Computational Linear Programming & Applications

Sammith Belur & Aaron Yip

MA 421: Linear Programming and Optimization Techniques

September 17, 2026

What are we discussing today?

- So far, we have studied the theoretical aspects of Linear Programming, especially the simplex algorithm
- A natural question is:

  "How are these ideas actually used in industrial, scientific, or other academic applications?"

- Today, we will see the connections between the **theory of Linear Programming** and the **computational tools used to solve LPs in practice**
- I will cover this by introducing commonly used optimization software, comparing different modeling approaches, and doing a few live computational demonstrations.

General form of an LP

Recall that a linear program is generally written in the form

$$\min_{x \in \mathbb{R}^n} c^T x$$

subject to some linear constraints such as

$$Ax \leq b, \quad \text{or} \quad Ax = b$$

Here,

- $x \in \mathbb{R}^n$ is the vector of **decision variables**
- $c \in \mathbb{R}^n$ contains the coefficients of the objective function
- $A \in \mathbb{R}^{m \times n}$ contains the coefficients of the constraints
- $b \in \mathbb{R}^m$ contains the corresponding constraint bounds

So once we formulate a linear program, the problem we want to solve is essentially described by the numerical data in c, A, and b

Method 1: MATLAB `linprog`

MATLAB's built-in tools gives us a function called `linprog` for solving linear programming problems. The function has been a part of MATLAB for decades and has evolved along with all the new algorithms that have come up for Optimization

Generally, we write a linear program as

$$\begin{aligned} \text{minimize} \quad & c^T x \\ \text{subject to} \quad & Ax \leq b, \\ & x \geq 0. \end{aligned}$$

MATLAB uses this exact representation, where you can simply use the command

```
x = linprog(c,A,b,[],[],lb,[])
```

to solve the LP and return an optimal solution x, *when it exists*

Note: MATLAB's documentation typically uses f instead of c for the objective coefficients.

Example: Setting up an LP in MATLAB

Let us consider the linear program

$$\begin{aligned} \text{maximize} \quad & 3x_1 + 5x_2 \\ \text{subject to} \quad & x_1 + 2x_2 \leq 8, \\ & 3x_1 + 2x_2 \leq 12, \\ & x_1, x_2 \geq 0. \end{aligned}$$

It is important to remember that `linprog` solves *minimization problems*, so we have to rewrite the objective function in the form

$$\max \ 3x_1 + 5x_2 \qquad \Longleftrightarrow \qquad \min \ -3x_1 - 5x_2.$$

We will formulate and solve this problem live.

Why are we using Computational Tools?

For the LP we just looked at, we only had two decision variables

We could have solved this problem by

- Using the graphical approach
- Checking its vertices
- Carrying out a few simple simplex iterations by hand

But what about when we have 50, 500, or even 5000 decision variables?

In applications the difficulty is not usually in writing the theory for each model, but efficiently constructing and solving it

We will now look at a larger, application-driven linear program to see how these computational tools become useful in practice

Example: The Diet Problem

Recall that in class, we introduced the **diet problem** using a simple example involving two foods: carrots and cabbage
In this example, we said that

$$x = \text{grams of carrots}, \qquad y = \text{grams of cabbage}$$

The goal was to satisfy minimum nutritional requirements while minimizing our total cost

The linear program we derived was

$$\begin{aligned} \text{minimize} \quad & 2x + 3y \\ \text{subject to} \quad & 2x + y \geq 6, \\ & x + 3y \geq 8, \\ & x, y \geq 0. \end{aligned}$$

This gave us a nice, two-variable example which we could solve quite easily. Now we look at a much larger version of the same idea, where computational tools are required.

Extension: The Stigler Diet Problem

The diet problem we covered in class is a simplified version of a much larger optimization problem studied by economist *George Stigler*, where he analyzed 77 foods and 9 nutritional requirements.

Let

$$x_i = \text{amount of food } i, \qquad i = 1, \ldots, 77.$$

The problem can be written as

$$\begin{aligned} \min \quad & \sum_{i=1}^{77} c_i x_i \\ \text{s.t.} \quad & \sum_{i=1}^{77} a_{ji} x_i \geq b_j, \qquad j = 1, \ldots, 9, \\ & x_i \geq 0. \end{aligned}$$

So the same modeling idea now gives us a linear program with **77 decision variables**.

