# CS498 Homework1

November 21, 2025

Gradescope Entry Code: PK5XK2

## 1 Problem 1 — Matrix-Form LPs & Programmatic Solvers

This problem is a warm-up to ensure you are comfortable (1) writing linear programs in **matrix form**, and (2) building simple LP solvers **programmatically** using **Gurobi**. You will complete both a **written component** and a **coding component**. The coding portion will be autograded on Gradescope, and **your code will be tested on multiple LPs**, *not only the example* shown in the setup.

---

### 1.1 Setup Example (for the Written Part Only)

Consider the linear program:

$$\begin{aligned} \max\ & 3x + 2y \\ \text{s.t. } & x + y \le 4 \\ & 2x + y \le 5 \\ & x \ge 0,\ y \ge 0. \end{aligned}$$

This example is used **only** for the written portion.
Your code must handle general LPs of the form:

$$\max\ c^\top x \quad \text{s.t. } Ax \le b,\ x \ge 0.$$

---

### 1.2 Task 1 (Written): Matrix Form [30 Points]

Write the example LP above in matrix form $Ax \le b$ and vector $c$, where:

- $x = \begin{bmatrix} x \\ y \end{bmatrix}$,
- $A$ is an $m \times 2$ matrix,
- $b$ is length $m$,
- $c$ is length 2.

Next, plot the feasible region, a few objective function level sets, and the visually point out the optimal solution of the LP.

Upload this to *Problem 0 Written* on Gradescope.

---

### 1.3 Programming Tasks (Autograded)

Submit a file named **`hw1_p1_lp.py`** containing the **three functions defined below**. Your code will be tested on several different LPs; do **not** hard-code the example.

Import NumPy and Gurobi at the top of your file.

---

### 1.4 Function 1 — Build a Gurobi Model [30 Points]

```
def build_model(A, b, c):
    """
    Input:
        A: (m × n) NumPy array
        b: length-m NumPy array
        c: length-n NumPy array

    Output:
        A Gurobi model for the LP:

            maximize      c^T x
            subject to    A x <= b
                           x >= 0

        where x has n nonnegative decision variables.
    """
```

---

### 1.5 Function 2 — Solve an LP Using Gurobi [10 Points]

```
def solve_with_gurobi(A, b, c):
    """
    Input:
        A: (m × n) NumPy array
        b: length-m NumPy array
        c: length-n NumPy array

    Builds the model using build_model(A, b, c),
    solves it with Gurobi, and returns (x_opt, obj_val):
        x_opt: optimal solution vector (NumPy array length n)
        obj_val: optimal objective value (float)
    """
```

This function must: - call your `build_model` function, - optimize the model, - extract the solution and objective value.

---

### 1.6 Function 3 — Enumerate 2D Polytope Vertices [30 Points]

```
def enumerate_vertices(A, b, c):
    """
    Enumerates all vertices (extreme points) of the 2D polytope

          P = { x in R^2 : A x <= b, x>=0 },

    and returns:

        vertices: list of NumPy arrays (each of length 2)
        best_x:   vertex achieving max c^T x subject to A x <= b and x>=0 from vertices
        best_obj: the corresponding objective value of best_x

    Assumptions:
        - A is an (m × 2) matrix.
        - Vertices are finite.

    This must work for *arbitrary* 2D LPs, not just the example.
    """
```

You may use any correct geometric method (e.g., intersecting constraint pairs + feasibility checks). Consider the python function `itertools.combinations`.

---

### 1.7 Important Notes

#### 1.7.1 Autograder

Your code will be tested on **multiple randomly generated LPs** of various sizes and shapes.

#### 1.7.2 Requirements

- Submit **`hw1_p1_lp.py`** to *Homework 1 Problem 1* on Gradescope.
- Do **not** submit additional files.
- Your functions must have *exactly* the names and signatures shown above.

---

## 2 Problem 2 — Primal and Dual: Two Sides of the Same Coin

**Goal:** Write the dual of a simple LP manually, solve both, and verify strong duality numerically.

### 2.1 Context

In class we saw that every LP in standard form

$$\max c^\top x \quad \text{s.t. } Ax \leq b, x \geq 0$$

has a **dual** LP

$$\min b^\top y \quad \text{s.t. } A^\top y \geq c, y \geq 0,$$

and that both share the same optimal value when both are feasible.

