Lesson 7 – Solving Optimization with SciPy

What is SciPy?

SciPy is a popular Python library used for math, science, and engineering. It has a tool called `linprog` that lets us solve optimization problems like the ones we've been modeling.
Learn more: https://scipy.org/ | Docs for `linprog`: https://docs.scipy.org/doc/scipy/reference/generated/scipy.optimize.linprog.html

Using SciPy in Python

- **Google Colab:** SciPy is already installed.
- **Local Python:** install once with `pip install scipy`.
- **Import it in your code:**

```
from scipy.optimize import linprog
```

From Problem to Model to Code

Before using `linprog`, always write out:

1. **Variables:** what are we deciding? (e.g., how many breads or cakes)
2. **Objective:** what do we want to do? (maximize profit or minimize cost)
3. **Constraints:** what limits or rules must we follow?
4. **Bounds:** are the variables positive or limited?

Once you have that, you can translate it into Python lists for `linprog`.

How `linprog` reads your problem

- **c:** numbers from the objective function (the profits or costs). Example: if cost $= 2x_1 + 5x_2$, then `c = [2, 5]`.

- **A_ub and b_ub:** describe the constraints that have "$\leq$". Example: if $2x_1 + 4x_2 \leq 40$, then `A_ub = [[2, 4]]`, `b_ub = [40]`.
- **bounds:** limits for each variable, for example "$-1 \leq x \leq 10$" corresponds to `(-1, 10)`, and "$x \geq 0$" corresponds to `(0, None)`.

Maximization vs. Minimization (The Simple Trick)

The `linprog` solver always tries to **minimize**. So if your problem says **maximize**, we can flip it around:

Easy rule

To turn a **maximize** problem into a **minimize** one, multiply the objective by -1. After solving, multiply the result by -1 again to get your original maximum value.

Example:

$$\text{Maximize } 3x_1 + 2x_2$$

We tell the solver to minimize the opposite:

$$\text{Minimize } -3x_1 - 2x_2$$

That's why we write:

```
c = [-3, -2] % negate to turn "max" into "min"
```

When you print `res.fun`, that's the minimum value of the negative version. So the real maximum is `-res.fun`.

Remember!

- Use negative signs for maximizing problems.
- Multiply $\geq$ constraints by -1 to make them $\leq$.
- Keep variables nonnegative unless stated otherwise.

Example 1: Bakery Problem (maximize profit)

A local bakery produces two popular products: loaves of bread (x_1) and cakes (x_2). Each loaf of bread earns a profit of \$2, while each cake earns a profit of \$5. However, both products require oven time, each loaf needs 2 hours, and each cake needs 4 hours. Since the bakery only has 40 hours of oven time available each week, it must decide how many loaves and cakes to bake to maximize its profit.

Model:

- **Variables:** x_1 = number of loaves of bread, x_2 = number of cakes
- **Objective:** maximize profit $2x_1 + 5x_2$
- **Constraint:** oven time $2x_1 + 4x_2 \leq 40$
- **Bounds:** $x_1, x_2 \geq 0$

Python translation:

```python
from scipy.optimize import linprog

# Maximization -> negate the coefficients
# we maximize 2x1 + 5x2 by minimizing -2x1 - 5x2
c = [-2, -5]

# Inequality constraint: 2x1 + 4x2 <= 40 (resource limit)
A_ub = [[2, 4]]
b_ub = [40]

#Bounds for the variables: x1 >= 0 and x2 >= 0 (you can't make a negative number of
    items)
bounds = [(0, None), (0, None)]

# This line solves the optimization problem (finds the best values of x1 and x2).
# The word "highs" tells Python to use a fast, modern solving method.
# There are also older methods like "simplex" or "interior-point", but "highs" is the
    one we use now.
res = linprog(c, A_ub=A_ub, b_ub=b_ub, bounds=bounds, method="highs")

print("Optimal (x1, x2):", res.x)
print("Max profit:", -res.fun) # flip sign back
```

Output

```
Optimal (x1, x2): [20  0]
Max profit: 40.0
```

Example 2: Transportation (minimize cost)

A delivery company must decide how to transport products using two options: **trucks** (x_1) and **trains** (x_2). Each truck shipment costs \$3, and each train shipment costs \$5. The company needs to send at least 10 shipments in total but wants to spend as little money as possible.

Model:

- **Variables:** x_1 = trucks shipment, x_2 = trains shipment
- **Objective:** minimize cost $3x_1 + 5x_2$
- **Constraint:** need at least 10 shipments ($x_1 + x_2 \geq 10$)
- **Bounds:** $x_1, x_2 \geq 0$

Turning the constraint into $\leq$:

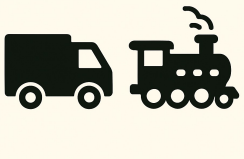
$$x_1 + x_2 \geq 10 \quad \text{becomes} \quad -x_1 - x_2 \leq -10$$

Python translation:

```python
from scipy.optimize import linprog

c = [3, 5]
A_ub = [[-1, -1]]
b_ub = [-10]
bounds = [(0, None), (0, None)]

res = linprog(c, A_ub=A_ub, b_ub=b_ub, bounds=bounds, method="highs")

print("Optimal (x1, x2):", res.x)
print("Minimum cost:", res.fun)
```

Output

```
Optimal (x1, x2): [10.  0.]
Minimum cost: 30.0
```



Practice Problems

For each problem:

1. Write the variables, objective, and constraints.
2. Turn the model into Python lists: `c`, `A_ub`, `b_ub`, and `bounds`.
3. Solve with `linprog`.

1. **Snack Mix (max):** A small snack shop is making trail mix with **nuts** (x_1) and **raisins** (x_2). Each bag of nuts gives a profit of \$1, and each bag of raisins gives a profit of \$2. Because of

limited packaging space, the shop must follow this constraint:

$$x_1 + 2x_2 \leq 8$$

2. **Study Hours (max):** A student has 10 hours. Each hour on Subject 1 (x_1) gives 4 points, Subject 2 (x_2) gives 3 points. Maximize total points.

3. **Toy Factory (max):** A small toy factory produces two kinds of toys: **Toy A** (x_1) and **Toy B** (x_2). Each Toy A brings a profit of \$5, while each Toy B brings a profit of \$7. The factory is limited by its resources, which must satisfy the following constraints:

$$3x_1 + 4x_2 \leq 60, \quad x_1 + x_2 \leq 20$$