Why is the Stigler Diet Problem Interesting?

When Stigler studied this problem in 1944, there was no general computational method available to solve this linear program

During his research he wrote

> *"There does not appear to be any direct method of finding the minimum of a linear function subject to linear conditions."*

After George Dantzig produced the simplex method, another person named Jack Laderman solved the full problem in 1947 using it!

The computation required approximately 120 person-days using desk calculators.

Source: Google OR-Tools, *The Stigler Diet Problem*.

Setting up the Stigler Diet Problem

We use the historical data set from **Google OR-Tools**:

$$77 \text{ foods}, \qquad 9 \text{ nutritional requirements}.$$

The nutrient values are given **per dollar spent**, so let

$$x_i = \text{dollars spent on food } i, \qquad i = 1, \ldots, 77.$$

Therefore, the total cost is

$$\boxed{\min \sum_{i=1}^{77} x_i}$$

and hence

$$c = \begin{pmatrix} 1, & 1, & , \cdots , & 1 \end{pmatrix}^T.$$

Source: Google OR-Tools, *The Stigler Diet Problem*.

Putting the Problem into `linprog`

The nutritional constraints can be written as

$$Ax \geq b,$$

where

$$A \in \mathbb{R}^{9\times 77}, \qquad b \in \mathbb{R}^{9}.$$

Since `linprog` expects inequalities of the form

$$Ax \leq b,$$

we simply rewrite

$$\boxed{Ax \geq b \quad \Longleftrightarrow \quad -Ax \leq -b}$$

together with

$$x \geq 0.$$

Now we solve the 77-variable problem in MATLAB.

Why look beyond MATLAB?

MATLAB and `linprog` are excellent for solving linear programs, especially when written in matrix form

Advantages of MATLAB:

- Natural connection to linear algebra and matrix notation
- Easy to use for numerical computation and visualization
- Convenient for small and medium-sized LPs

However, as models become more complicated, we often have to spend more time manually constructing

$$A, \qquad b, \qquad c$$

and converting constraints into the exact form expected by the solver.

This motivates a different approach:

What if we could write the optimization problem almost exactly as it appears mathematically?

Method 2: CVXPY

CVXPY is a Python-based modeling language for convex optimization developed at Stanford and now maintained by a broader community.

The main difference is that we can describe an optimization problem much more closely to how we write it mathematically. We do not need to convert the objective and constraint into matrix form.

CVXPY then transforms the model into an appropriate form and passes it to a numerical solver.

Download CVXPY: `https://www.cvxpy.org/install/`

CVXPY Example: Linear Programming

Mathematical formulation

$$
\begin{aligned}
\text{maximize} \quad & 3x_1 + 5x_2 \\
\text{subject to} \quad & x_1 + 2x_2 \leq 8, \\
& 3x_1 + 2x_2 \leq 12, \\
& x_1 \geq 0, \\
& x_2 \geq 0.
\end{aligned}
$$

CVXPY implementation

```python
import cvxpy as cp
import numpy as np

x_1 = cp.Variable(1)
x_2 = cp.Variable(1)

objective_function = cp.Maximize(3 * x_1 + 5 * x_2)

constraints = [
    x_1 + 2 * x_2 <= 8,
    3 * x_1 + 2 * x_2 <= 12,
    x_1 >= 0,
    x_2 >= 0
]

problem = cp.Problem(objective_function, constraints)

objective_value = problem.solve(solver='Scipy')

#Display objective values
print("Optimal x_1 =", x_1.value)
print("Optimal x_2 =", x_2.value)
print("Optimal objective value =", objective_value)
```

Why should we use CVXPY?

For a simple linear program, `linprog` and CVXPY can solve essentially the same optimization problem.

The main difference is **how we describe the model**.