### 2.2 Tasks

1. Consider:
$$\max 3x_1 + 5x_2 \quad \text{s.t. } \begin{cases} x_1 + 2x_2 \leq 8 \\ 2x_1 + x_2 \leq 8 \\ x_1, x_2 \geq 0 \end{cases}$$
2. **Derive its dual manually.**
3. **Solve both LPs in Gurobi (Or Manually!).**
   - Build two models (`model_primal`, `model_dual`).
   - Compare optimal values (`ObjVal`).
4. **Interpret the result.**
   - Verify Strong Duality (i.e. they're equal).

You should submit the following to Homework 1 Problem 2 on Gradescope: the primal optimal solution $(x_1^*, x_2^*)$, the corresponding primal optimal objective value, the dual optimal solution, the dual optimal objective value, and the code used to obtain these results (or the plots if you did it geometrically).

---

# 3 Problem 3: Visualizing Linear Programs in 2D

Before you trust solvers, you should *see* what they're doing. In this problem, you'll plot a simple 2-variable linear program, guess the optimal solution, and then ask `gurobipy` to confirm whether your intuition was right.

## 3.1 Scenario

Imagine you're designing a micro-factory that produces two products: **Gadgets (x)** and **Widgets (y)**. Each consumes limited resources and generates profit.

You need to decide how many of each to produce under these linear constraints:

$$\begin{aligned} x + 2y &\leq 8 \qquad \text{(Resource A)} \\ 3x + y &\leq 9 \qquad \text{(Resource B)} \\ x, y &\geq 0 \end{aligned}$$

and your goal is to **maximize** the total profit $z = 3x + 2y$.

## 3.2 Tasks

1. **Plot the feasible region**:
   - Draw each constraint line in Matplotlib.
   - Shade the feasible polygon defined by the inequalities.
2. **Add objective lines**:
   - Draw a few "profit" lines $3x + 2y = c$ for different $c$ (for example $6, 9, 10, ...$).
   - Visually slide the line upward and *predict* where it last touches the feasible region.
3. **Visually guess the optimum:** estimate $(x^*, y^*)$ from your plot.
4. **Verify with `gurobipy`**:
   - Build and solve the LP in Python.
   - Print the optimal values and objective.
   - Mark the solver's optimal point on your plot.
5. **Reflect:**
   - Does the solver's answer match your geometric intuition?
   - Which constraints are "tight" at the optimum?

## 3.3 Deliverable

A plot showing: * The feasible region (shaded polygon) * A few objective lines * The optimal point from Guroby annotated * Write a sentence or two explaining how the geometry connects to what Gurobi reports.

You should submit your written answer to Homework 1 Problem 3 on Gradescope.

---

# 4 Problem 4: The Engineer's Diet Dilemma (Coding)

> You've just joined *OptiMeal Inc.*, a startup building the world's most efficient meal-planning engine. The finance team insists on cutting costs, while the nutritionists demand balance.

Your job: use **linear programming** to design the **cheapest** daily meal plan that still meets basic nutritional requirements.

This is a **coding-only** problem. It will be autograded on Gradescope.

---

## 4.1 Dataset and Setup

We'll work with a small food dataset in a **pandas DataFrame**.
Each row is a food, each column is a property (price, calories, etc.):

- `food_one_serving` – name of the food (string)
- `price_usd_per_serving` – cost of one serving (float)
- `calories_kcal`
- `protein_g`
- `carbs_g`
- `sugar_g`
- `fiber_g`
- `fat_g`
- `sodium_mg`

Example (the autograder will build something like this for you):

```
import pandas as pd
from io import StringIO

csv = StringIO("""food_one_serving,price_usd_per_serving,calories_kcal,protein_g,carbs_g,sugar
chicken,1.80,128,24,0,0,0,2.7,44
banana,0.30,105,1.3,27,14,3.1,0.4,1
yogurt,0.90,104,5.9,7.9,7.9,0,5.5,70
beans,1.10,120,8,21,1,7,0.5,2
spinach,0.40,7,0.9,1.1,0.1,0.7,0.1,24
almonds,0.70,160,6,6,1,3,14,1
""")

df = pd.read_csv(csv)
```

> **Important:** In your solution, **do not read files from disk**.
> The autograder will create a `DataFrame` and pass it into your function.

---

## 4.2 Decision Variables

We decide how many **servings** of each food to eat in a day.

Let: - $x_i \geq 0$ = servings of food $i$ (continuous, not necessarily integer) - There is one variable per row of the DataFrame.

You'll create these as Gurobi decision variables.

