CS498: Algorithmic Engineering
Lecture 1

Chandra Chekuri & Elfarouk Harb

University of Illinois Urbana-Champaign

01/20/2026

Outline

1. Course Logistics
   - Differences from CS374 and CS473
   - Content and Types of Projects in Class
   - Prerequisites
   - Grading
   - LLM Usage Policy
2. History of Linear Programming
3. Linear Programming: The Basics
4. The Engineer's Diet Dilemma
5. Interpreting and Debugging Gurobi Output

From Proofs to Solvers

From Proofs to Solvers

While standard algorithms courses focus on proving what is *computable*,
this course focuses on implementing what is *necessary*.

Relation to CS 374

CS 374: The Vocabulary

Relation to CS 374

CS 374: The Vocabulary

- Focuses on **Reductions**: Transforming Problem A to B to prove either tractability (e.g. problem is *P*) or hardness.

Relation to CS 374

CS 374: The Vocabulary

- Focuses on **Reductions**: Transforming Problem A to B to prove either tractability (e.g. problem is *P*) or hardness.
- **Goal:** Determine theoretical tractability (P vs NP).

Relation to CS 374

CS 374: The Vocabulary

- Focuses on **Reductions**: Transforming Problem A to B to prove either tractability (e.g. problem is *P*) or hardness.
- **Goal:** Determine theoretical tractability (P vs NP).
- **Output:** A formal proof.

Relation to CS 374

CS 374: The Vocabulary

- Focuses on **Reductions**: Transforming Problem A to B to prove either tractability (e.g. problem is P) or hardness.
- **Goal:** Determine theoretical tractability (P vs NP).
- **Output:** A formal proof.
- **Example Concept:**
  `3-SAT is NP-Complete, No Polynomial time algorithms, unless` $P = NP$

Relation to CS 374

CS 374: The Vocabulary

- Focuses on **Reductions**: Transforming Problem A to B to prove either tractability (e.g. problem is P) or hardness.
- **Goal:** Determine theoretical tractability (P vs NP).
- **Output:** A formal proof.
- **Example Concept:** `3-SAT is NP-Complete, No Polynomial time algorithms, unless` $P = NP$

CS 498: The Application

- Focuses on **Modelling**: Transforming Problem A to B so a solver for B can handle it.

Relation to CS 374

CS 374: The Vocabulary

- Focuses on **Reductions**: Transforming Problem A to B to prove either tractability (e.g. problem is P) or hardness.
- **Goal:** Determine theoretical tractability (P vs NP).
- **Output:** A formal proof.
- **Example Concept:** `3-SAT is NP-Complete, No Polynomial time algorithms, unless` $P = NP$

CS 498: The Application

- Focuses on **Modelling**: Transforming Problem A to B so a solver for B can handle it.
- **Goal:** Practical solutions for real-world instances.

Relation to CS 374

CS 374: The Vocabulary

- Focuses on **Reductions**: Transforming Problem A to B to prove either tractability (e.g. problem is P) or hardness.
- **Goal:** Determine theoretical tractability (P vs NP).
- **Output:** A formal proof.
- **Example Concept:** `3-SAT is NP-Complete, No Polynomial time algorithms, unless` $P = NP$

CS 498: The Application

- Focuses on **Modelling**: Transforming Problem A to B so a solver for B can handle it.
- **Goal:** Practical solutions for real-world instances.
- **Output:** A Python script or implementation of an Algorithm to solve the problem.

Relation to CS 374

CS 374: The Vocabulary

- Focuses on **Reductions**: Transforming Problem A to B to prove either tractability (e.g. problem is P) or hardness.
- **Goal:** Determine theoretical tractability (P vs NP).
- **Output:** A formal proof.
- **Example Concept:** `3-SAT is NP-Complete, No Polynomial time algorithms, unless` $P = NP$

CS 498: The Application

- Focuses on **Modelling**: Transforming Problem A to B so a solver for B can handle it.
- **Goal:** Practical solutions for real-world instances.
- **Output:** A Python script or implementation of an Algorithm to solve the problem.
- **Example Concept:** `How to Solve This 3-SAT Instance (1M vars) in < 5s`

Relation to CS 473

- **CS 473** analyzes the *internal mathematics of the engine*.
- Advanced algorithmic techniques (example: randomization, flow, advanced dynamic programming).
- Focus on proving efficiency (run-time) and approximation gurantees (bounds).

Relation to CS 473

- **CS 473** analyzes the *internal mathematics of the engine*.
- Advanced algorithmic techniques (example: randomization, flow, advanced dynamic programming).
- Focus on proving efficiency (run-time) and approximation gurantees (bounds).

- **CS 498** teaches you how to *drive the car*.
- We treat powerful solvers, that researchers have spent decades working on, as black boxes to be mastered.
- Focus on modeling complex constraints rather than implementing the solver itself.
- We still explain the theory behind the solvers, but the focus is on basics of theory

The “NP-Hard” Perspective

CS 374: ”Stop”

Proving a problem is NP-Hard is the end of the conversation. It means an **efficient worst-case algorithm does not exist**.

The “NP-Hard” Perspective

CS 374: ”Stop”

Proving a problem is NP-Hard is the end of the conversation. It means an **efficient worst-case algorithm does not exist**.

CS 473: ”Detour”

Accept that exact **provable** solutions are impossible. Pivot to designing algorithms that provide **guaranteed** *approximations*.

The “NP-Hard” Perspective

CS 374: ”Stop”

Proving a problem is NP-Hard is the end of the conversation. It means an **efficient worst-case algorithm does not exist**.

CS 473: ”Detour”

Accept that exact **provable** solutions are impossible. Pivot to designing algorithms that provide **guaranteed** *approximations*.

CS 498: ”Launch”

NP-Hardness is a worst-case warning, not a law of physics. Use SAT/SMT solvers to **crush real-world instances**. No more Grantees.

Modern Tooling Stack

We move beyond "pseudocode" to industrial-grade Python libraries used in Operations Research and Deep Learning.

- **Optimization:**
  `Gurobi`, `Pyomo` (Linear & Integer Programming)
- **Logic & Verification:**
  `Z3`, `PySAT` (SMT & SAT Solvers)
- **Differentiation:**
  `PyTorch` (Autograd & Neural Networks)

Course Comparison Matrix

Feature	CS 374 / 473	CS 498
Primary Goal	Proofs & Analysis	Models & Implementations
Hardness	Prove it's impossible in worst case	Use solvers to solve your instance anyway
Key Tools	Pencil, Paper, LaTeX	Gurobi, Z3, PyTorch, ...
Style	Purely Theoretical	Hybrid (Basics of Theory + Implementation)

Content and Types of Projects in Class

Part I: Discrete Optimization

Content and Types of Projects in Class

Part I: Discrete Optimization

- Linear & Integer Programming

Content and Types of Projects in Class

Part I: Discrete Optimization

- Linear & Integer Programming
- Modeling with `Gurobi`

Content and Types of Projects in Class

Part I: Discrete Optimization

- Linear & Integer Programming
- Modeling with `Gurobi`
- Supply chain & Network models

Content and Types of Projects in Class

Part I: Discrete Optimization

- Linear & Integer Programming
- Modeling with `Gurobi`
- Supply chain & Network models

Part II: Differentiable Systems

Content and Types of Projects in Class

Part I: Discrete Optimization

- Linear & Integer Programming
- Modeling with `Gurobi`
- Supply chain & Network models

Part II: Differentiable Systems

- First order and Second Order Optimization Techniques

Content and Types of Projects in Class

Part I: Discrete Optimization

- Linear & Integer Programming
- Modeling with `Gurobi`
- Supply chain & Network models

Part II: Differentiable Systems

- First order and Second Order Optimization Techniques
- Genetic Algorithms and Metaheuristics.

Content and Types of Projects in Class

Part I: Discrete Optimization

- Linear & Integer Programming
- Modeling with `Gurobi`
- Supply chain & Network models

Part II: Differentiable Systems

- First order and Second Order Optimization Techniques
- Genetic Algorithms and Metaheuristics.
- `PyTorch` & Autograd internals

Content and Types of Projects in Class

Part I: Discrete Optimization

- Linear & Integer Programming
- Modeling with `Gurobi`
- Supply chain & Network models

Part II: Differentiable Systems

- First order and Second Order Optimization Techniques
- Genetic Algorithms and Metaheuristics.
- `PyTorch` & Autograd internals
- Convex and Non-convex optimization

Content and Types of Projects in Class

Part I: Discrete Optimization

- Linear & Integer Programming
- Modeling with `Gurobi`
- Supply chain & Network models

Part II: Differentiable Systems

- First order and Second Order Optimization Techniques
- Genetic Algorithms and Metaheuristics.
- `PyTorch` & Autograd internals
- Convex and Non-convex optimization

Part III: Formal Methods

Content and Types of Projects in Class

Part I: Discrete Optimization

- Linear & Integer Programming
- Modeling with `Gurobi`
- Supply chain & Network models

Part II: Differentiable Systems

- First order and Second Order Optimization Techniques
- Genetic Algorithms and Metaheuristics.
- `PyTorch` & Autograd internals
- Convex and Non-convex optimization

Part III: Formal Methods

- SAT & SMT Solvers (`Z3`, `PySAT`)

Content and Types of Projects in Class

Part I: Discrete Optimization

- Linear & Integer Programming
- Modeling with `Gurobi`
- Supply chain & Network models

Part II: Differentiable Systems

- First order and Second Order Optimization Techniques
- Genetic Algorithms and Metaheuristics.
- `PyTorch` & Autograd internals
- Convex and Non-convex optimization

Part III: Formal Methods

- SAT & SMT Solvers (`Z3`, `PySAT`)
- Logic encodings (Sudoku, Scheduling)

Content and Types of Projects in Class

Part I: Discrete Optimization

- Linear & Integer Programming
- Modeling with `Gurobi`
- Supply chain & Network models

Part II: Differentiable Systems

- First order and Second Order Optimization Techniques
- Genetic Algorithms and Metaheuristics.
- `PyTorch` & Autograd internals
- Convex and Non-convex optimization