MATLAB `linprog`	**CVXPY**
Works directly with A, b, c	Code closely follows the mathematical formulation
Constraints must be converted into the required solver form	Constraints can be written more naturally
Designed primarily for linear programming	Extends naturally to broader classes of optimization problems

As models become larger or more complicated, CVXPY often makes the formulation easier to write, read, and modify.

Convexity and Concavity

Recall the definition of *convexity* from class, where we said that for a function $f : \mathbb{R}^n \to \mathbb{R}$, f is **convex** if, for all x, y and $0 \leq \lambda \leq 1$,

$$f(\lambda x + (1 - \lambda)y) \leq \lambda f(x) + (1 - \lambda)f(y).$$

Similarly, f is **concave** if

$$f(\lambda x + (1 - \lambda)y) \geq \lambda f(x) + (1 - \lambda)f(y).$$

A function of the form $f(x) = c^T x + d$ is called an *affine* function, which is a linear function plus some offset d
An affine function is both convex and concave

Therefore, linear programs naturally fall inside the broader class of **convex optimization problems**.

Introduction to Convex Optimization

Convex optimization is a subfield of optimization where the objective function is convex and the feasible region is convex.

These problems are especially important because they have strong mathematical structure:

- any local minimum is also a **global minimum**,
- many convex problems can be solved efficiently and reliably,
- linear programming is a special case of convex optimization.

This broader framework is what allows tools such as CVXPY to handle much more than just linear programs.

Convex optimization is also widely used in applications including **quantitative finance, aerospace and control, signal processing, machine learning,** and many other engineering and scientific fields.

Disciplined Convex Programming (DCP)

The next natural question is: how does CVXPY know whether the problem we have written has the right mathematical structure?

CVXPY uses a framework called **Disciplined Convex Programming (DCP)**.

DCP is a set of rules that allows CVXPY to determine whether the optimization problem we have written is **convex**.

To do this, CVXPY keeps track of whether expressions are affine, convex, or concave and checks whether they are combined in a valid manner

If the model satisfies these rules, CVXPY can transform the problem into a standard form and pass it to a numerical solver.

Linear programs automatically satisfy the DCP framework.

Read more about DCP: `dcp.stanford.edu`

The Stigler Diet Problem using CVXPY

Let's solve same 77-variable diet problem we did with MATLAB, now using CVXPY.

Recall the model we had was:

$$\min \sum_{i=1}^{77} x_i$$

subject to

$$Ax \geq b, \qquad x \geq 0.$$

Unlike `linprog`, CVXPY allows us to write these constraints directly in the same form as the mathematical model using what are called *CVXPY atoms*. A list of all their atoms can be found here:
`https://www.cvxpy.org/api_reference/cvxpy.atoms.html`

Beyond Linear Programming

CVXPY is not limited to linear objectives and constraints.

Many important optimization problems are **nonlinear**, but still **convex**. For example:

- **Least Squares**

$$\min_x \|Ax - b\|_2^2$$

- **Quadratic Programming**

$$\min_x \frac{1}{2} x^T Q x + c^T x, \qquad Q \succeq 0$$

The same CVXPY framework can be used for all of these problems.

We will learn more about CVXPY for non-linear programming when we cover it in class.

Solver: Gurobi

Gurobi is a high-performance optimization solver used extensively in research and industrial applications

It can solve several classes of optimization problems, including LPs, QPs, and Mixed-integer programs among others
Gurobi provides interfaces in several languages, including

Python, MATLAB, C/C++, Java, R.

For Purdue students and faculty, Gurobi provides **free full-featured academic licenses**. Download the license for Gurobi at the link below.

Read more: `https://www.gurobi.com/academics/`

Modeling Language vs. Solver

It is useful to distinguish between a **modeling language** and a **solver**.

$$\boxed{\text{Mathematical Model} \longrightarrow \text{Modeling Interface} \longrightarrow \text{Solver}}$$

For example,

$$\text{CVXPY} \qquad \longrightarrow \qquad \text{Gurobi}$$

is one possible workflow.

- **CVXPY** helps us express the optimization problem.
- **Gurobi** contains the numerical algorithms that actually solve the resulting model.

Gurobi can also be used directly through its own APIs, for example through Python using `gurobipy`.