---

### 4.3 Objective: Minimize Cost

Minimize the **total** daily cost:

$$\text{minimize} \quad \sum_i \text{price}_i \, x_i.$$

Use the `price_usd_per_serving` column.

---

### 4.4 Nutritional Constraints

You must enforce:

- **Minimums**:
  - Total calories $\geq$ `calories_min`
  - Total protein $\geq$ `protein_min`
  - Total fiber $\geq$ `fiber_min`
- **Maximums**:
  - Total sugar $\leq$ `sugar_max`
  - Total fat $\leq$ `fat_max`
  - Total sodium $\leq$ `sodium_max`

These requirements will be passed as a Python **dictionary**:

```
requirements = {
    "calories_min": 2000,
    "protein_min": 100,
    "fiber_min": 50,
    "sugar_max": 50,
    "fat_max": 120,
    "sodium_max": 2300,
}
```

Again, the *exact* numbers used in the autograder may differ, but the keys are the same.

---

### 4.5 What You Must Implement

Create a file named **`hw1_p4_diet.py`** that defines **exactly** this function:

```
import pandas as pd
import numpy as np
import gurobipy as gp
```

```python
def solve_diet(df, requirements):
    """
    Solve the diet LP:

        minimize    total cost (USD)
        subject to  daily nutritional requirements

    Inputs
    ------
    df : pandas.DataFrame
        Each row is a food. Must contain the columns:
        - "food_one_serving"
        - "price_usd_per_serving"
        - "calories_kcal"
        - "protein_g"
        - "fiber_g"
        - "sugar_g"
        - "fat_g"
        - "sodium_mg"

    requirements : dict
        With keys:
        - "calories_min"
        - "protein_min"
        - "fiber_min"
        - "sugar_max"
        - "fat_max"
        - "sodium_max"

    Outputs
    -------
    servings : np.ndarray, shape (len(df),)
        servings[i] is the number of servings of row i in df.

    min_cost : float
        The minimum total cost (objective value).
    """
    # TODO: implement
    raise NotImplementedError
```

### 4.6 Requirements for your implementation

- Use **Gurobi** (`gurobipy`) to set up and solve the LP.
- Decision variables: one per food (each row of `df`), **continuous** and **0** (you can't eat a negative serving).
- Objective: minimize total cost using `price_usd_per_serving`.
- Constraints:

  - `calories_kcal` sum `calories_min`
  - `protein_g` sum `protein_min`
  - `fiber_g` sum `fiber_min`
  - `sugar_g` sum `sugar_max`
  - `fat_g` sum `fat_max`
  - `sodium_mg` sum `sodium_max`
- Return:
  - `servings`: a NumPy array of length `len(df)`
  - `min_cost`: a float

You **may not** assume a specific number of foods. Your code should work for **any** DataFrame with the required columns.

---

### 4.7 Quick Intro: Using `pandas` for This Problem

You will typically access data like:

```
n = len(df)                  # number of foods
prices = df["price_usd_per_serving"].values   # NumPy array of length n
calories = df["calories_kcal"].values
protein  = df["protein_g"].values
fiber    = df["fiber_g"].values
sugar    = df["sugar_g"].values
fat      = df["fat_g"].values
sodium   = df["sodium_mg"].values
```

You can index rows by integer position:

```
for i in range(n):
    price_i = prices[i]
    cals_i  = calories[i]
    # ...
```

The autograder will call **only** your `solve_diet(df, requirements)` function. The problem is worth **100 points** (all from the autograder tests).

---

# 5 Problem 5: The LP Detective

> A mysterious optimization has been solved. You've found the *answer*, but not the *question*.

You join **OptiSolve Forensics**, a consulting unit that reverse-engineers optimization problems from partial clues. Someone claims they solved a linear program, and the resulting point $x^* = (2, 3)$ was "optimal.'' But the objective coefficients were lost.

Your mission: uncover which objectives could make that claim true, or expose their filthy lies to the light of day!

## 5.1 Setup

The only surviving file lists the constraints of the primal problem:

$$\begin{aligned} x + y &\leq 5 && (1)\\ x + 3y &\leq 11 && (2)\\ x &\geq 0 && (3)\\ y &\geq 0 && (4) \end{aligned}$$

The true objective was of the form $\max; c_1 x + c_2 y$ for some unknown vector $c = (c_1, c_2)$.