Part III: Formal Methods

- SAT & SMT Solvers (`Z3`, `PySAT`)
- Logic encodings (Sudoku, Scheduling)
- Automated verification and Solving Puzzles with SAT/SMT Solvers

Content and Types of Projects in Class

Part I: Discrete Optimization

- Linear & Integer Programming
- Modeling with `Gurobi`
- Supply chain & Network models

Part II: Differentiable Systems

- First order and Second Order Optimization Techniques
- Genetic Algorithms and Metaheuristics.
- `PyTorch` & Autograd internals
- Convex and Non-convex optimization

Part III: Formal Methods

- SAT & SMT Solvers (`Z3`, `PySAT`)
- Logic encodings (Sudoku, Scheduling)
- Automated verification and Solving Puzzles with SAT/SMT Solvers

Parts IV & V: Synthesis

Content and Types of Projects in Class

Part I: Discrete Optimization

- Linear & Integer Programming
- Modeling with `Gurobi`
- Supply chain & Network models

Part II: Differentiable Systems

- First order and Second Order Optimization Techniques
- Genetic Algorithms and Metaheuristics.
- `PyTorch` & Autograd internals
- Convex and Non-convex optimization

Part III: Formal Methods

- SAT & SMT Solvers (`Z3`, `PySAT`)
- Logic encodings (Sudoku, Scheduling)
- Automated verification and Solving Puzzles with SAT/SMT Solvers

Parts IV & V: Synthesis

- Data-Driven Optimization (aka Data Science)

Content and Types of Projects in Class

Part I: Discrete Optimization

- Linear & Integer Programming
- Modeling with `Gurobi`
- Supply chain & Network models

Part II: Differentiable Systems

- First order and Second Order Optimization Techniques
- Genetic Algorithms and Metaheuristics.
- `PyTorch` & Autograd internals
- Convex and Non-convex optimization

Part III: Formal Methods

- SAT & SMT Solvers (`Z3`, `PySAT`)
- Logic encodings (Sudoku, Scheduling)
- Automated verification and Solving Puzzles with SAT/SMT Solvers

Parts IV & V: Synthesis

- Data-Driven Optimization (aka Data Science)
- LLMs as Reasoning Engines, prompting, consistency, etc.

Content and Types of Projects in Class

Part I: Discrete Optimization

- Linear & Integer Programming
- Modeling with `Gurobi`
- Supply chain & Network models

Part II: Differentiable Systems

- First order and Second Order Optimization Techniques
- Genetic Algorithms and Metaheuristics.
- `PyTorch` & Autograd internals
- Convex and Non-convex optimization

Part III: Formal Methods

- SAT & SMT Solvers (`Z3`, `PySAT`)
- Logic encodings (Sudoku, Scheduling)
- Automated verification and Solving Puzzles with SAT/SMT Solvers

Parts IV & V: Synthesis

- Data-Driven Optimization (aka Data Science)
- LLMs as Reasoning Engines, prompting, consistency, etc.
- Program Synthesis.

Projects I: Optimization & Logic

1. The "Fleetline" Challenge (Week 3)

Projects I: Optimization & Logic

1. The "Fleetline" Challenge (Week 3)

- A real problem from a YC startup (Who are giving us a guest lecture).
- Problem: You must assign truck drivers to loads to maximize revenue while respecting complex human constraints (e.g., *"Driver A must be back in Chicago by Friday for their daughter's rehearsal"*).

Projects I: Optimization & Logic

1. **The "Fleetline" Challenge (Week 3)**
   - A real problem from a YC startup (Who are giving us a guest lecture).
   - Problem: You must assign truck drivers to loads to maximize revenue while respecting complex human constraints (e.g., *"Driver A must be back in Chicago by Friday for their daughter's rehearsal"*).
2. **SMT for Scheduling (Week 9)**

Projects I: Optimization & Logic

1. **The "Fleetline" Challenge (Week 3)**
   - A real problem from a YC startup (Who are giving us a guest lecture).
   - Problem: You must assign truck drivers to loads to maximize revenue while respecting complex human constraints (e.g., *"Driver A must be back in Chicago by Friday for their daughter's rehearsal"*).

2. **SMT for Scheduling (Week 9)**
   - Solve very complex Puzzles beyond most humans reach using SMT solvers.

Projects I: Optimization & Logic

1. The "Fleetline" Challenge (Week 3)

- A real problem from a YC startup (Who are giving us a guest lecture).
- Problem: You must assign truck drivers to loads to maximize revenue while respecting complex human constraints (e.g., *"Driver A must be back in Chicago by Friday for their daughter's rehearsal"*).

2. SMT for Scheduling (Week 9)

- Solve very complex Puzzles beyond most humans reach using SMT solvers.
- Learn to encode massive real-world scheduling conflicts into SMT Solvers (`Z3`).

Projects I: Optimization & Logic

1. The “Fleetline” Challenge (Week 3)

- A real problem from a YC startup (Who are giving us a guest lecture).
- Problem: You must assign truck drivers to loads to maximize revenue while respecting complex human constraints (e.g., *“Driver A must be back in Chicago by Friday for their daughter’s rehearsal”*).

2. SMT for Scheduling (Week 9)

- Solve very complex Puzzles beyond most humans reach using SMT solvers.
- Learn to encode massive real-world scheduling conflicts into SMT Solvers (`Z3`).

Projects II: AI & Neurosymbolic Agents

3. Evolution & Gradients (Week 6-7)

Projects II: AI & Neurosymbolic Agents

3. Evolution & Gradients (Week 6-7)

- Solve NP-Hard problems (like TSP) using Genetic Algorithms and Metaheuristics.

Projects II: AI & Neurosymbolic Agents

3. Evolution & Gradients (Week 6-7)

- Solve NP-Hard problems (like TSP) using Genetic Algorithms and Metaheuristics.
- Look under the hood of Deep Learning by building your own Autodiff engine from scratch before training real neural-nets in `PyTorch`.

Projects II: AI & Neurosymbolic Agents

3. Evolution & Gradients (Week 6-7)

- Solve NP-Hard problems (like TSP) using Genetic Algorithms and Metaheuristics.
- Look under the hood of Deep Learning by building your own Autodiff engine from scratch before training real neural-nets in `PyTorch`.

4. Data-Driven Pipelines (Week 11)

Projects II: AI & Neurosymbolic Agents

3. Evolution & Gradients (Week 6-7)

- Solve NP-Hard problems (like TSP) using Genetic Algorithms and Metaheuristics.
- Look under the hood of Deep Learning by building your own Autodiff engine from scratch before training real neural-nets in `PyTorch`.

4. Data-Driven Pipelines (Week 11)

Build end-to-end ML pipelines (feature engineering, regression) to predict real-world parameters (e.g., housing prices) and integrate them directly into optimization objectives.

Projects II: AI & Neurosymbolic Agents

3. Evolution & Gradients (Week 6-7)

- Solve NP-Hard problems (like TSP) using Genetic Algorithms and Metaheuristics.
- Look under the hood of Deep Learning by building your own Autodiff engine from scratch before training real neural-nets in `PyTorch`.

4. Data-Driven Pipelines (Week 11)

Build end-to-end ML pipelines (feature engineering, regression) to predict real-world parameters (e.g., housing prices) and integrate them directly into optimization objectives.

5. The AIME Agent (Week 14)

Projects II: AI & Neurosymbolic Agents

3. Evolution & Gradients (Week 6-7)

- Solve NP-Hard problems (like TSP) using Genetic Algorithms and Metaheuristics.
- Look under the hood of Deep Learning by building your own Autodiff engine from scratch before training real neural-nets in `PyTorch`.

4. Data-Driven Pipelines (Week 11)

Build end-to-end ML pipelines (feature engineering, regression) to predict real-world parameters (e.g., housing prices) and integrate them directly into optimization objectives.

5. The AIME Agent (Week 14)

Build a neurosymbolic reasoning agent to solve **Math Olympiad (AIME)** problems. You will engineer prompts, implement self-consistency checks, and use open-source LLMs to tackle high-level reasoning tasks.

Projects II: AI & Neurosymbolic Agents

3. Evolution & Gradients (Week 6-7)

- Solve NP-Hard problems (like TSP) using Genetic Algorithms and Metaheuristics.
- Look under the hood of Deep Learning by building your own Autodiff engine from scratch before training real neural-nets in `PyTorch`.

4. Data-Driven Pipelines (Week 11)

Build end-to-end ML pipelines (feature engineering, regression) to predict real-world parameters (e.g., housing prices) and integrate them directly into optimization objectives.

5. The AIME Agent (Week 14)

Build a neurosymbolic reasoning agent to solve **Math Olympiad (AIME)** problems. You will engineer prompts, implement self-consistency checks, and use open-source LLMs to tackle high-level reasoning tasks.

Prerequisites

1. Theory

- **CS 374** is assumed.
- We won't reteach NP-Hardness; we assume you know what it implies.

Prerequisites

1. Theory

- **CS 374** is assumed.
- We won't reteach NP-Hardness; we assume you know what it implies.

2. Attitude

- **Coding Heavy**: This is an engineering class.
- **Resilience**: You must be willing to read documentation, debug strange library errors, and explore new tools.

Prerequisites

1. Theory

- **CS 374** is assumed.
- We won't reteach NP-Hardness; we assume you know what it implies.

2. Attitude

- **Coding Heavy**: This is an engineering class.
- **Resilience**: You must be willing to read documentation, debug strange library errors, and explore new tools.

3. Coding

Python Literacy Check

```python
import numpy as np

A = np.array([[1, 2], [3, 4]])
b = np.array([5, 6])

# If you know what this does

x = np.linalg.solve(A, b)

# or can look it up quick
# ...you're Gucci.
```

Grading Structure

60%
Weekly Homeworks

- **Groups of 2-4**.
- *The more, the merrier.*
- High volume of problems; working alone is a competitive disadvantage.

Grading Structure

60%	10%
Weekly Homeworks	**3 ”Pulse Checks”**

- **Groups of 2-4**.
- *The more, the merrier.*
- High volume of problems; working alone is a competitive disadvantage.

- After Parts I, II, III.
- In-class, short, individual quizzes.
- **Goal**: Check if you are alive.
- If you understand the bare minimum, you get 100%.

