Linear Optimization Models

Shixuan Zhang

ISEN 320-501, Fall 2023

1 Linear Optimization Formulations

A linear optimization (LO) model is a constrained optimization model with continuous variables, affine linear constraints, and a linear objective function. Recall that a function $f : \mathbb{R}^n \to \mathbb{R}$ is called affine linear if there exist $f_0, f_1, \ldots, f_n \in \mathbb{R}$ such that

$$f(x) = f_0 + f_1 x_1 + \cdots + f_n x_n,$$

for any $(x_1, \ldots, x_n) \in \mathbb{R}^n$. It is further *linear* if $f_0 = 0$. Thus given problem data $c_1, \ldots, c_n, b_1, \ldots, b_m \in \mathbb{R}$, and $a_{ij} \in \mathbb{R}$ for $i = 1, \ldots, n$ and $j = 1, \ldots, m$, a simple form of linear optimization can be formulated by

$$\begin{array}{rrcccl} \min \,/\, \max & c_1 x_1 & + & \cdots & + c_n x_n & \\ \text{s.t.} & a_{11} x_1 & + & \cdots & + a_{1n} x_n & \leq b_1 \\ & \vdots & & \ddots & \vdots & \vdots \\ & a_{m1} x_1 & + & \cdots & + a_{mn} x_n & \leq b_m \\ & x_1, & & \cdots & x_n & \in \mathbb{R} \end{array} \tag{1}$$

or simply as

$$\begin{aligned} \min \,/\, \max \quad & \sum_{i=1}^{n} c_i x_i \\ \text{s.t.} \quad & \sum_{i=1}^{n} a_{ji} x_i \leq b_j, \quad j = 1, \ldots, m, \\ & x_i \in \mathbb{R}, \quad i = 1, \ldots, n. \end{aligned}$$

An alternative way is to use a matrix and vectors

$$A = \begin{bmatrix} a_{11} & \cdots & a_{1n} \\ \vdots & \ddots & \vdots \\ a_{m1} & \cdots & a_{mn} \end{bmatrix}, \quad b = \begin{bmatrix} b_1 \\ \vdots \\ b_m \end{bmatrix}, \quad c = \begin{bmatrix} c_1 \\ \vdots \\ c_n \end{bmatrix}, \tag{2}$$

and write our LO model more compactly as

$$\begin{aligned} \min \text{ / } \max \quad & c^\mathsf{T} x \\ \text{s.t.} \quad & Ax \le b, \\ & x \in \mathbb{R}^n. \end{aligned} \tag{3}$$

Here, the convention is that for any two vectors $u = (u_1, \ldots, u_n), v = (v_1, \ldots, v_n) \in \mathbb{R}^n$, we write

$$u \le v \iff u_i \le v_i, \quad i = 1, \ldots, n.$$

We claim that any linear optimization problem in general can be transformed into the above simple form (1) or (2). We have already seen that an inequality constraint $Ax \ge b$ can be rewritten as $-Ax \le -b$. When we have equality constraints, we can always reformulate them as

$$A^{\mathsf{eq}} x = b^{\mathsf{eq}} \iff \begin{bmatrix} A^{\mathsf{eq}} \\ -A^{\mathsf{eq}} \end{bmatrix} x \le \begin{bmatrix} b^{\mathsf{eq}} \\ -b^{\mathsf{eq}} \end{bmatrix}. \tag{4}$$

When we have explicit bounds on variables, for example,

$$x_i^{\mathsf{lb}} \le x_i \le x_i^{\mathsf{ub}}, \quad i = 1, \ldots, n, \tag{5}$$

we can also write them as inequality constraints, with $e_i \in \mathbb{R}^n$ that has 1 in its ith component and 0 in all other components,

$$\begin{bmatrix} -e_i^\mathsf{T} \\ e_i^\mathsf{T} \end{bmatrix} x \le \begin{bmatrix} -x_i^{\mathsf{lb}} \\ x_i^{\mathsf{ub}} \end{bmatrix}. \tag{6}$$

Therefore, the formulation (1) or (2) is a conceptually simple way of writing *any* LO model. In practice, however, it is not always necessary that we convert a LO model into such formulation.

