quad & \sum_{i=1}^{m} a_i w_i \le W \\
& a_i \in \mathbb{Z}^+ && \forall\, i \in [m]
\end{aligned}
$$

- $z^* > 1 \Rightarrow$ found pattern with negative reduced cost
- We need to keep the integer variables because a_i is the number of rolls of width w_i to include in the pattern: not easily rounded
- This problem is called the knapsack problem

Knapsack problem

- A classic combinatorial optimization problem

Knapsack problem

- A classic combinatorial optimization problem
- We have a knapsack with maximum weight capacity W and m item types
  - Each item i has a value p_i
  - Each item i also has a weight w_i

Knapsack problem

- A classic combinatorial optimization problem
- We have a knapsack with maximum weight capacity W and m item types
  - Each item i has a value p_i
  - Each item i also has a weight w_i
- We need to decide how many of each item type to take to maximize value while respecting the weight constraint

Knapsack problem

- A classic combinatorial optimization problem
- We have a knapsack with maximum weight capacity W and m item types
  - Each item i has a value p_i
  - Each item i also has a weight w_i
- We need to decide how many of each item type to take to maximize value while respecting the weight constraint

Solving the knapsack problem

We can solve the problem in two ways:

1. Using integer optimization (Lecture 20)
2. Using dynamic programming (Lecture 21)

For now all we need to know is that we *can* solve it easily

Column generation

The column generation algorithm

Consider a minimization problem with many variables (equivalently, constraint matrix has many columns)

1. Initialize a subset of columns $\mathcal{J}$
2. Solve the minimization problem restricted to $\mathcal{J}$ and get an optimal primal and dual solution
3. Use the optimal dual solution to formulate the subproblem: finding the column j with lowest reduced cost
4. Solve the subproblem and obtain an optimal objective z^* and column $\mathbf{A}_j$
   - If $z^* \geq 0$, all columns have non-negative reduced cost and the current primal solution is optimal $\Rightarrow$ Algorithm terminates
   - If $z^* < 0$, column $\mathbf{A}_j$ has negative reduced cost: add j to $\mathcal{J}$ and go to step 2.

Implementation details

How to initialize the set of columns $\mathcal{J}$?

An easy solution is m starting patterns, where the i-th pattern has $\lfloor W/w_i \rfloor$ rolls of width w_i.

What to do with variables exiting the basis?

Two options, depending on memory and other technical implementation details

1. Keep them in $\mathcal{J}$ (restricted problem size grows at each iteration but subproblem becomes easier)
2. Discard them from $\mathcal{J}$ (restricted problem size stays constant but subproblem is a bit harder)

Cutting planes

Lecture 5 callback: Kelley's algorithm

Problem

Consider the convex optimization problem

$$\begin{aligned} \min \quad & f(\mathbf{x}) \\ \text{s.t.} \quad & \mathbf{A}\mathbf{x} \leq \mathbf{b} \\ & \mathbf{x} \in \mathbb{R}^n \end{aligned}$$

where f is an arbitrary differentiable convex function

Lecture 5 callback: Kelley's algorithm

Problem

Consider the convex optimization problem

$$\begin{aligned} \min \quad & f(\mathbf{x}) \\ \text{s.t.} \quad & \mathbf{A}\mathbf{x} \leq \mathbf{b} \\ & \mathbf{x} \in \mathbb{R}^n \end{aligned}$$

where f is an arbitrary differentiable convex function

Claim

We can solve this problem using linear programming!

Lecture 5 callback: Kelley's algorithm

Iterative algorithm: start with $\mathbf{x}^0, LB, UB, \varepsilon$. Set $i = 0$

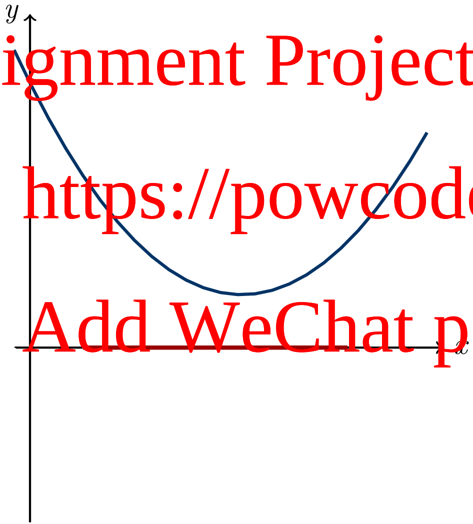
While $UB - LB \geq \varepsilon$:

1. Solve the LP:

$$\begin{aligned} \min \quad & z \\ \text{s.t.} \quad & \mathbf{A}\mathbf{x} \leq \mathbf{b} \\ & z \geq f(\mathbf{x}^k) + \nabla f(\mathbf{x}^k)^T(\mathbf{x} - \mathbf{x}^k) \quad 0 \leq k \leq 1 \end{aligned}$$

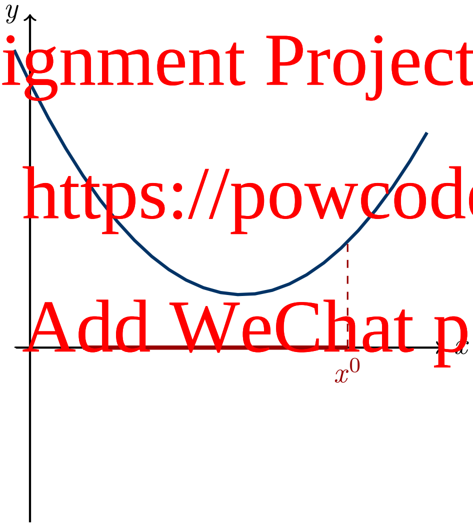
2. Obtain optimal solution $z^*, \mathbf{x}^*$
3. Update $LB \leftarrow z^*$
4. Update $UB \leftarrow \min(UB, f(\mathbf{x}^*))$
5. Store $\mathbf{x}^{i+1} \leftarrow \mathbf{x}^*$
6. Increment $i \leftarrow i + 1$

Lecture 5 callback: Kelley's algorithm



$$\min \quad \frac{(x-4)^2}{4} + 1$$
$$\text{s.t.} \quad 1 \le x \le 6$$

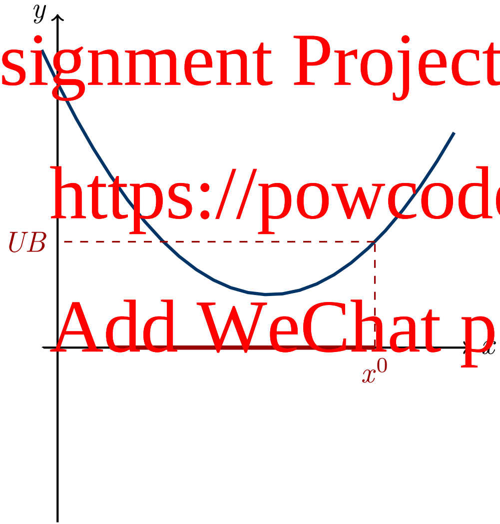
Lecture 5 callback: Kelley's algorithm



$$\min \quad \frac{(x-4)^2}{4} + 1$$
$$\text{s.t.} \quad 1 \le x \le 6$$

$$x^0 = 6$$

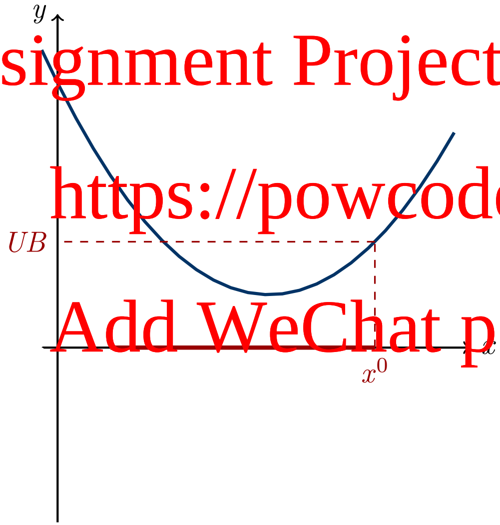
Lecture 5 callback: Kelley's algorithm



$$\min \quad \frac{(x-4)^2}{4} + 1$$
$$\text{s.t.} \quad 1 \le x \le 6$$

$$x^0 = 6$$
$$UB = 2$$

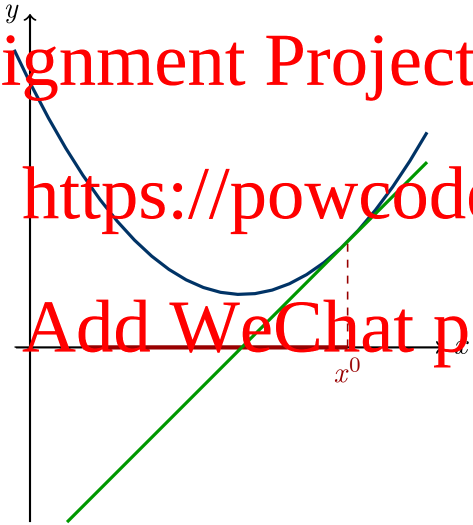
Lecture 5 callback: Kelley's algorithm



$$\begin{aligned} \min \quad & \frac{(x-4)^2}{4} + 1 \\ \text{s.t.} \quad & 1 \le x \le 6 \end{aligned}$$