Grading Structure

60%	10%	30%
Weekly Homeworks	**3 ”Pulse Checks”**	**Individual Final Project**

- **Groups of 2-4**.
- *The more, the merrier.*
- High volume of problems; working alone is a competitive disadvantage.

- After Parts I, II, III.
- In-class, short, individual quizzes.
- **Goal**: Check if you are alive.
- If you understand the bare minimum, you get 100%.

- **Algorithmic Engineering**.
- Build a system, implement a paper, or optimize a complex pipeline.
- Compare performance (speed/quality).

1. Course Logistics
   - Differences from CS374 and CS473
   - Content and Types of Projects in Class
   - Prerequisites
   - Grading
   - LLM Usage Policy
2. History of Linear Programming
3. Linear Programming: The Basics
4. The Engineer's Diet Dilemma
5. Interpreting and Debugging Gurobi Output

LLM Usage Policy: “Productivity, not Replacement”

The Rule:

- You **CAN** use LLMs (ChatGPT, Gemini, Claude, Copilot).

LLM Usage Policy: “Productivity, not Replacement”

The Rule:

- You **CAN** use LLMs (ChatGPT, Gemini, Claude, Copilot).
- You **MUST** acknowledge usage and explain exactly what you asked the LLM to do.

LLM Usage Policy: “Productivity, not Replacement”

The Rule:

- You **CAN** use LLMs (ChatGPT, Gemini, Claude, Copilot).
- You **MUST** acknowledge usage and explain exactly what you asked the LLM to do.

The “Random Audit”:

- Each week, random students will be asked to **explain their code/solutions** in person.

LLM Usage Policy: “Productivity, not Replacement”

The Rule:

- You **CAN** use LLMs (ChatGPT, Gemini, Claude, Copilot).
- You **MUST** acknowledge usage and explain exactly what you asked the LLM to do.

The “Random Audit”:

- Each week, random students will be asked to **explain their code/solutions** in person.
- If you blind-copied without understanding $\rightarrow$ **Big Problems.**

LLM Usage Policy: “Productivity, not Replacement”

Kosher vs. Not Kosher

✓**Good:**

LLM Usage Policy: “Productivity, not Replacement”

Kosher vs. Not Kosher

✓**Good:**

- “Write a Python function to parse this DIMACS file format.”

LLM Usage Policy: “Productivity, not Replacement”

Kosher vs. Not Kosher

✓**Good:**

- “Write a Python function to parse this DIMACS file format.”
- “Here is an Algorithm to solve this problem ... Encode the constraint this way ... Please implement my idea in Python.”

LLM Usage Policy: “Productivity, not Replacement”

Kosher vs. Not Kosher

✓**Good:**

- “Write a Python function to parse this DIMACS file format.”
- “Here is an Algorithm to solve this problem ... Encode the constraint this way ... Please implement my idea in Python.”

X Bad:

“Here is the PDF of the homework, solve Problem 3 for me.”

Questions?

Ready to build?

> _

The Toy Factory Example

Scenario: You build two products: **Widgets** (x_1) and **Gadgets** (x_2).

The Toy Factory Example

Scenario: You build two products: **Widgets** (x_1) and **Gadgets** (x_2).
Profits:

- Widget: \$3 profit
- Gadget: \$4 profit

The Toy Factory Example

Scenario: You build two products: **Widgets** (x_1) and **Gadgets** (x_2).

Profits:

- Widget: \$3 profit
- Gadget: \$4 profit

Constraints:

- **Metal:** Have 10kg. Widget uses 1, Gadget uses 2.

The Toy Factory Example

Scenario: You build two products: **Widgets** (x_1) and **Gadgets** (x_2).

Profits:

- Widget: \$3 profit
- Gadget: \$4 profit

Constraints:

- **Metal:** Have 10kg. Widget uses 1, Gadget uses 2.
- **Wood:** Have 15kg. Widget uses 2, Gadget uses 1.

The Toy Factory Example

Scenario: You build two products: **Widgets** (x_1) and **Gadgets** (x_2).

Profits:

- Widget: \$3 profit
- Gadget: \$4 profit

Constraints:

- **Metal:** Have 10kg. Widget uses 1, Gadget uses 2.
- **Wood:** Have 15kg. Widget uses 2, Gadget uses 1.

The LP Model:

$$\max \quad 3x_1 + 4x_2$$

The Toy Factory Example

Scenario: You build two products: **Widgets** (x_1) and **Gadgets** (x_2).

Profits:

- Widget: \$3 profit
- Gadget: \$4 profit

Constraints:

- **Metal:** Have 10kg. Widget uses 1, Gadget uses 2.
- **Wood:** Have 15kg. Widget uses 2, Gadget uses 1.

The LP Model:

$$\begin{aligned} \max \quad & 3x_1 + 4x_2 \\ \text{s.t.} \quad & 1x_1 + 2x_2 \leq 10 \end{aligned}$$

The Toy Factory Example

Scenario: You build two products: **Widgets** (x_1) and **Gadgets** (x_2).

Profits:

- Widget: \$3 profit
- Gadget: \$4 profit

Constraints:

- **Metal:** Have 10kg. Widget uses 1, Gadget uses 2.
- **Wood:** Have 15kg. Widget uses 2, Gadget uses 1.

The LP Model:

$$\begin{aligned} \max \quad & 3x_1 + 4x_2 \\ \text{s.t.} \quad & 1x_1 + 2x_2 \leq 10 \\ & 2x_1 + 1x_2 \leq 15 \end{aligned}$$

The Toy Factory Example

Scenario: You build two products: **Widgets** (x_1) and **Gadgets** (x_2).

Profits:

- Widget: \$3 profit
- Gadget: \$4 profit

Constraints:

- **Metal:** Have 10kg. Widget uses 1, Gadget uses 2.
- **Wood:** Have 15kg. Widget uses 2, Gadget uses 1.

The LP Model:

$$\begin{aligned} \max \quad & 3x_1 + 4x_2 \\ \text{s.t.} \quad & 1x_1 + 2x_2 \leq 10 \\ & 2x_1 + 1x_2 \leq 15 \\ & x_1, x_2 \geq 0 \end{aligned}$$

Act I: The Dark Ages (Pre-1947)

Before 1947, the idea of writing a massive planning problem as a single mathematical equation was unknown.

Act I: The Dark Ages (Pre-1947)

Before 1947, the idea of writing a massive planning problem as a single mathematical equation was unknown.

- **Fourier (1823):** Solved small systems of inequalities.

Act I: The Dark Ages (Pre-1947)

Before 1947, the idea of writing a massive planning problem as a single mathematical equation was unknown.

- **Fourier (1823):** Solved small systems of inequalities.
- **Leonid Kantorovich (1939):** Invented LP in the USSR to optimize plywood production.

Act I: The Dark Ages (Pre-1947)

Before 1947, the idea of writing a massive planning problem as a single mathematical equation was unknown.

- **Fourier (1823):** Solved small systems of inequalities.
- **Leonid Kantorovich (1939):** Invented LP in the USSR to optimize plywood production.
- **The Tragedy:** The Soviet government ignored him. His work remained unknown to the West for decades.

Act I: The Dark Ages (Pre-1947)

Before 1947, the idea of writing a massive planning problem as a single mathematical equation was unknown.

- **Fourier (1823):** Solved small systems of inequalities.
- **Leonid Kantorovich (1939):** Invented LP in the USSR to optimize plywood production.
- **The Tragedy:** The Soviet government ignored him. His work remained unknown to the West for decades.

Motzkin's Thesis (1936)

Listed only **42 papers** in all of history on linear inequalities. Today, there are tens of thousands per year.

Act II: WWII & George Dantzig

The Setup:

- George Dantzig spent WWII planning US Air Force logistics by hand.

Act II: WWII & George Dantzig

The Setup:

- George Dantzig spent WWII planning US Air Force logistics by hand.
- 1946: The Air Force asks: “**Can you mechanize the planning process?**”

Act II: WWII & George Dantzig

The Setup:

- George Dantzig spent WWII planning US Air Force logistics by hand.
- 1946: The Air Force asks: “**Can you mechanize the planning process?**”
- He built a dynamic model of resources and activities, but something was missing.

Act II: WWII & George Dantzig

The Setup:

- George Dantzig spent WWII planning US Air Force logistics by hand.
- 1946: The Air Force asks: “**Can you mechanize the planning process?**”
- He built a dynamic model of resources and activities, but something was missing.
- Dantzig realized he needed an **Explicit Objective Function** to optimize on top of his linear constraints.

Act II: WWII & George Dantzig

The Setup:

- George Dantzig spent WWII planning US Air Force logistics by hand.
- 1946: The Air Force asks: "**Can you mechanize the planning process?**"
- He built a dynamic model of resources and activities, but something was missing.
- Dantzig realized he needed an **Explicit Objective Function** to optimize on top of his linear constraints.
- But how to solve a system with thousands of linear constraints and linear objective? He needed help.

Act III: Meeting Von Neumann (Oct 1947)

Dantzig visits **John von Neumann** at Princeton.

Act III: Meeting Von Neumann (Oct 1947)

Dantzig visits **John von Neumann** at Princeton.

1. Dantzig starts explaining his Air Force model in tedious detail.

Act III: Meeting Von Neumann (Oct 1947)

Dantzig visits **John von Neumann** at Princeton.

1. Dantzig starts explaining his Air Force model in tedious detail.
2. Von Neumann cuts him off: “**Get to the point.**”

Act III: Meeting Von Neumann (Oct 1947)

Dantzig visits **John von Neumann** at Princeton.

1. Dantzig starts explaining his Air Force model in tedious detail.
2. Von Neumann cuts him off: “**Get to the point.**”
3. Dantzig writes the linear programming problem on the board.

Act III: Meeting Von Neumann (Oct 1947)

Dantzig visits **John von Neumann** at Princeton.

1. Dantzig starts explaining his Air Force model in tedious detail.
2. Von Neumann cuts him off: “**Get to the point.**”
3. Dantzig writes the linear programming problem on the board.