Example 1. *A farmer wants to determine how many acres of corn and wheat to plant this year. Related information is given as follows.*

- *An acre of wheat yields 25 bushels of wheat and requires 10 hours of labor per week.*
- *An acre of corn yields 10 bushels of corn and requires 4 hours of labor per week.*
- *All wheat and corn can be sold at \$4 a bushel.*
- *Seven acres of land and 40 hours of labor per week are available.*
- *Government regulations require that at least 30 bushels of corn be produced.*

The goal is to maximize the total revenue. To formulate it as a LO model, let x_1 denote the number of acres of corn to plant and x_2 the number of acres of wheat to plant this year, both of

which are continuous and nonnegative

$$x_1, x_2 \geq 0 \iff -x_1 \leq 0, -x_2 \leq 0.$$

Their upper bounds are not explicitly given to us. For the first constraint, we notice that our land area is limited

$$x_1 + x_2 \leq 7.$$

We write the labor time restriction as our second constraint

$$4 \cdot x_1 + 10 \cdot x_2 \leq 40.$$

The government regulations on corn production gives us the third constraint

$$10 \cdot x_1 \geq 30 \iff -x_1 \leq -3.$$

Note that the variable bound $-x_1 \leq 0$ is implied by this constraint and can thus be discarded. The objective is to maximize the revenue, which is

$$\max \quad 4 \cdot 10 \cdot x_1 + 4 \cdot 25 \cdot x_2.$$

This LO model can also be written in the matrix form (2) with

$$A = \begin{bmatrix} 1 & 1 \\ 4 & 10 \\ -1 & 0 \\ 0 & -1 \end{bmatrix}, \quad b = \begin{bmatrix} 7 \\ 40 \\ -3 \\ 0 \end{bmatrix}, \quad c = \begin{bmatrix} 40 \\ 100 \end{bmatrix}.$$

2 Computer Modeling Tools and Solvers

We have seen two mathematically equivalent ways of writing a LO model. However, depending on the input problem data, sometimes it is more convenient to use the first way (1), rather than the second one (2), in terms of computer programming. We will mainly use `Python 3.11` for our computer programming. There are two types of computer tools used for optimization problems:

- modeling interface, which facilitates model building by providing variable and constraint handles, and sometimes also a domain-specific language for constraints and objective expressions;
- underlying solver, which takes the problem data and executes appropriate numerical algorithms that aim to find optimal solutions.

For this course, we use the following two `Python` packages: `OR-Tools 9.5` and `SciPy 1.11`.

The installation guides can be found on `https://developers.google.com/optimization/install/python` and `https://scipy.org/install/`, respectively. The package `OR-Tools` provides a full modeling interface to LO and mixed-integer linear optimization (MILO) problems. Its installation automatically includes some open-source solvers, such as `GLOP` and `PDLP`, and it also connects to proprietary solvers such as `Gurobi` and `CPLEX`. The package `SciPy` is a popular scientific computing `Python` package, which provides a thin wrapper of a powerful open-source LO and MILO solver `HiGHS`. Next we briefly describe and compare ways of building and solving LO models using `OR-Tools` and `SciPy`.

We illustrate the modeling using Example 1 and begin with `OR-Tools`, which is often the more convenient one to use, especially for beginners. The module can be loaded as follows.

```
from ortools.linear_solver import pywraplp
```

Next we declare a LO solver, `GLOP`.

```
solver = pywraplp.Solver.CreateSolver("GLOP")
```

To define continuous variables, we can use the `NumVar` function in the `Solver` class.

```
x1 = solver.NumVar(0.0, solver.infinity(), "x1")
x2 = solver.NumVar(0.0, solver.infinity(), "x2")
```

The first two arguments are the lower and upper bounds of the defined variable (as defind in (5)). Here, we set the lower bounds to be 0.0, and because we do not know any upper bound of the variables, we put `solver.infinite()` (i.e., $+\infty$) as the second argument. The third argument, `"x1"` or `"x2"`, is the variable name, which can be of help if we want to debug or export the model later. The constraints in Example 1 can be coded as follows.

```
solver.Add(x1 + x2 <= 7)
solver.Add(4*x1 + 10*x2 <= 40)
solver.Add(-x1 <= -3)
```

To set the objective function, we can use the following code.

```
solver.Maximize(40*x1 + 100*x2)
```

Now we are ready to let the solver solve this model.

```
status = solver.Solve()
```

The `status` stores the information returned by the solver, which can be used to check whether we have found an optimal solution.

```
if status == pywraplp.Solver.OPTIMAL:
    print("Optimal value =", solver.Objective().Value())
```

```
        print("x1 =", x1.solution_value())
        print("x2 =", x2.solution_value())
    else:
        print("The solver is unable to find an optimal solution.")
```