$$\begin{aligned} x^0 &= 6 \\ UB &= 2 \\ LB &= -\infty \end{aligned}$$

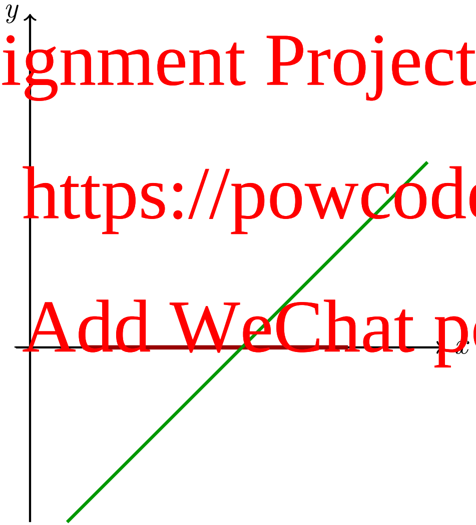
Lecture 5 callback: Kelley's algorithm



$$\min \quad \frac{(x-4)^2}{4} + 1$$
$$\text{s.t.} \quad 1 \leq x \leq 6$$

$$x^0 = \quad 6$$
$$UB = \quad 2$$
$$LB = \quad -\infty$$

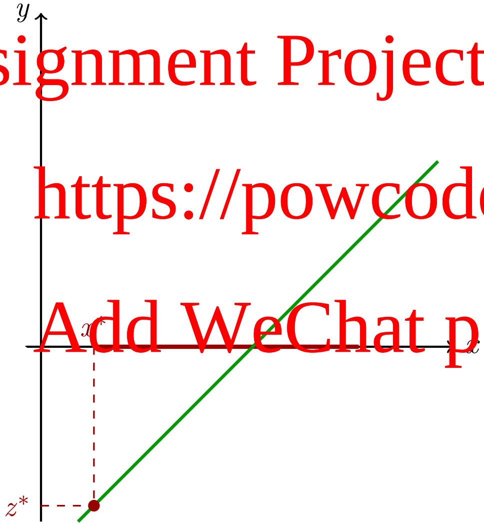
Lecture 5 callback: Kelley's algorithm



$$\begin{aligned} \text{min} \quad & \frac{(x-4)^2}{4} + 1 \\ \text{s.t.} \quad & 1 \le x \le 6 \end{aligned}$$

$$\begin{aligned} x^0 &= \quad 6 \\ UB &= \quad 2 \\ LB &= \quad -\infty \end{aligned}$$