The Revelation

Von Neumann stands up: “**Oh—that!**”
He proceeds to lecture Dantzig for 90 minutes on **Duality** and **Geometry**.
Von Neumann had already derived the theory of LP while inventing Game Theory.

Act IV: The Mic Drop

Conference, 1948: Dantzig presents LP to a room of heavyweights.
Harold Hotelling (Economics Giant) stands up:

> *"But we all know the world is non-linear."*

Act IV: The Mic Drop

Conference, 1948: Dantzig presents LP to a room of heavyweights.
Harold Hotelling (Economics Giant) stands up:

> *"But we all know the world is non-linear."*

Dantzig freezes. The room goes silent. Then Von Neumann raises his hand:

Act IV: The Mic Drop

Conference, 1948: Dantzig presents LP to a room of heavyweights.
Harold Hotelling (Economics Giant) stands up:

"But we all know the world is non-linear."

Dantzig freezes. The room goes silent. Then Von Neumann raises his hand:

***"If the axioms of linear programming fit your problem, use it.
If not, don't."***

He sat down. The field of Linear Programming was born.

For more historical readings, read "REMINISCENCES ABOUT THE ORIGINS OF LINEAR PROGRAMMING" by Dantzig himself!

The Toy Factory Example

Scenario: You build two products: **Widgets** (x_1) and **Gadgets** (x_2).

Profits:

- Widget: \$3 profit
- Gadget: \$4 profit

Constraints:

- **Metal:** Have 10kg. Widget uses 1, Gadget uses 2.
- **Wood:** Have 15kg. Widget uses 2, Gadget uses 1.

The LP Model:

$$\begin{aligned} \max \quad & 3x_1 + 4x_2 \\ \text{s.t.} \quad & 1x_1 + 2x_2 \leq 10 \\ & 2x_1 + 1x_2 \leq 15 \\ & x_1, x_2 \geq 0 \end{aligned}$$

The Canonical Form

Every LP can be written in Matrix Notation: $\max \mathbf{c}^T\mathbf{x}$ s.t. $\mathbf{A}\mathbf{x} \leq \mathbf{b}$.

The Canonical Form

Every LP can be written in Matrix Notation: $\max \mathbf{c}^T\mathbf{x}$ s.t. $\mathbf{A}\mathbf{x} \leq \mathbf{b}$.

For our Factory:

$$\underbrace{\begin{bmatrix} 3 \\ 4 \end{bmatrix}^T}_{\mathbf{c}^T} \underbrace{\begin{bmatrix} x_1 \\ x_2 \end{bmatrix}}_{\mathbf{x}}$$

The Canonical Form

Every LP can be written in Matrix Notation: $\max \mathbf{c}^T\mathbf{x}$ s.t. $\mathbf{A}\mathbf{x} \leq \mathbf{b}$.

For our Factory:

$$\underbrace{\begin{bmatrix} 3 \\ 4 \end{bmatrix}^T}_{\mathbf{c}^T} \underbrace{\begin{bmatrix} x_1 \\ x_2 \end{bmatrix}}_{\mathbf{x}} \quad \text{subject to}$$

The Canonical Form

Every LP can be written in Matrix Notation: $\max \mathbf{c}^T\mathbf{x}$ s.t. $\mathbf{A}\mathbf{x} \le \mathbf{b}$.

For our Factory:

$$\underbrace{\begin{bmatrix}3\\4\end{bmatrix}^T}_{\mathbf{c}^T}\underbrace{\begin{bmatrix}x_1\\x_2\end{bmatrix}}_{\mathbf{x}} \quad \text{subject to} \quad \underbrace{\begin{bmatrix}1 & 2\\2 & 1\\-1 & 0\\0 & -1\end{bmatrix}}_{\mathbf{A}}\underbrace{\begin{bmatrix}x_1\\x_2\end{bmatrix}}_{\mathbf{x}} \le \underbrace{\begin{bmatrix}10\\15\\0\\0\end{bmatrix}}_{\mathbf{b}}$$

The Canonical Form

Every LP can be written in Matrix Notation: $\max \mathbf{c}^T\mathbf{x}$ s.t. $\mathbf{A}\mathbf{x} \leq \mathbf{b}$.

For our Factory:

$$\underbrace{\begin{bmatrix}3\\4\end{bmatrix}^T}_{\mathbf{c}^T}\underbrace{\begin{bmatrix}x_1\\x_2\end{bmatrix}}_{\mathbf{x}} \quad \text{subject to} \quad \underbrace{\begin{bmatrix}1 & 2\\2 & 1\\-1 & 0\\0 & -1\end{bmatrix}}_{\mathbf{A}}\underbrace{\begin{bmatrix}x_1\\x_2\end{bmatrix}}_{\mathbf{x}} \leq \underbrace{\begin{bmatrix}10\\15\\0\\0\end{bmatrix}}_{\mathbf{b}}$$

- **x**: Decision Variables (The knobs we turn).
- **c**: Objective Coefficients (Profits/Costs).
- **A**: Constraint Matrix (Resource usage).
- **b**: Right-Hand Side (Capacities).

Pathologies: When things go wrong

Before we solve it, what if we *can't*?

Pathologies: When things go wrong

Before we solve it, what if we *can't*?

1. Infeasibility

No solution satisfies all constraints.

$$x \leq 2 \quad \text{AND} \quad x \geq 3$$

The feasible region is **Empty**.
Gurobi: `Model is infeasible.`

Pathologies: When things go wrong

Before we solve it, what if we *can't*?

1. Infeasibility

No solution satisfies all constraints.

$$x \leq 2 \quad \text{AND} \quad x \geq 3$$

The feasible region is **Empty**.
Gurobi: `Model is infeasible.`

2. Unboundedness

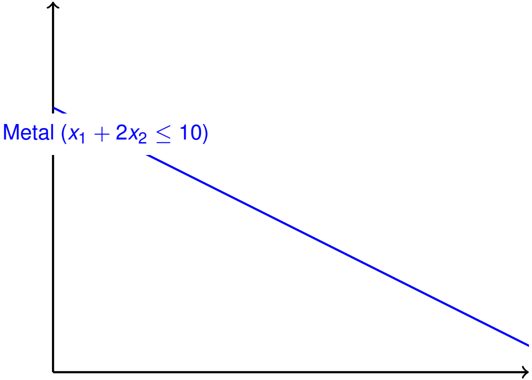
The region is open in the direction of improvement.

$$\max x \quad \text{s.t.} \quad x \geq 5$$

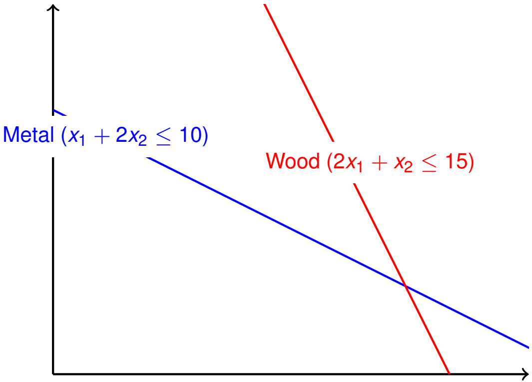
You can increase profit to ∞.
Gurobi: `Model is unbounded.`