After executing the script, an output is displayed below.

```
Optimal value = 399.99999999999994
x1 = 3.0
x2 = 2.7999999999999994
```

As `OR-Tools` connects to different solvers, we can replace the definition of `solver` in the `OR-Tools` model with the following line.

```
solver = pywraplp.Solver.CreateSolver("Clp")
```

Without any changes to other parts, the output could become the following.

```
Optimal value = 400.0
x1 = 5.0
x2 = 2.0
```

This shows us that calling different underlying solvers may give different optimal solutions, even when the model is unchanged.

Given the problem data matrix and vectors, we can also use `SciPy` to solve this LO model. The modules can be loaded using the following code.

```
import numpy as np
from scipy.optimize import linprog
```

The problem data can be coded as follows.

```
c = np.array([40, 100])
b = np.array([7, 40, -3, 0])
A = np.array([[1, 1],
              [4, 10],
              [-1, 0],
              [0, -1]])
```

Note that `SciPy` by default only takes minimization problems, so we call the `linprog` function with arguments `-c`, `b`, and `A`, to solve the LO model.

```
result = linprog(-c, A_ub=A, b_ub=b)
```

Here, keywords `A_ub` and `b_ub` refer to the inequality constraints in (2). When the problem has equality constraints (as in (4)) and variable bounds (as in (5)), they can be directly added using keywords `A_eq`, `b_eq`, and `bounds`. We now retrieve the results (with the opposite of the optimal value).

```
print(result.message)
print("Optimal value =", -result.fun)
print("[x1 x2] =", result.x)
```

The output is displayed below.

```
Optimization terminated successfully. (HiGHS Status 7: Optimal)
Optimal value = 400.0
[x1 x2] = [3.  2.8]
```

To summarize, compared with `SciPy`, the package `OR-Tools` allows

(i) adding constraints by directly using variable handles, without the need to write down the matrix representation;

(ii) both minimization and maximization;

(iii) an easy change of the underlying solver.

However, `SciPy` can be more lightweight sometimes and connects to the solver `HiGHS`. One should choose their computer tools depending on the problem and the purpose of use. For this course, we will mostly rely on `OR-Tools` for simplicity.

3 More Examples and Models

Example 2. *A student aims to improve their diet. Based on a nutrition specialist recommendation, they want their daily intake to contain at least 60 g of protein, 800 mg of calcium, 75 mg of vitamin C, and 2,000 calories. They would like to find a least expensive menu consisting of five food types: almond butter, brown rice, orange juice, salmon, and wheat bread. The serving size, cost per serving, and nutrition information for each food type is provided in the table below.*

Food type	Cost ($)	Protein (g)	Calcium (mg)	Vitamin C (mg)	Calories
Almond butter (100 g)	2.90	15	270	1	600
Brown rice (200 g)	3.20	5	20	0	215
Orange juice (250 g)	0.50	2	25	106	110
Salmon (150 g)	4.50	39	23	0	280
Wheat bread (25 g)	0.30	3	35	0	66
Required ingestion	-	60	800	75	2,000

We define decision variables for the amount of each food type to be consumed daily, all of which are nonnegative:

$x_1 \geq 0$: *servings of almond butter consumed daily,*
$x_2 \geq 0$: *servings of brown rice consumed daily,*
$x_3 \geq 0$: *servings of orange juice consumed daily,*
$x_4 \geq 0$: *servings of salmon consumed daily,*
$x_5 \geq 0$: *servings of wheat bread consumed daily.*

The constraints express the minimum daily requirements for protein

$$15x_1 + 4x_2 + 2x_3 + 39x_4 + 3x_5 \geq 60,$$

for calcium

$$270x_1 + 20x_2 + 25x_3 + 23x_4 + 35x_5 \geq 800,$$

for vitamin C

$$x_1 + 106x_3 \geq 75,$$

and for calories

$$600x_1 + 215x_2 + 110x_3 + 280x_4 + 66x_5 \geq 2000.$$

The objective is to minimize the cost, which is a linear function of the decision variables:

$$\min \quad 2.9x_1 + 3.2x_2 + 0.5x_3 + 4.5x_4 + 0.3x_5.$$

We program the model in the script `model_diet.py`*, and get the following output.*

```
The cost of a least expensive diet is $9.09.
The intake amount of each food type in the diet is shown below.
   Almond butter: 0.00 g
      Brown rice: 0.00 g
    Orange juice: 176.89 g
          Salmon: 0.00 g
     Wheat bread: 728.09 g
```