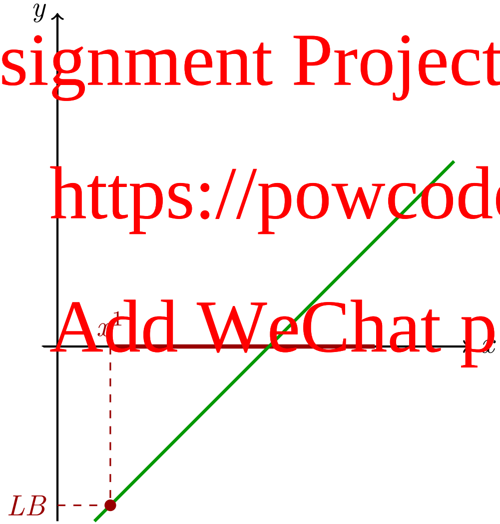
Lecture 5 callback: Kelley's algorithm



$$\min \quad \frac{(x-4)^2}{4} + 1$$
$$\text{s.t.} \quad 1 \le x \le 6$$

$$x^0 = 6$$
$$UB = 2$$
$$LB = -\infty$$

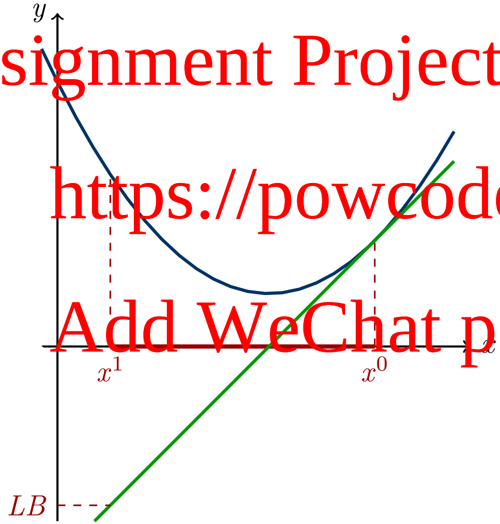
Lecture 5 callback: Kelley's algorithm



$$\min \quad \frac{(x-4)^2}{4} + 1$$
$$\text{s.t.} \quad 1 \le x \le 6$$

$$x^1 = 1$$
$$UB = 2$$
$$LB = -3$$

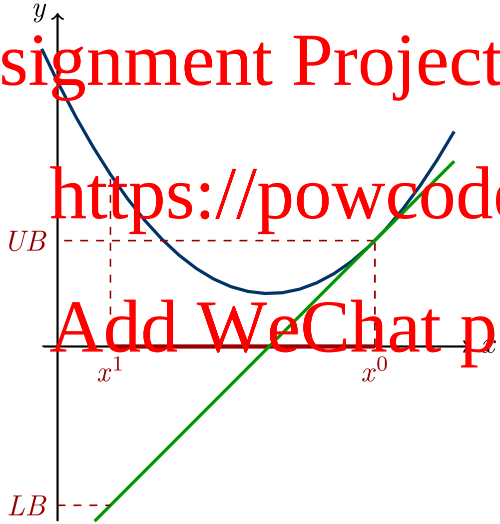
Lecture 5 callback: Kelley's algorithm



$$\begin{aligned} \text{min} \quad & \frac{(x-4)^2}{4} + 1 \\ \text{s.t.} \quad & 1 \leq x \leq 6 \end{aligned}$$

$$\begin{aligned} x^1 &= \quad 1 \\ UB &= \quad 2 \\ LB &= \quad -3 \end{aligned}$$

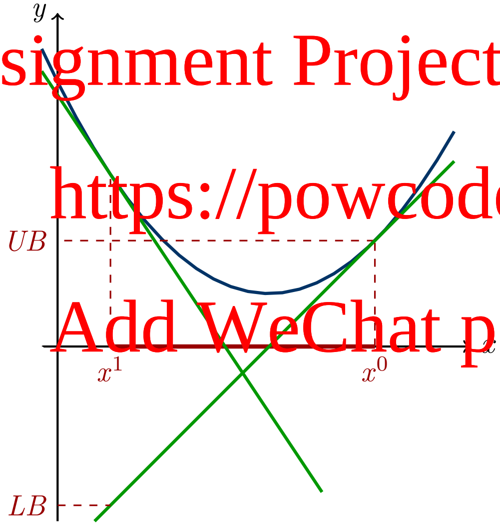
Lecture 5 callback: Kelley's algorithm



$$\begin{aligned} \text{min} \quad & \frac{(x-4)^2}{4} + 1 \\ \text{s.t.} \quad & 1 \leq x \leq 6 \end{aligned}$$

$$\begin{aligned} x^1 &= \quad 1 \\ UB &= \quad 2 \\ LB &= \quad -3 \end{aligned}$$