Geometry: The Feasible Region

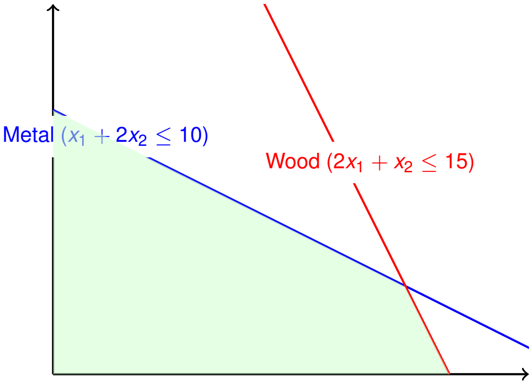
Geometry: The Feasible Region



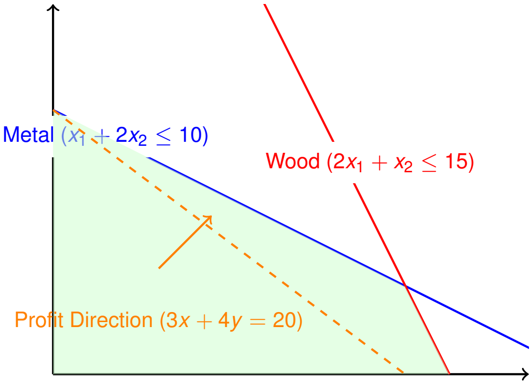
Geometry: The Feasible Region



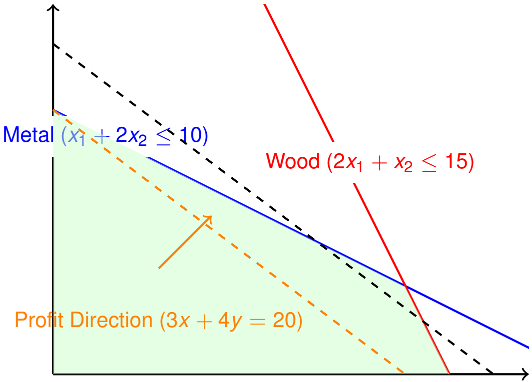
Geometry: The Feasible Region



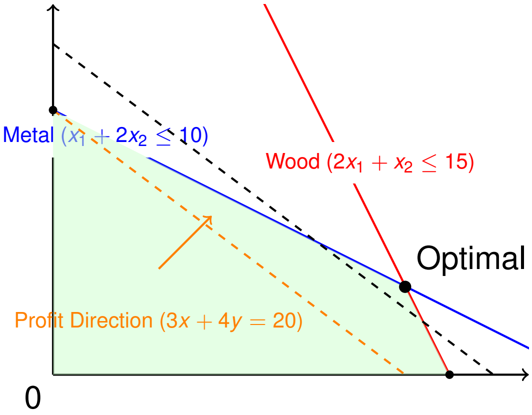
Geometry: The Feasible Region



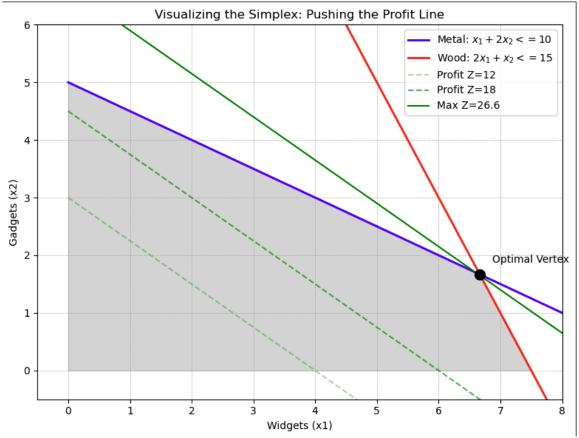
Geometry: The Feasible Region



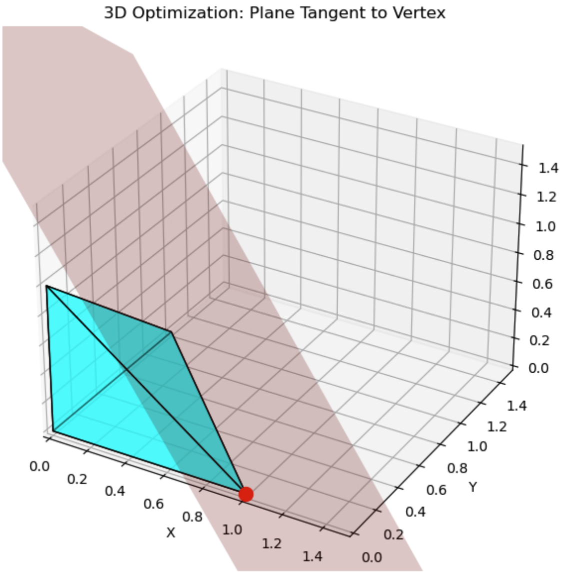
Geometry: The Feasible Region



2D


Visualizing the Simplex: Pushing the Profit Line
Metal: $x_1 + 2x_2 <= 10$
Wood: $2x_1 + x_2 <= 15$
Profit Z=12
Profit Z=18
Max Z=26.6
Optimal Vertex
Gadgets (x2)
Widgets (x1)
0
1
2
3
4
5
6
7
8

3D


3D Optimization: Plane Tangent to Vertex
X
Y
0.0
0.2
0.4
0.6
0.8
1.0
1.2
1.4

The Fundamental Theorem of LP

Theorem

If a Linear Program has an optimal solution, there exists a **Vertex** (corner point) of the feasible region that is optimal.

The Fundamental Theorem of LP

Theorem

If a Linear Program has an optimal solution, there exists a **Vertex** (corner point) of the feasible region that is optimal.

Proof Sketch (Convexity Argument):

1. Any point x in the polytope is a weighted average (convex combination) of the polytope's vertices $v_1, \ldots, v_k$: $x = \sum \alpha_i v_i$ with $\sum_i \alpha_i = 1, \alpha_i \geq 0$.

The Fundamental Theorem of LP

Theorem

If a Linear Program has an optimal solution, there exists a **Vertex** (corner point) of the feasible region that is optimal.

Proof Sketch (Convexity Argument):

1. Any point x in the polytope is a weighted average (convex combination) of the polytope's vertices $v_1, \ldots, v_k$: $x = \sum \alpha_i v_i$ with $\sum_i \alpha_i = 1, \alpha_i \geq 0$.
2. The objective $f(x) = c^T x$ is linear.

The Fundamental Theorem of LP

Theorem

If a Linear Program has an optimal solution, there exists a **Vertex** (corner point) of the feasible region that is optimal.

Proof Sketch (Convexity Argument):

1. Any point x in the polytope is a weighted average (convex combination) of the polytope's vertices $v_1, \ldots, v_k$: $x = \sum \alpha_i v_i$ with $\sum_i \alpha_i = 1, \alpha_i \geq 0$.
2. The objective $f(x) = c^T x$ is linear.
3. Linearity means $f(x) = f(\sum_i \alpha_i v_i) = \sum_i \alpha_i f(v_i)$.

The Fundamental Theorem of LP

Theorem

If a Linear Program has an optimal solution, there exists a **Vertex** (corner point) of the feasible region that is optimal.

Proof Sketch (Convexity Argument):

1. Any point x in the polytope is a weighted average (convex combination) of the polytope's vertices $v_1, \ldots, v_k$: $x = \sum \alpha_i v_i$ with $\sum_i \alpha_i = 1, \alpha_i \geq 0$.
2. The objective $f(x) = c^T x$ is linear.
3. Linearity means $f(x) = f(\sum_i \alpha_i v_i) = \sum_i \alpha_i f(v_i)$.
4. An average cannot be larger than the maximum of its components.

The Fundamental Theorem of LP

Theorem

If a Linear Program has an optimal solution, there exists a **Vertex** (corner point) of the feasible region that is optimal.

Proof Sketch (Convexity Argument):

1. Any point x in the polytope is a weighted average (convex combination) of the polytope's vertices $v_1, \ldots, v_k$: $x = \sum \alpha_i v_i$ with $\sum_i \alpha_i = 1, \alpha_i \geq 0$.
2. The objective $f(x) = c^T x$ is linear.
3. Linearity means $f(x) = f(\sum_i \alpha_i v_i) = \sum_i \alpha_i f(v_i)$.
4. An average cannot be larger than the maximum of its components.
5. Therefore, $f(x) \leq \max_i f(v_i)$. The max is at a corner!

From Math to Code (Gurobi)

We don't solve these LPs by hand. We assume the **Solver** is a black box.

From Math to Code (Gurobi)

We don't solve these LPs by hand. We assume the **Solver** is a black box.

```python
import gurobipy as gp
from gurobipy import GRB

m = gp.Model("factory")
```

From Math to Code (Gurobi)

We don't solve these LPs by hand. We assume the **Solver** is a black box.

```python
import gurobipy as gp
from gurobipy import GRB

m = gp.Model("factory")

# Variables
x1 = m.addVar(name="widgets")
x2 = m.addVar(name="gadgets")
```

From Math to Code (Gurobi)

We don't solve these LPs by hand. We assume the **Solver** is a black box.

```python
import gurobipy as gp
from gurobipy import GRB

m = gp.Model("factory")

# Variables
x1 = m.addVar(name="widgets")
x2 = m.addVar(name="gadgets")

# Objective
m.setObjective(3*x1 + 4*x2, GRB.MAXIMIZE)
```

From Math to Code (Gurobi)

We don't solve these LPs by hand. We assume the **Solver** is a black box.

```
import gurobipy as gp
from gurobipy import GRB

m = gp.Model("factory")


# Variables
x1 = m.addVar(name="widgets")
x2 = m.addVar(name="gadgets")

# Objective
m.setObjective(3*x1 + 4*x2, GRB.MAXIMIZE)

# Constraints
m.addConstr(1*x1 + 2*x2 <= 10, "metal")
m.addConstr(2*x1 + 1*x2 <= 15, "wood")
```

From Math to Code (Gurobi)

We don't solve these LPs by hand. We assume the **Solver** is a black box.

```
import gurobipy as gp
from gurobipy import GRB

m = gp.Model("factory")

# Variables
x1 = m.addVar(name="widgets")
x2 = m.addVar(name="gadgets")

# Objective
m.setObjective(3*x1 + 4*x2, GRB.MAXIMIZE)

# Constraints
m.addConstr(1*x1 + 2*x2 <= 10, "metal")
m.addConstr(2*x1 + 1*x2 <= 15, "wood")

m.optimize()
```

From Math to Code (Gurobi)

We don't solve these LPs by hand. We assume the **Solver** is a black box.

```python
import gurobipy as gp
from gurobipy import GRB

m = gp.Model("factory")

# Variables
x1 = m.addVar(name="widgets")
x2 = m.addVar(name="gadgets")
# Objective
m.setObjective(3*x1 + 4*x2, GRB.MAXIMIZE)
# Constraints
m.addConstr(1*x1 + 2*x2 <= 10, "metal")
m.addConstr(2*x1 + 1*x2 <= 15, "wood")
m.optimize()
print(x1.X, x2.X)
```

The Scenario

OptiMeal Inc. has a conflict:

- **Finance Team:** “Cut costs! Food is too expensive.”
- **Nutritionists:** “We need to meet daily health requirements.”

The Scenario

OptiMeal Inc. has a conflict:

- **Finance Team:** “Cut costs! Food is too expensive.”
- **Nutritionists:** “We need to meet daily health requirements.”

Your Mission:

- Use LP to design the cheapest daily meal plan.

The Scenario

OptiMeal Inc. has a conflict:

- **Finance Team:** "Cut costs! Food is too expensive."
- **Nutritionists:** "We need to meet daily health requirements."

Your Mission:

- Use LP to design the cheapest daily meal plan.
- You can eat fractional servings (e.g., 0.5 bananas).

The Scenario

OptiMeal Inc. has a conflict:

- **Finance Team:** “Cut costs! Food is too expensive.”
- **Nutritionists:** “We need to meet daily health requirements.”

Your Mission:

- Use LP to design the cheapest daily meal plan.
- You can eat fractional servings (e.g., 0.5 bananas).
- **Objective:** Min Cost.

The Scenario

OptiMeal Inc. has a conflict:

- **Finance Team:** “Cut costs! Food is too expensive.”
- **Nutritionists:** “We need to meet daily health requirements.”

Your Mission:

- Use LP to design the cheapest daily meal plan.
- You can eat fractional servings (e.g., 0.5 bananas).
- **Objective:** Min Cost.
- **Constraints:** Calorie floor, Protein floor, Sugar ceiling, etc.