Example 3. *An investor is considering 6 projects for potential investment for the upcoming year. The required investment and end-of-year payout amounts are described in the following table. Partial investment (i.e., financing only a fraction of the project instead of the whole*

	Project					
	1	2	3	4	5	6
Investment ($·1000)	10	25	35	45	50	60
Payout ($·1000)	12	30	41	55	65	77

project) is allowed for each project, with the payout proportional to the investment amount. For

example, if the investor decides to invest \$5,000 in project 2, the corresponding payout will be \$30,000·(\$5,000/\$25,000)=\$6,000. There are \$100,000 available for investment.

We define variables

$$0 \leq x_i \leq 1\text{: fraction of project } i \text{ financed, for } i = 1, \ldots, 6.$$

The only constraint is the limit on the investment, which is

$$10x_1 + 25x_2 + 35x_3 + 45x_4 + 50x_5 + 60x_6 \leq 100.$$

The objective is to maximize the total payout:

$$\max \quad 12x_1 + 30x_2 + 41x_3 + 55x_4 + 65x_5 + 77x_6.$$

We program the model in the script `model_allocation.py`*, and get the following output.*

```
The maximum payout is $129166.67.
The investment on each project is shown below.
Project 0: $0.00
Project 1: $0.00
Project 2: $0.00
Project 3: $0.00
Project 4: $50000.00
Project 5: $50000.00
```

Some models are not immediately LO models, that is, there could be nonlinear constraints or integer variables, but they can be *reformulated* or *relaxed* as LO models.

Example 4. *A painter needs to complete a job that requires 50 gallons of brown paint and 50 gallons of gray paint. The required shades of brown and gray can be obtained my mixing the primary colors (red, yellow, and blue) in the proportions given in the following table. The same*

Color	Red	Yellow	Blue
Brown	40%	30%	30%
Gray	30%	30%	40%

shades can be obtained by mixing secondary colors (orange, green, and purple), each of which is based on mixing two out of three primary colors in equal proportions (red/yellow for orange, yellow/blue for green, and red/blue for purple). The painter currently has 20 gallons each of red, yellow, and blue paint, and 10 gallons each of orange, green, and purple paint. If needed, they can purchase any of the primary color paints for \$20 per gallon, however they would like to save by utilizing the existing paint supplies as much as possible.

We use indices $i = 1, \ldots, 6$, for red, yellow, blue, orange, green, and purple colors, respectively, and indices $j = 1, 2$, for brown and gray colors, respectively. Our decision variables can be defined as

$$x_{ij} \geq 0\text{: gallons of paint of color } i \text{ used to obtain color } j \text{ paint,}$$

for $i = 1, \ldots, 6$, $j = 1, 2$, and

$$y_i \geq 0\text{: gallons of paint of color } i \text{ purchased, } i = 1, 2, 3.$$

The total amount of brown and gray paint made must be at least 50 gallons each:

$$\sum_{i=1}^{6} x_{ij} \geq 50, \quad j = 1, 2.$$

The amount of paint used should not exceed its availability

$$\begin{aligned} x_{i1} + x_{i2} - y_i &\leq 20, \quad i = 1, 2, 3, \\ x_{i1} + x_{i2} &\leq 10, \quad i = 4, 5, 6. \end{aligned}$$

To express the constraints ensuring that the mixing yields the right shade of brown, note that only three out of six colors used for mixing contain red, and the total amount of red paint (including that coming from orange and purple paints) used in the brown mix is

$$x_{11} + 0.5x_{41} + 0.5x_{61}.$$

Hence, a constraint for the proportion of red color in the brown mix can be written as follows:

$$\frac{x_{11} + 0.5x_{41} + 0.5x_{61}}{\sum_{i=1}^{6} x_{i1}} = 0.4.$$

We can multiply both sides by $\sum_{i=1}^{6} x_{i1}$ and reformulate *it as a linear constraint*

$$0.6x_{11} - 0.4x_{21} - 0.4x_{31} + 0.1x_{41} - 0.4x_{51} + 0.1x_{61} = 0.$$

Similarly, the proportion of yellow and blue colors in the brown mix is given by:

$$\frac{x_{21} + 0.5_{41} + 0.5x_{51}}{\sum_{i=1}^{6} x_{i1}} = 0.3 \iff -0.3x_{11} + 0.7x_{21} - 0.3x_{31} + 0.2x_{41} + 0.2x_{51} - 0.3x_{61} = 0,$$

and

$$\frac{x_{31} + 0.5x_{51} + 0.5x_{61}}{\sum_{i=1}^{6} x_{i1}} = 0.3 \iff -0.3x_{11} - 0.3x_{21} + 0.7x_{31} - 0.3x_{41} + 0.2x_{51} + 0.2x_{61} = 0.$$