## 5.2 Tasks

1. **Feasibility:** Verify whether the reported $x^* = (2, 3)$ actually satisfies all constraints. If not, call out the lies!
2. **Active-set geometry:** Find which constraints are *binding* (equalities) at the feasible point $x^*$. Sketch the feasible region and label the active edges.
3. **Objective reconstruction:** Derive all objective vectors $c$ that would make $x^*$ optimal. *(Hint: for a maximization problem with $Ax \leq b$, the valid $c$ lie inside the cone generated by the active constraint normals.)*
4. **Hypothesis testing:** Choose several candidate $c$ vectors, some on the edges of that cone, some inside it, and resolve the LP using Gurobi. Which ones reproduce $x^*$?

## 5.3 Deliverable

A concise report (and optional plots) showing:

- Identified active constraints
- One or more valid $c$ vectors that make $x^*$ optimal (plot $c_1, c_2$ and think of which $c_1, c_2$ cause $x^*$ to be the optimal solution of the LP).
- A description of the set of vectors $c$ that would make $x^*$ optimal.
- Numerical verification from Gurobi that for some of your $c$ vectors, $x^*$ is indeed an optimal vector!
- Brief explanation of geometric intuition

---

# 6 Problem 6 — LP Escape Room: Diagnose Infeasible or Unbounded Models

You're trapped in **OptiLock Labs** with only a solver and its status code. Your goal: identify whether the model is **infeasible** or **unbounded**, and show proof.

## 6.1 Mission

Two short Gurobi models are provided:

1. One **infeasible** (contradictory constraints)
2. One **unbounded** (objective can increase forever)

Your task is to:

1. **Classify** each model as *infeasible* or *unbounded* (based on status + logs).
2. For the infeasible model, run `computeIIS()` and **list the constraints/bounds** that form the minimal conflicting set.
3. For the unbounded model, extract the **unbounded direction vector** using `Var.UnbdRay`.
4. Explain briefly what went wrong in each model and how to fix it.
5. Show your code in your answer!

## 6.2 Starter Code

```
import gurobipy as gp
from gurobipy import GRB

def scenario_1():
    m = gp.Model("scenario_1")
    m.Params.OutputFlag = 1      # show log
    m.Params.InfUnbdInfo = 1     # help produce an unbounded ray for LPs

    # Variables: x1, x2, x3 >= 0
    x1 = m.addVar(lb=0.0, name="x1")
    x2 = m.addVar(lb=0.0, name="x2")
    x3 = m.addVar(lb=0.0, name="x3")

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

    # Constraints
    m.addConstr(x1 - x2 <= 5, name="c1")
    m.addConstr(2 * x1 + x3 >= 3, name="c2")
    m.addConstr(x2 + x3 >= 1, name="c3")

    m.optimize()
    return m

def scenario_2():
    m = gp.Model("scenario_2")
```

```
m.Params.OutputFlag = 1  # show log
m.Params.InfUnbdInfo = 1     # help produce an unbounded ray for LPs

# Variables: x1, x2, x3 >= 0
x1 = m.addVar(lb=0.0, name="x1")
x2 = m.addVar(lb=0.0, name="x2")
x3 = m.addVar(lb=0.0, name="x3")

# Dummy objective: minimize 0
m.setObjective(0 * x1 + 0 * x2 + 0 * x3, GRB.MINIMIZE)

# Constraints
m.addConstr(x1 + x2 + x3 >= 10, name="demand_total")
m.addConstr(x1 + x2 <= 3,       name="cap_xy")
m.addConstr(x2 + x3 <= 3,       name="cap_yz")

m.optimize()
return m
```

### 6.3 Deliverable

For each scenario:

| Item | What to include |
|---|---|
| **Status + Classification** | `m.Status` → `INFEASIBLE` or `UNBOUNDED` |
| **Evidence** | From solver log (e.g., "Unbounded model" or "IIS found") |
| **Proof** | • If infeasible → list names from `c.IISConstr`, `v.IISLB`, `v.IISUB` • If unbounded → print nonzero `v.UnbdRay` values |
| **Fix idea** | One line explaining the modeling mistake (missing bound, contradictory constraint, etc.) |

- Use `m.Status` and constants in `gp.GRB` (`GRB.INFEASIBLE`, `GRB.UNBOUNDED`).
- For IIS:

  ```
  m.computeIIS()
  [c.ConstrName for c in m.getConstrs() if c.IISConstr]
  ```

- For UnbdRay:

  ```
  {v.VarName: v.UnbdRay for v in m.getVars() if abs(v.UnbdRay) > 1e-9}
  ```

**Deliver a concise write-up ( ½ page)** with the classifications, supporting evidence (IIS / UnbdRay), and a one-sentence fix for each case.