The Data (Nutrition & Costs)

Food	Cost ($)	Cal	Prot (g)	Carb (g)	Sugar (g)	Fiber (g)	Fat (g)
Chicken	1.80	128	24.0	0.0	0.0	0.0	2.7
Banana	0.30	105	1.3	27.0	14.0	3.1	0.4
Yogurt	0.90	104	5.9	7.9	7.9	0.0	5.5
Beans	1.10	120	8.0	21.0	1.0	7.0	0.5
Spinach	0.40	7	0.9	1.1	0.1	0.7	0.1
Almonds	0.70	160	6.0	6.0	1.0	3.0	14.0

The Data (Nutrition & Costs)

Food	Cost ($)	Cal	Prot (g)	Carb (g)	Sugar (g)	Fiber (g)	Fat (g)
Chicken	1.80	128	24.0	0.0	0.0	0.0	2.7
Banana	0.30	105	1.3	27.0	14.0	3.1	0.4
Yogurt	0.90	104	5.9	7.9	7.9	0.0	5.5
Beans	1.10	120	8.0	21.0	1.0	7.0	0.5
Spinach	0.40	7	0.9	1.1	0.1	0.7	0.1
Almonds	0.70	160	6.0	6.0	1.0	3.0	14.0

Requirements:

- Calories ≥ 2000
- Protein ≥ 100g
- Fiber ≥ 50g
- Sugar ≤ 50g
- Fat ≤ 120g
- Sodium ≤ 2300mg

The Mathematical Model

Let x_j be the number of servings of food j. Let c_j be the cost of food j. Let a_{ij} be the amount of nutrient i in food j.

The Mathematical Model

Let x_j be the number of servings of food j. Let c_j be the cost of food j. Let a_{ij} be the amount of nutrient i in food j.

$$\min \sum_{j \in \text{Foods}} c_j x_j \qquad \text{(Minimize Cost)}$$

The Mathematical Model

Let x_j be the number of servings of food j. Let c_j be the cost of food j. Let a_{ij} be the amount of nutrient i in food j.

$$\min \sum_{j \in \text{Foods}} c_j x_j \qquad \text{(Minimize Cost)}$$

$$\text{s.t.} \sum_j \text{Cal}_j \cdot x_j \geq 2000$$

The Mathematical Model

Let x_j be the number of servings of food j. Let c_j be the cost of food j. Let a_{ij} be the amount of nutrient i in food j.

$$\min \quad \sum_{j \in \text{Foods}} c_j x_j \qquad \text{(Minimize Cost)}$$

$$\begin{aligned} \text{s.t.} \quad \sum_j \text{Cal}_j \cdot x_j &\geq 2000 \\ \sum_j \text{Prot}_j \cdot x_j &\geq 100 \end{aligned}$$

The Mathematical Model

Let x_j be the number of servings of food j. Let c_j be the cost of food j. Let a_{ij} be the amount of nutrient i in food j.

$$\min \sum_{j \in \text{Foods}} c_j x_j \qquad \text{(Minimize Cost)}$$

$$\begin{aligned}
\text{s.t.} \quad \sum_j \text{Cal}_j \cdot x_j &\geq 2000 \\
\sum_j \text{Prot}_j \cdot x_j &\geq 100 \\
\sum_j \text{Sugar}_j \cdot x_j &\leq 50
\end{aligned}$$

The Mathematical Model

Let x_j be the number of servings of food j. Let c_j be the cost of food j. Let a_{ij} be the amount of nutrient i in food j.

$$\min \sum_{j \in \text{Foods}} c_j x_j \qquad \text{(Minimize Cost)}$$

$$\begin{aligned}
\text{s.t.} \quad & \sum_j \text{Cal}_j \cdot x_j && \geq 2000 \\
& \sum_j \text{Prot}_j \cdot x_j && \geq 100 \\
& \sum_j \text{Sugar}_j \cdot x_j && \leq 50 \\
& \ldots
\end{aligned}$$

The Mathematical Model

Let x_j be the number of servings of food j. Let c_j be the cost of food j. Let a_{ij} be the amount of nutrient i in food j.

$$
\begin{array}{llll}
\min & \displaystyle\sum_{j \in \text{Foods}} c_j x_j & & \text{(Minimize Cost)} \\
\text{s.t.} & \displaystyle\sum_j \text{Cal}_j \cdot x_j & \geq & 2000 \\
 & \displaystyle\sum_j \text{Prot}_j \cdot x_j & \geq & 100 \\
 & \displaystyle\sum_j \text{Sugar}_j \cdot x_j & \leq & 50 \\
 & \ldots & & \\
 & x_j & \geq & 0
\end{array}
$$

Implementation in Gurobi

```
params = {
    "Chicken" : { "price": 1.80, "protein": 24.0, "sugar": 0.0, "..." : "..." },
    "Banana"  : { "price": 0.30, "protein": 1.3,  "sugar": 14.0, "..." : "..." },
    "Yogurt"  : { "price": 0.90, "protein": 5.9,  "sugar": 7.9,  "..." : "..." },
    "Beans"   : { "price": 1.10, "protein": 8.0,  "sugar": 1.0,  "..." : "..." },
    ...
}
foods = list(params.keys())
```

Implementation in Gurobi

```
params = {
    "Chicken" : { "price": 1.80, "protein": 24.0, "sugar": 0.0, "..." : "..." },
    "Banana"  : { "price": 0.30, "protein": 1.3,  "sugar": 14.0, "..." : "..." },
    "Yogurt"  : { "price": 0.90, "protein": 5.9,  "sugar": 7.9,  "..." : "..." },
    "Beans"   : { "price": 1.10, "protein": 8.0,  "sugar": 1.0,  "..." : "..." },
    ...
}
foods = list(params.keys())
# Variables: x[food]
x = m.addVars(foods, lb=0.0, name="servings")
```

Implementation in Gurobi

```python
params = {
    "Chicken" : { "price": 1.80, "protein": 24.0, "sugar": 0.0, "..." : "..." },
    "Banana"  : { "price": 0.30, "protein": 1.3,  "sugar": 14.0, "..." : "..." },
    "Yogurt"  : { "price": 0.90, "protein": 5.9,  "sugar": 7.9,  "..." : "..." },
    "Beans"   : { "price": 1.10, "protein": 8.0,  "sugar": 1.0,  "..." : "..." },
    ...
}
foods = list(params.keys())
# Variables: x[food]
x = m.addVars(foods, lb=0.0, name="servings")
# Objective: Minimize Cost
obj_expr = 0
for food in foods:
    obj_expr += params[food]["price"] * x[i]
m.setObjective(obj_expr, GRB.MINIMIZE)
```

Implementation in Gurobi

```
params = {
    "Chicken" : { "price": 1.80, "protein": 24.0, "sugar": 0.0, "..." : "..." },
    "Banana"  : { "price": 0.30, "protein": 1.3,  "sugar": 14.0, "..." : "..." },
    "Yogurt"  : { "price": 0.90, "protein": 5.9,  "sugar": 7.9,  "..." : "..." },
    "Beans"   : { "price": 1.10, "protein": 8.0,  "sugar": 1.0,  "..." : "..." },
    ...
}
foods = list(params.keys())
# Variables: x[food]
x = m.addVars(foods, lb=0.0, name="servings")
# Objective: Minimize Cost
obj_expr = 0
for food in foods:
    obj_expr += params[food]["price"] * x[i]
m.setObjective(obj_expr, GRB.MINIMIZE)
# Constraints (Example: Protein & Sugar)
const_protein = m.addConstr(
                    gp.quicksum( params[fd]["protein"] * x[fd] for fd in foods) >= 100, "min_protein"
                )
```

Implementation in Gurobi

```
params = {
    "Chicken" : { "price": 1.80, "protein": 24.0, "sugar": 0.0, "..." : "..." },
    "Banana"  : { "price": 0.30, "protein": 1.3,  "sugar": 14.0, "..." : "..." },
    "Yogurt"  : { "price": 0.90, "protein": 5.9,  "sugar": 7.9,  "..." : "..." },
    "Beans"   : { "price": 1.10, "protein": 8.0,  "sugar": 1.0,  "..." : "..." },
    ...
}
foods = list(params.keys())
# Variables: x[food]
x = m.addVars(foods, lb=0.0, name="servings")
# Objective: Minimize Cost
obj_expr = 0
for food in foods:
    obj_expr += params[food]["price"] * x[i]
m.setObjective(obj_expr, GRB.MINIMIZE)
# Constraints (Example: Protein & Sugar)
const_protein = m.addConstr(
                    gp.quicksum( params[fd]["protein"] * x[fd] for fd in foods) >= 100, "min_protein"
                )
const_sugar = m.addConstr(gp.quicksum(params[food]["sugar"]  * x[i] for i in foods) <= 50, "max_sugar")
```

Implementation in Gurobi

```
params = {
    "Chicken" : { "price": 1.80, "protein": 24.0, "sugar": 0.0, "..." : "..." },
    "Banana"  : { "price": 0.30, "protein": 1.3,  "sugar": 14.0, "..." : "..." },
    "Yogurt"  : { "price": 0.90, "protein": 5.9,  "sugar": 7.9,  "..." : "..." },
    "Beans"   : { "price": 1.10, "protein": 8.0,  "sugar": 1.0,  "..." : "..." },
    ...
}
foods = list(params.keys())
# Variables: x[food]
x = m.addVars(foods, lb=0.0, name="servings")
# Objective: Minimize Cost
obj_expr = 0
for food in foods:
    obj_expr += params[food]["price"] * x[i]
m.setObjective(obj_expr, GRB.MINIMIZE)
# Constraints (Example: Protein & Sugar)
const_protein = m.addConstr(
                    gp.quicksum( params[fd]["protein"] * x[fd] for fd in foods) >= 100, "min_protein"
                )
const_sugar = m.addConstr(gp.quicksum(params[food]["sugar"]  * x[i] for i in foods) <= 50, "max_sugar")
# ... rest of the requirements ...
m.optimize()
```

Reading the Tea Leaves (Gurobi Output)

When you run `m.optimize()`, Gurobi populates attributes on the objects.

Model Attributes:

- `m.Status`: Did it work?
  (2=Opt, 3=Infeas, 5=Unbdd)

Reading the Tea Leaves (Gurobi Output)

When you run `m.optimize()`, Gurobi populates attributes on the objects.

Model Attributes:

- `m.Status`: Did it work?
  (2=Opt, 3=Infeas, 5=Unbdd)
- `m.ObjVal`: The total profit/cost (Z).

Reading the Tea Leaves (Gurobi Output)

When you run `m.optimize()`, Gurobi populates attributes on the objects.