The constraints describing the proportion of each of the primary colors in the gray paint mix can be derived analogously:

$$\begin{aligned} 0.7x_{12} - 0.3x_{22} - 0.3x_{32} + 0.2x_{42} - 0.3x_{52} + 0.2x_{62} &= 0, \\ -0.3x_{12} + 0.7x_{22} - 0.3x_{32} + 0.2x_{42} + 0.2x_{52} - 0.3x_{62} &= 0, \\ -0.4x_{12} - 0.4x_{22} + 0.6x_{32} - 0.4x_{42} + 0.1x_{52} + 0.1x_{62} &= 0. \end{aligned}$$

Finally, we aim to minimize the cost of purchasing primary color paints:

$$\min \quad 20 \sum_{i=1}^{3} y_i.$$

We code the LO model in the script `model_mixing.py` *and the output is displayed below.*

```
The minimum paint cost is 200.00.
x11 = 15.0  x12 = 10.0
x21 = 10.0  x22 = 10.0
x31 = 15.0  x32 = 10.0
x41 = 10.0  x42 = 0.0
x51 = 0.0   x52 = 10.0
x61 = 0.0   x62 = 10.0
y1 = 5.0
y2 = 0.0
y3 = 5.0
```

Example 5. *A hospital uses a 12-hour shift schedule for its nurses, with each nurse working either day shifts (7:00 am-7:00 pm) or night shifts (7:00 pm-7:00 am). Each nurse works 3 consecutive day shifts or 3 consecutive night shifts and then has 4 days off. The hospital is aiming to design a schedule for day-shift nurses that minimizes the total number of nurses employed. The minimum number of nurses required for each day shift during a week is given in the following table:*

Day of week/shift	Nurses required
Monday (Mo)	16
Tuesday (Tu)	12
Wednesday (We)	18
Thursday (Th)	13
Friday (Fr)	15
Saturday (Sa)	9
Sunday (Su)	7

In addition, it is required that at least half of the day-shift nurses have weekends (Saturday and

Sunday) off.

Note that a nurse's schedule can be defined by the first day of the three-day cycle. Thus we define the decision variables as follows.

$x_1 \in \mathbb{Z}_{\geq 0}$: *number of nurses on Mo-Tu-We schedule*
$x_2 \in \mathbb{Z}_{\geq 0}$: *number of nurses on Tu-We-Th schedule*
$x_3 \in \mathbb{Z}_{\geq 0}$: *number of nurses on We-Th-Fr schedule*
$x_4 \in \mathbb{Z}_{\geq 0}$: *number of nurses on Th-Fr-Sa schedule*
$x_5 \in \mathbb{Z}_{\geq 0}$: *number of nurses on Fr-Sa-Su schedule*
$x_6 \in \mathbb{Z}_{\geq 0}$: *number of nurses on Sa-Su-Mo schedule*
$x_7 \in \mathbb{Z}_{\geq 0}$: *number of nurses on Su-Mo-Tu schedule*

Here, $x \in \mathbb{Z}_{\geq 0}$ means that $x \in \mathbb{Z}$ and $x \geq 0$. On Monday, there are $x_1 + x_6 + x_7$ nurses working, so by requirement we should have

$$x_1 + x_6 + x_7 \geq 16.$$

Similarly, for the other days of the week, we have constraints

$$\begin{aligned} x_1 + x_2 + x_7 &\geq 12, \\ x_1 + x_2 + x_3 &\geq 18, \\ x_2 + x_3 + x_4 &\geq 13, \\ x_3 + x_4 + x_5 &\geq 15, \\ x_4 + x_5 + x_6 &\geq 9, \\ x_5 + x_6 + x_7 &\geq 7. \end{aligned}$$