Gurobi Example: Linear Programming

Mathematical formulation

$$
\begin{aligned}
\text{maximize} \quad & 3x_1 + 5x_2 \\
\text{subject to} \quad & x_1 + 2x_2 \leq 8, \\
& 3x_1 + 2x_2 \leq 12, \\
& x_1 \geq 0, \\
& x_2 \geq 0.
\end{aligned}
$$

`gurobipy` implementation

```
import gurobipy as gp
from gurobipy import GRB

model = gp.Model("LP_example")

x_1 = model.addVar(lb=0, name="x_1")
x_2 = model.addVar(lb=0, name="x_2")

model.setObjective(
    3*x_1 + 5*x_2,
    GRB.MAXIMIZE
)

model.addConstr(
    x_1 + 2*x_2 <= 8
)

model.addConstr(
    3*x_1 + 2*x_2 <= 12
)

model.optimize()

print("x_1 =", x_1.X)
print("x_2 =", x_2.X)
print("Objective =", model.ObjVal)
```

More Optimization Tools

The tools we have seen today represent several different ways of working with optimization problems:

- `linprog`: a direct, matrix-based interface for solving LPs
- **CVXPY**: a high-level modeling language that closely follows the mathematical formulation
- **Gurobi**: a high-performance numerical optimization solver, which also has capabilities to model the problem

There are many other tools you may encounter, including

- **Google OR-Tools**: open-source optimization toolkit
- **JuMP**: optimization modeling language for Julia
- **HiGHS**: open-source solver for LPs and related problems

The software may change, but the central task remains the same: formulate the mathematical model correctly and choose an appropriate tool to solve it.

Course Project Ideas

The following are possible **starting points** for the course project. The project should go beyond simply calling an optimization solver.

- **Smoothed Analysis of Linear Programming Algorithms**
  Study worst-case, average-case, and *smoothed* analysis for the simplex method, and investigate how smoothed analysis helps explain its strong practical performance despite poor worst-case complexity.

- **Compare Linear Programming Algorithms**
  Perform an experimental study comparing simplex and interior-point methods on problems with different sizes and structures.

- **Large-Scale / Sparse Linear Programs**
  Study how problem size and sparsity affect runtime, memory usage, and numerical performance of LP Algorithms.

Course Project Ideas

- **Application-Based Modeling**
  Formulate and analyze a substantial LP arising from an application of your choice. Study how the solution changes as assumptions, parameters, or constraints are modified.
- **Advanced: The Ellipsoid Method**
  Study the theory behind the ellipsoid method and compare it with simplex and interior-point methods computationally and theoretically.

A strong computational project should typically contain

$$\boxed{\text{Model} + \text{Implementation} + \text{Experiments} + \text{Analysis}}$$

These ideas are only starting points, you need to think deeper and develop this into a substantial study. More project ideas will come up as we study more material in the course, and we will come up with more ideas then as well.

Resources & References

Linear Programming / Solvers

- **Vanderbei, R. J.**
  Linear Programming: Foundations and Extensions
- **MATLAB** `linprog`
  MathWorks Documentation
- **Gurobi**
  Official Documentation
- **Stigler Diet Problem**
  Google OR-Tools Example
- **Google OR-Tools**
  Optimization Documentation

Convex Optimization / CVXPY

- **CVXPY Documentation**
  cvxpy.org
- **CVXPY Examples**
  cvxpy.org/examples
- **Disciplined Convex Programming**
  dcp.stanford.edu
- **Boyd & Vandenberghe**
  Convex Optimization
  Free online textbook

Main Takeaways

Today we saw how Linear Programming moves from theory to computation:

$$\boxed{\text{Model} \longrightarrow \text{Formulate} \longrightarrow \text{Solve} \longrightarrow \text{Interpret}}$$

- `linprog` provides a direct, matrix-based approach to solving Linear Programs.
- CVXPY lets us describe optimization problems closer to their mathematical formulation.
- Gurobi illustrates the distinction between **modeling software** and the **solver** used underneath.
- Computational tools allow LP theory to scale to large applied problems and extend naturally into broader convex optimization problems.

Questions?