Model Attributes:

- `m.Status`: Did it work?
  (2=Opt, 3=Infeas, 5=Unbdd)
- `m.ObjVal`: The total profit/cost (Z).

Variable Attributes:

- `var.X`: The optimal value
  ($x_1 = 6.66$).

Reading the Tea Leaves (Gurobi Output)

When you run `m.optimize()`, Gurobi populates attributes on the objects.

Model Attributes:

- `m.Status`: Did it work?
  (2=Opt, 3=Infeas, 5=Unbdd)
- `m.ObjVal`: The total profit/cost (Z).

Variable Attributes:

- `var.X`: The optimal value
  ($x_1 = 6.66$).
- `var.RC`: **Reduced Cost**. How much the objective coefficient must improve before this variable becomes non-zero (More next week).

Reading the Tea Leaves (Gurobi Output)

When you run `m.optimize()`, Gurobi populates attributes on the objects.

Model Attributes:

- `m.Status`: Did it work? (2=Opt, 3=Infeas, 5=Unbdd)
- `m.ObjVal`: The total profit/cost (Z).

Variable Attributes:

- `var.X`: The optimal value ($x_1 = 6.66$).
- `var.RC`: **Reduced Cost**. How much the objective coefficient must improve before this variable becomes non-zero (More next week).

Constraint Attributes:

- `constr.Slack`: Difference between LHS and RHS.

Reading the Tea Leaves (Gurobi Output)

When you run `m.optimize()`, Gurobi populates attributes on the objects.

Model Attributes:

- `m.Status`: Did it work? (2=Opt, 3=Infeas, 5=Unbdd)
- `m.ObjVal`: The total profit/cost (Z).

Variable Attributes:

- `var.X`: The optimal value ($x_1 = 6.66$).
- `var.RC`: **Reduced Cost**. How much the objective coefficient must improve before this variable becomes non-zero (More next week).

Constraint Attributes:

- `constr.Slack`: Difference between LHS and RHS.
- `constr.Pi` (π): **Shadow Price**. *"If I had 1 more unit of Metal, how much more profit would I make?". More on this next week!*

Reading the Tea Leaves (Gurobi Output)

When you run `m.optimize()`, Gurobi populates attributes on the objects.

Model Attributes:

- `m.Status`: Did it work? (2=Opt, 3=Infeas, 5=Unbdd)
- `m.ObjVal`: The total profit/cost (Z).

Variable Attributes:

- `var.X`: The optimal value ($x_1 = 6.66$).
- `var.RC`: **Reduced Cost**. How much the objective coefficient must improve before this variable becomes non-zero (More next week).

Constraint Attributes:

- `constr.Slack`: Difference between LHS and RHS.
- `constr.Pi` (π): **Shadow Price**. *"If I had 1 more unit of Metal, how much more profit would I make?". More on this next week!*

Warning

Attributes like `.X` and `.Pi` are only available if `m.Status == 2` (Optimal). Always check status first!

Infeasibility Diagnosis

```python
import gurobipy as gp
import gurobipy

m = gp.Model("Infeasible")
x = m.addVar(name="x")
m.setObjective(-1*x, gp.GRB.MAXIMIZE)
m.addConstr(x>=3)
m.addConstr(x<=2)
m.optimize()
print("Optimize status:", m.Status)
```

Infeasibility Diagnosis

```python
import gurobipy as gp
import gurobipy

m = gp.Model("Infeasible")
x = m.addVar(name="x")
m.setObjective(-1*x, gp.GRB.MAXIMIZE)
m.addConstr(x>=3)
m.addConstr(x<=2)
m.optimize()
print("Optimize status:", m.Status)
```

```
Gurobi Optimizer version 12.0.3 build v12.0.3rc0 (mac64[arm]
- Darwin 23.1.0 23B2073)

CPU model: Apple M3 Max
Thread count: 14 physical cores, 14 logical processors, using up to 14 threa

Optimize a model with 2 rows, 1 columns and 2 nonzeros
Model fingerprint: 0xf5b06d2b
Coefficient statistics:
  Matrix range     [1e+00, 1e+00]
  Objective range  [1e+00, 1e+00]
  Bounds range     [0e+00, 0e+00]
  RHS range        [2e+00, 3e+00]
Presolve time: 0.00s

Solved in 0 iterations and 0.00 seconds (0.00 work units)
Infeasible model
Optimize status: 3
```

What about Larger Models?

```
import gurobipy as gp

m = gp.Model("TrickyInfeasible")

# Variables
x = m.addVar(lb=-0, ub=8, name="x")
y = m.addVar(lb=-0, ub=8, name="y")
```

What about Larger Models?

```python
import gurobipy as gp

m = gp.Model("TrickyInfeasible")

# Variables
x = m.addVar(lb=-0, ub=8, name="x")
y = m.addVar(lb=-0, ub=8, name="y")
# Arbitrary bounded objective
m.setObjective(x + y, gp.GRB.MINIMIZE)
```

What about Larger Models?

```python
import gurobipy as gp

m = gp.Model("TrickyInfeasible")

# Variables
x = m.addVar(lb=-0, ub=8, name="x")
y = m.addVar(lb=-0, ub=8, name="y")
# Arbitrary bounded objective
m.setObjective(x + y, gp.GRB.MINIMIZE)
#Constraints
m.addConstr(2*x + y <= 4, name="c1_budget1")
m.addConstr(x + 2*y <= 4, name="c2_budget2")
m.addConstr(x + y >= 5,   name="c3_demand")
m.addConstr(x <= 8,       name="c4_x_cap")
m.addConstr(y <= 8,       name="c5_y_cap")
```

What about Larger Models?

```python
import gurobipy as gp

m = gp.Model("TrickyInfeasible")

# Variables
x = m.addVar(lb=-0, ub=8, name="x")
y = m.addVar(lb=-0, ub=8, name="y")
# Arbitrary bounded objective
m.setObjective(x + y, gp.GRB.MINIMIZE)
#Constraints
m.addConstr(2*x + y <= 4, name="c1_budget1")
m.addConstr(x + 2*y <= 4, name="c2_budget2")
m.addConstr(x + y >= 5,   name="c3_demand")
m.addConstr(x <= 8,       name="c4_x_cap")
m.addConstr(y <= 8,       name="c5_y_cap")
m.optimize()
print("Optimize status:", m.Status)
```

What about Larger Models?

```python
import gurobipy as gp

m = gp.Model("TrickyInfeasible")

# Variables
x = m.addVar(lb=-0, ub=8, name="x")
y = m.addVar(lb=-0, ub=8, name="y")
# Arbitrary bounded objective
m.setObjective(x + y, gp.GRB.MINIMIZE)
#Constraints
m.addConstr(2*x + y <= 4, name="c1_budget1")
m.addConstr(x + 2*y <= 4, name="c2_budget2")
m.addConstr(x + y >= 5,   name="c3_demand")
m.addConstr(x <= 8,       name="c4_x_cap")
m.addConstr(y <= 8,       name="c5_y_cap")
m.optimize()
print("Optimize status:", m.Status)
```

```
Optimize a model with 5 rows, 2 columns and 8 nonzeros
Model fingerprint: 0x00fc1d77
Coefficient statistics:
  Matrix range     [1e+00, 2e+00]
  Objective range  [1e+00, 1e+00]
  Bounds range     [8e+00, 8e+00]
  RHS range        [4e+00, 8e+00]
Presolve removed 2 rows and 0 columns
Presolve time: 0.01s

Solved in 0 iterations and 0.01 seconds (0.00 work units)
Infeasible model
Optimize status: 3
```

Irreducible Infeasible Subsystem (IIS)

What is an IIS?

- When a model is **infeasible**, the full set of constraints cannot all be satisfied simultaneously.

Irreducible Infeasible Subsystem (IIS)

What is an IIS?

- When a model is **infeasible**, the full set of constraints cannot all be satisfied simultaneously.
- An **IIS** is a *minimal subset of constraints and bounds* that is still infeasible.

Irreducible Infeasible Subsystem (IIS)

What is an IIS?

- When a model is **infeasible**, the full set of constraints cannot all be satisfied simultaneously.
- An **IIS** is a *minimal subset of constraints and bounds* that is still infeasible.
- “Minimal” = removing *any* constraint from that subset makes it feasible again.

Irreducible Infeasible Subsystem (IIS)

What is an IIS?

- When a model is **infeasible**, the full set of constraints cannot all be satisfied simultaneously.
- An **IIS** is a *minimal subset of constraints and bounds* that is still infeasible.
- "Minimal" = removing *any* constraint from that subset makes it feasible again.
- IISs help pinpoint the true source of infeasibility in large models.

Good News

Gurobi can compute an IIS for you automatically!

Computing an IIS in Gurobi

If the model is infeasible, we can ask Gurobi to identify the conflicting constraints.

Computing an IIS in Gurobi

If the model is infeasible, we can ask Gurobi to identify the conflicting constraints.

```
if m.Status == GRB.INFEASIBLE:
    print("\nModel is infeasible; computing IIS...")
    m.computeIIS()

    print("Constraints in the IIS:")
    for c in m.getConstrs():
        if c.IISConstr:   # True if part of the IIS
            print(f"  {c.ConstrName}")
```

Example IIS Output

```
Model is infeasible; computing IIS...

Iteration    Objective       Primal Inf.    Dual Inf.      Time
       0    0.0000000e+00   2.500000e+00   0.000000e+00      0s

IIS computed: 3 constraints and 0 bounds
IIS runtime: 0.00 seconds (0.00 work units)
Constraints in the IIS:
  c1_budget1
  c2_budget2
  c3_demand
```

Example IIS Output

```
Model is infeasible; computing IIS...

Iteration    Objective       Primal Inf.    Dual Inf.      Time
       0    0.0000000e+00   2.500000e+00   0.000000e+00      0s

IIS computed: 3 constraints and 0 bounds
IIS runtime: 0.00 seconds (0.00 work units)
Constraints in the IIS:
  c1_budget1
  c2_budget2
  c3_demand
```

- Remember, these constraints correspond to $2x + y \leq 4$, $x + 2y \leq 4$, and $x + y \geq 5$. Adding the first 2 inequalities contradicts the third.