Clearly any of these constraints imply that $\sum_{i=1}^{7} x_i \geq 1$. Thus the requirement that half of the day-shift nurses have weekends off can be expressed as

$$\frac{x_1 + x_2 + x_3}{\sum_{i=1}^{7} x_i} \geq \frac{1}{2}.$$

As done in Example 4, we can multiply both sides by $2\sum_{i=1}^{7} x_i$, and rewrite this constraint as a linear one

$$x_1 + x_2 + x_3 - x_4 - x_5 - x_6 - x_7 \geq 0.$$

The objective is to minimize the total number of nurses $\sum_{i=1}^{7} x_i$*, so the model can be written as*

$$
\begin{aligned}
\min \quad & \sum_{i=1}^{7} x_i \\
\text{s.t.} \quad & x_1 + x_6 + x_7 \geq 16, \\
& x_1 + x_2 + x_7 \geq 12, \\
& x_1 + x_2 + x_3 \geq 18, \\
& x_2 + x_3 + x_4 \geq 13, \\
& x_3 + x_4 + x_5 \geq 15, \\
& x_4 + x_5 + x_6 \geq 9, \\
& x_5 + x_6 + x_7 \geq 7, \\
& \sum_{i=1}^{3} x_i - \sum_{i=4}^{7} x_i \geq 0, \\
& x_i \in \mathbb{Z}_{\geq 0}, \quad i = 1, \ldots, 7.
\end{aligned}
\qquad \xrightarrow{\text{relax}} \qquad
\begin{aligned}
\min \quad & \sum_{i=1}^{7} x_i \\
\text{s.t.} \quad & x_1 + x_6 + x_7 \geq 16, \\
& x_1 + x_2 + x_7 \geq 12, \\
& x_1 + x_2 + x_3 \geq 18, \\
& x_2 + x_3 + x_4 \geq 13, \\
& x_3 + x_4 + x_5 \geq 15, \\
& x_4 + x_5 + x_6 \geq 9, \\
& x_5 + x_6 + x_7 \geq 7, \\
& \sum_{i=1}^{3} x_i - \sum_{i=4}^{7} x_i \geq 0, \\
& x_i \geq 0, \quad i = 1, \ldots, 7.
\end{aligned}
$$

The original model (on the left) is not *a LO model due to the integrality conditions on the variables. Nevertheless, all of the constraints are affine linear and the objective function is also linear. We can thus* relax *the integrality conditions and get a LO model (on the right) by only imposing* $x_i \geq 0$ *for each* $i = 1, \ldots, 7$*. This relaxed LO model is coded in the script* `model_scheduling.py` *and returns the following result.*

```
The minimum number of nurses is 31.0.
x1 = 11.00
x2 = 0.00
x3 = 10.00
x4 = 3.00
x5 = 2.00
x6 = 4.00
x7 = 1.00
```

Note that although the integrality conditions were relaxed, the solver actually returns an integral *optimal solution to the LO model, which means that we have found an optimal solution to the original model.*

Example 6. *A company plans the monthly trampoline production quantities, where the demand during the next four months is*

$$d_1 = 110, \quad d_2 = 120, \quad d_3 = 130, \quad d_4 = 100.$$

Currently, the company has an inventory of 20 trampolines. During each month, it can manufacture up to 100 trampolines with regular-time labor for \$120 per unit. With overtime labor,

it can manufacture more trampolines, costing \$150 per unit. A per unit inventory cost of \$10 is charged at the end of each month. The warehouse can fit up to 25 trampolines. The management wants to develop a plan to minimize the total production and inventory costs. To build the model, we denote the index set for the planning horizon as $T := \{1,2,3,4\}$. *The decision variables are*

$x_t \geq 0$: *number of units made using regular-time labor during month* $t \in T$,
$y_t \geq 0$: *number of units made using overtime labor during month* $t \in T$,
$l_t \geq 0$: *inventory level at the end of month* $t \in T$.

while these variables should be integers in practice, we temporarily relax the integrality conditions to formulate a LO model. Note that

$$l_t = l_{t-1} + (x_t + y_t) - d_t, \quad t \in T,$$

where $l_0 = 20$. *Here, the inventory variables can be eliminated, but they often help understand and interpret the model. The total cost consists of three parts:*

- *regular-time production cost:* $120 \sum_{t=1}^{4} x_t$,
- *overtime production cost:* $150 \sum_{t=1}^{4} y_t$,
- *inventory cost:* $10 \sum_{t=1}^{4} l_t$.