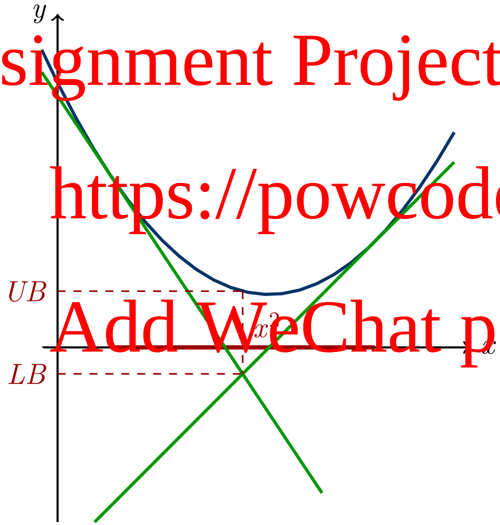
Lecture 5 callback: Kelley's algorithm



$$\min \quad \frac{(x-4)^2}{4} + 1$$
$$\text{s.t.} \quad 1 \leq x \leq 6$$

$$x^1 = \quad 1$$
$$UB = \quad 2$$
$$LB = \quad -3$$

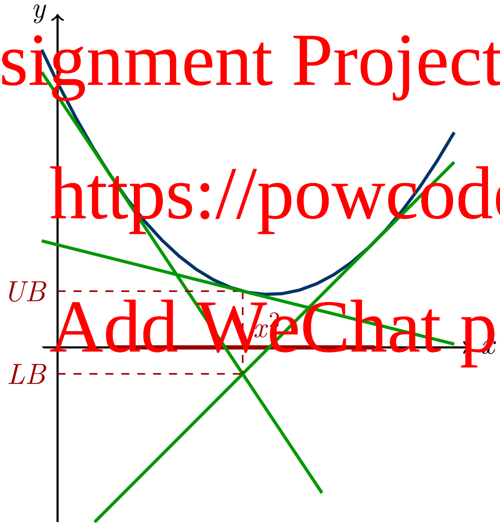
Lecture 5 callback: Kelley's algorithm



$$\min \quad \frac{(x-4)^2}{4} + 1$$
$$\text{s.t.} \quad 1 \le x \le 6$$

$x^2 = 3.5$
$UB = 1.0625$
$LB = -0.5$

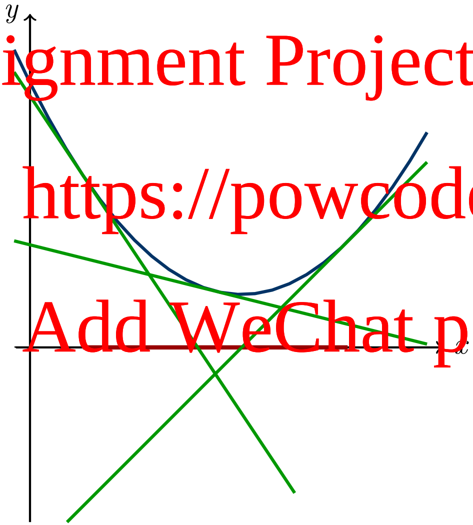
Lecture 5 callback: Kelley's algorithm



$$\min \quad \frac{(x-4)^2}{4} + 1$$
$$\text{s.t.} \quad 1 \le x \le 6$$

$$x^2 = 3.5$$
$$UB = 1.0625$$
$$LB = -0.5$$

Lecture 5 callback: Kelley's algorithm



$$\begin{aligned} \min \quad & \frac{(x-4)^2}{4} + 1 \\ \text{s.t.} \quad & 1 \leq x \leq 6 \end{aligned}$$

Also called

Outer approximation algorithm

Cutting plane algorithms in linear programming

- Consider a linear optimization problem

$$\begin{aligned} \min \quad & \mathbf{c}^\top \mathbf{x} \\ \text{s.t.} \quad & \mathbf{A}\mathbf{x} \geq \mathbf{b} \\ & \mathbf{x} \geq \mathbf{0}, \end{aligned}$$

  where $\mathbf{A}$ is $m \times n$, with $m \gg n$
- Cutting plane idea: select a subset of the rows of $\mathbf{A}$ and solve the problem with only these rows
  - Solution is always gonna be at least as good as the true optimum (fewer constraints are imposed)
  - But it could be infeasible

A generic LP cutting plane algorithm

Cutting plane algorithm for problems with too many constraints

1. Start with a set of constraints $\mathcal{I} \subset \{1, \ldots, m\}$
2. Solve the restricted problem and obtain an optimal primal and dual solution
3. Use the optimal primal solution to find the most violated constraint among those not included in the restricted problem
   - Either no constraint is violated and the current solution is optimal
   - Or a constraint is violated and we add it to the set $\mathcal{I}$, and go to step 2

Difference with Kelley's algorithm

- Kelley's algorithm approximates a convex nonlinear function as the maximum of linear functions
  - In particular, the solution is always feasible
- The cutting plane algorithm for LP iteratively restricts the feasible set by adding more and more constraints

Finding a violated constraint

- Consider all the constraints not included in the primal, $\mathbf{a}^\top \mathbf{x} \geq b_i$
- How to find out if one of them is violated?

Finding a violated constraint

- Consider all the constraints not included in the primal, $\mathbf{a}^{i\top}\mathbf{x} \geq b_i$
- How to find out if one of them is violated?

Solve an optimization problem!

$$\min_{i \notin \mathcal{I}} \mathbf{a}^{i\top}\mathbf{x}^* - b_i$$

- If this subproblem is easy, then a cutting plane algorithm is a good idea
- If it is hard, then it is probably not a good idea

Relationship between column generation and cutting planes

- A column generation algorithm for the primal problem is equivalent to a cutting plane algorithm for the dual

- To see why, recall:
  - Column generation is for too many variables
  - Cutting planes is for too many constraints
  - Variables in the primal $\Leftrightarrow$ constraints in the dual

- Furthermore, the column generation is finding a column with negative reduced cost $\Leftrightarrow$ finding a violated constraint in the dual!