Example IIS Output

```
Model is infeasible; computing IIS...

Iteration     Objective        Primal Inf.     Dual Inf.       Time
       0    0.0000000e+00    2.500000e+00    0.000000e+00      0s

IIS computed: 3 constraints and 0 bounds
IIS runtime: 0.00 seconds (0.00 work units)
Constraints in the IIS:
  c1_budget1
  c2_budget2
  c3_demand
```

- Remember, these constraints correspond to $2x + y \leq 4$, $x + 2y \leq 4$, and $x + y \geq 5$. Adding the first 2 inequalities contradicts the third.
- These are the **minimal conflicting constraints**.

Example IIS Output

```
Model is infeasible; computing IIS...

Iteration    Objective       Primal Inf.    Dual Inf.      Time
       0    0.0000000e+00   2.500000e+00   0.000000e+00      0s

IIS computed: 3 constraints and 0 bounds
IIS runtime: 0.00 seconds (0.00 work units)
Constraints in the IIS:
  c1_budget1
  c2_budget2
  c3_demand
```

- Remember, these constraints correspond to $2x + y \leq 4$, $x + 2y \leq 4$, and $x + y \geq 5$. Adding the first 2 inequalities contradicts the third.
- These are the **minimal conflicting constraints**.
- Removing any one of them would make the model feasible.

Example IIS Output

```
Model is infeasible; computing IIS...

Iteration     Objective        Primal Inf.     Dual Inf.      Time
       0    0.0000000e+00   2.500000e+00   0.000000e+00      0s

IIS computed: 3 constraints and 0 bounds
IIS runtime: 0.00 seconds (0.00 work units)
Constraints in the IIS:
  c1_budget1
  c2_budget2
  c3_demand
```

- Remember, these constraints correspond to $2x + y \leq 4$, $x + 2y \leq 4$, and $x + y \geq 5$. Adding the first 2 inequalities contradicts the third.
- These are the **minimal conflicting constraints**.
- Removing any one of them would make the model feasible.
- Great for isolating modeling mistakes in large LPs/MIPs.

Unbounded LPs and Infinite Directions

Unbounded LP = The objective can grow without limit while staying feasible.

Unbounded LPs and Infinite Directions

Unbounded LP = The objective can grow without limit while staying feasible.
Gurobi not only detects unboundedness, it returns an *unbounded ray*.

- An **unbounded ray** is a vector d such that:

$$x + \lambda d \text{ is feasible for all } \lambda > 0$$

  and the objective coefficient $c^T d > 0$ (for maximization).

Unbounded LPs and Infinite Directions

Unbounded LP = The objective can grow without limit while staying feasible.

Gurobi not only detects unboundedness, it returns an *unbounded ray*.

- An **unbounded ray** is a vector d such that:

  $$x + \lambda d \text{ is feasible for all } \lambda > 0$$

  and the objective coefficient $c^T d > 0$ (for maximization).
- Gurobi provides this via the attribute:

  `var.UnbdRay`

Unbounded LPs and Infinite Directions

Unbounded LP = The objective can grow without limit while staying feasible.

Gurobi not only detects unboundedness, it returns an *unbounded ray*.

- An **unbounded ray** is a vector d such that:

$$x + \lambda d \text{ is feasible for all } \lambda > 0$$

  and the objective coefficient $c^T d > 0$ (for maximization).

- Gurobi provides this via the attribute:

```
var.UnbdRay
```

- Nonzero components of the ray indicate which variables "run off to infinity."

Unbounded LPs and Infinite Directions

Unbounded LP = The objective can grow without limit while staying feasible.
Gurobi not only detects unboundedness, it returns an *unbounded ray*.

- An **unbounded ray** is a vector d such that:

  $$x + \lambda d \text{ is feasible for all } \lambda > 0$$

  and the objective coefficient $c^T d > 0$ (for maximization).
- Gurobi provides this via the attribute:

  ```
  var.UnbdRay
  ```

- Nonzero components of the ray indicate which variables “run off to infinity.”

Interpretation

The unbounded ray shows *how* the LP escapes to infinity.

Example of an Unbounded LP?

Example (Maximization):

$$\max \; x + y$$

$$\text{s.t. } x - y \geq 1$$

$$x, y \geq 0$$

Example of an Unbounded LP?

Example (Maximization):

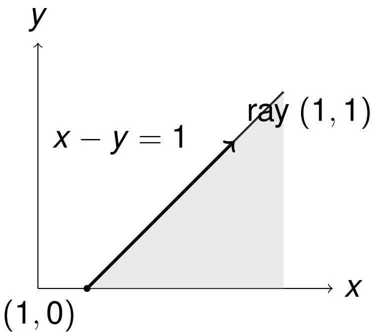
$$\max \; x + y$$

$$\text{s.t. } x - y \geq 1$$

$$x, y \geq 0$$

- Feasible region goes to ∞.
- Objective increases without bound.
- No vertex optimum exists.

Geometry of the Unbounded Ray

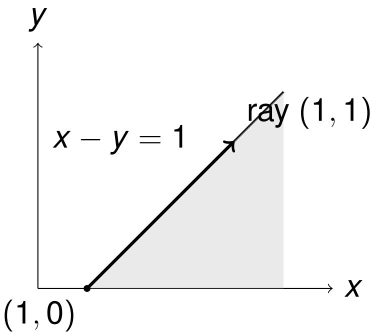


Feasible region:

$$x - y \geq 1, \quad x \geq 0, \quad y \geq 0.$$

- From the feasible point $(1, 0)$ we can move along $(x, y) = (1, 0) + \lambda(1, 1) = (1 + \lambda, \lambda), \ \lambda \geq 0$.

Geometry of the Unbounded Ray

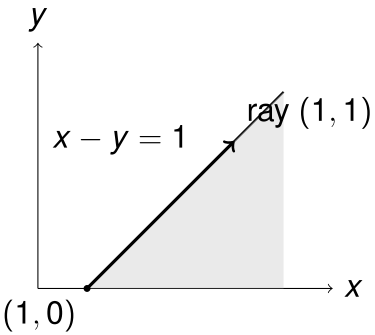


Feasible region:

$$x - y \geq 1, \quad x \geq 0, \quad y \geq 0.$$

- From the feasible point $(1,0)$ we can move along $(x,y) = (1,0) + \lambda(1,1) = (1+\lambda, \lambda), \ \lambda \geq 0.$
- The objective $x + y$ grows without bound:
$$1 + 2\lambda \to \infty.$$

Geometry of the Unbounded Ray



Feasible region:

$$x - y \geq 1, \quad x \geq 0, \quad y \geq 0.$$

- From the feasible point $(1, 0)$ we can move along $(x, y) = (1, 0) + \lambda(1, 1) = (1 + \lambda, \lambda), \ \lambda \geq 0.$
- The objective $x + y$ grows without bound:
$$1 + 2\lambda \to \infty.$$
- Gurobi's `UnbdRay` returns this direction.

Gurobi Example: Unbounded Model + Ray

```python
import gurobipy as gp
from gurobipy import GRB

m = gp.Model("Unbounded")
x = m.addVar(lb=0, name="x")
y = m.addVar(lb=0, name="y")
m.setObjective(x + y, GRB.MAXIMIZE)
m.addConstr(x - y >= 1, name="c1_skew")
# KEY: ask Gurobi to compute ray info
m.setParam(GRB.Param.InfUnbdInfo, 1)
m.optimize()
```

Gurobi Example: Unbounded Model + Ray

```python
import gurobipy as gp
from gurobipy import GRB

m = gp.Model("Unbounded")
x = m.addVar(lb=0, name="x")
y = m.addVar(lb=0, name="y")
m.setObjective(x + y, GRB.MAXIMIZE)
m.addConstr(x - y >= 1, name="c1_skew")
# KEY: ask Gurobi to compute ray info
m.setParam(GRB.Param.InfUnbdInfo, 1)
m.optimize()

print("Status:", m.Status)
if m.Status == GRB.UNBOUNDED:
    print("\nUnbounded Ray:")
    for v in m.getVars():
        print(f"{v.VarName}: {v.UnbdRay}")
```

Gurobi Example: Unbounded Model + Ray

```python
import gurobipy as gp
from gurobipy import GRB

m = gp.Model("Unbounded")
x = m.addVar(lb=0, name="x")
y = m.addVar(lb=0, name="y")
m.setObjective(x + y, GRB.MAXIMIZE)
m.addConstr(x - y >= 1, name="c1_skew")
# KEY: ask Gurobi to compute ray info
m.setParam(GRB.Param.InfUnbdInfo, 1)
m.optimize()

print("Status:", m.Status)
if m.Status == GRB.UNBOUNDED:
    print("\nUnbounded Ray:")
    for v in m.getVars():
        print(f"{v.VarName}: {v.UnbdRay}")
```

```
Status: 5

Unbounded Ray:
x: 1.0
y: 1.0
```

Gurobi Example: Unbounded Model + Ray

```python
import gurobipy as gp
from gurobipy import GRB

m = gp.Model("Unbounded")
x = m.addVar(lb=0, name="x")
y = m.addVar(lb=0, name="y")
m.setObjective(x + y, GRB.MAXIMIZE)
m.addConstr(x - y >= 1, name="c1_skew")
# KEY: ask Gurobi to compute ray info
m.setParam(GRB.Param.InfUnbdInfo, 1)
m.optimize()

print("Status:", m.Status)
if m.Status == GRB.UNBOUNDED:
    print("\nUnbounded Ray:")
    for v in m.getVars():
        print(f"{v.VarName}: {v.UnbdRay}")
```

```
Status: 5

Unbounded Ray:
x: 1.0
y: 1.0
```

- The ray $(1, 1)$ means both x and y can increase indefinitely.
- The constraint $x - y \geq 1$ stays satisfied for all $(x, y) = (1, 0) + \lambda(1, 1)$.
- Objective grows as $x + y \to +\infty$.

TODOs after Lecture.

- **Install Gurobi:** Get your academic license working.
- Code and Solve **The Diet Problem** in HW1.
- Use **Tools** like `m.computeIIS()` and `var.UnbdRay` to find the conflict in toy infeasible models and unbounded models.