Thus the model can be written as (with the parameter $l_0 = 20$*)*

$$\begin{aligned} \min \quad & 120 \sum_{t=1}^{4} x_t + 150 \sum_{t=1}^{4} y_t + 10 \sum_{t=1}^{4} l_t \\ \text{s.t.} \quad & x_t \leq 100, \quad t \in T, \\ & l_t = l_{t-1} + x_t + y_t - d_t, \quad t \in T, \\ & l_t \leq 25, \quad t \in T, \\ & x_t, y_t, l_t \geq 0, \quad t \in T. \end{aligned}$$

We code the LO model in the script `model_inventory.py` *and the output is displayed below.*

```
The minimum cost is $54100.00.
The regular-time, overtime labor, and inventory level in
each period are shown below.
Period 1:
regular-time labor is 100.00
overtime labor is 0.00
inventory level is 10.00
Period 2:
regular-time labor is 100.00
overtime labor is 10.00
inventory level is 0.00
```

```
Period 3:
regular-time labor is 100.00
overtime labor is 30.00
inventory level is 0.00
Period 4:
regular-time labor is 100.00
overtime labor is -0.00
inventory level is 0.00
```

The obtained solution is indeed integer-valued *so the solution is feasible and optimal even when we enforce the integrality conditions on the number of trampolines.*

Example 7. *A wholesale company specializing in one product has $m = 3$ warehouses W_i, $i = 1, \ldots, m$ serving $n = 4$ retail locations R_j, $j = 1, \ldots, n$. Transporting one unit of the product from W_i to R_j costs c_{ij} dollars, $i = 1, \ldots, m$, and $j = 1, \ldots, n$. The company has s_i units of product available to ship from W_i, $i = 1, \ldots, m$. To satisfy the demand, at least d_j units of the product must be delivered to R_j. The values of s_i, d_j, and c_{ij} for $i = 1, \ldots, m$ and $j = 1, \ldots, n$ are given by*

$$\begin{bmatrix} s_1 \\ s_2 \\ s_3 \end{bmatrix} = \begin{bmatrix} 40 \\ 50 \\ 60 \end{bmatrix}, \quad \begin{bmatrix} d_1 \\ d_2 \\ d_3 \\ d_4 \end{bmatrix} = \begin{bmatrix} 17 \\ 33 \\ 23 \\ 47 \end{bmatrix}, \quad \begin{bmatrix} c_{11} & c_{12} & c_{13} & c_{14} \\ c_{21} & c_{22} & c_{23} & c_{24} \\ c_{31} & c_{32} & c_{33} & c_{34} \end{bmatrix} = \begin{bmatrix} 3 & 2 & 1 & 1 \\ 2 & 3 & 5 & 4 \\ 3 & 5 & 7 & 8 \end{bmatrix}.$$

The goal is to find out how many units of the product should be shipped from each warehouse to each retail location so that the company's overall transportation costs are minimized. The decision variables are

$$x_{ij} \geq 0\text{: the product quantitiy shipped from } W_i \text{ to } R_j,\ i = 1, \ldots, m,\ j = 1, \ldots, n.$$

We need to make sure that the number of units shipped out of W_i does not exceed s_i

$$\sum_{j=1}^{n} x_{ij} \leq s_i,\ i = 1, \ldots, m.$$

To satisfy the demand at R_j, we must have

$$\sum_{i=1}^{m} x_{ij} \geq d_j,\ j = 1, \ldots, n.$$

The objective is to minimize the total cost of transportation, so the LO model can be written as

$$
\begin{aligned}
\min \quad & \sum_{i=1}^{m}\sum_{j=1}^{n} c_{ij}x_{ij} \\
\text{s.t.} \quad & \sum_{j=1}^{n} x_{ij} \le s_i, \quad i = 1,\ldots,m, \\
& \sum_{i=1}^{m} x_{ij} \ge d_j, \quad j = 1,\ldots,n, \\
& x_{ij} \ge 0, \quad i = 1,\ldots,m,\ j = 1,\ldots,n.
\end{aligned}
$$

We code this LO model in the script `model_transportation.py` *and the output is displayed below.*

```
The minimum shipment cost is 336.00.
The shipment plan is displayed below.
location\warehouse     1     2     3
        1           0.00  0.00 17.00
        2           0.00 20.00 13.00
        3          23.00  0.00  0.00
        4          17.00 30.00  0.00
```