Contents

2 *Optimization Modeling*

Optimization modeling is a systematic approach to solving complex problems by defining the problem, creating a mathematical model that represents the system or process, and finding the best solution to the model. At its core, optimization is the process of finding the best representation of the problems that can be solved using mathematical techniques (often needs to be simple enough), while also be as close as possible to the real-world problems to be useful for the business or engineering applications (often needs to be complex enough). Finding the right balance between simplicity and complexity is the key to successful optimization modeling.

2.1 *Basic Model Setup*

As illustrated in the previous chapter, we often have a design specifications and performance measures tailored to the problems we are trying to solve. These are often formalized as the *objective function* and *constraints*, all defined for a given set of *decision variables*.

Without loss of generality, let's consider a simple optimization problem for a single decision variable x. Our goal is to minimize an objective function $f(x)$, given that the feasible solution is within a feasible set $\mathcal{X}$. Mathematically, we can write this as:

$$\begin{aligned} \underset{x}{\text{minimize}} \quad & f(x) \\ \text{subject to} \quad & g(x) \leq 0, \\ & h(x) = 0, \\ & x \in \mathcal{X} \end{aligned} \tag{2.1}$$

Do not be intimidated by this mathematical notation. The goal of writing this as a mathematical model is to provide a clear and concise representation of the problem. We will abstract the mathematical representation to a more intuitive form in the following sections. For now, let's focus on understanding the basic components of the optimization model in equation (2.1).

- The *objective function* $f(x)$ is a mathematical function that we want to minimize. While we are focusing on minimization in this example (think of cost minimization, energy minimization, etc.), we can also maximize the objective function in other cases (think of profit maximization, user engagement maximization, etc.).

- The *decision variable* x is the parameter that we can adjust to optimize the objective function. For any given x, the objective function $f(x)$ will return a value that represents the performance measure we want to optimize. For our example, x is a single-dimension variable, and so we can simply try different values of x to see how the objective function changes and pick the one that gives the best performance (i.e. minimum value in this case). For more realistic models, we will need to use larger dimensional decision variables, and therefore trying out all possible combinations of the decision variables would be effectively impossible. Later, we will discuss how to systematically find the best value of x using optimization algorithms, which can scale up to large-dimensional x. We call the value of x that minimizes the objective function the *optimal solution* of the optimization problem and denote it as x^* (read as "x star").

- The *constraints* $g(x) \leq 0$ and $h(x) = 0$ are conditions that the solution must satisfy. These constraints can be related to the design specifications, physical limitations, or other requirements that the solution must meet. We can have multiple constraints that the solution must satisfy, and each constraint can be formulated as an inequality or equality that the decision variable x must satisfy.

- The *feasible set* $\mathcal{X}$ is the set of all possible values that x can take and still valid for the problem. This set can be defined by a set of constraints that the solution must satisfy. For example, if x represents the number of units to produce, then x must be a non-negative integer. In this case, the feasible set $\mathcal{X}$ is the set of all non-negative integers. It is also possible that the feasible set is defined by the constraints $g(x) \leq 0$ and $h(x) = 0$ that the solution must satisfy. In this case,

the feasible set $\mathcal{X}$ is the set of all x that satisfy the constraints. Under such a setting, we will not need to explicitly define the feasible set $\mathcal{X}$, as it is implicitly defined by the other constraints. We will see this in the examples later.

To better understand the essence of identifying the objective function, decision variables, constraints, and feasible set, let's consider a simple example as shown in exercise 2.1 and work through the process of modeling the optimization problem.

Exercise 2.1. PLX is an electricity provider that is looking to optimize its service levels to maximize its profits. The company has identified that the profit $f(x)$ (in millions of Rupiah) is a function of the service level x and can be expressed as

$$f(x) = -\frac{(x-95)^2 \cdot (x+100)^2}{1000} + 10000. \tag{2.2}$$

The company has also identified that the service level x is a continuous variable that can take any value between 0 and 100 for this profit function to be valid (see figure 2.1 for plot).

a. **Identify the objective function, constraints, and decision variables for an optimization model that can help PLX achieve its goal!**

b. **Based on figure 2.1, what value of service level x would lead to the highest profit for PLX?**

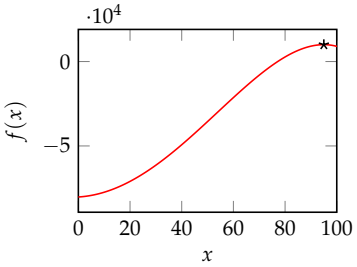

Figure 2.1: The profit $f(x)$ for PLX as a function of the service level x. The optimal service level x^* is marked with a star.

Solution:

a. The objective function is the profit $f(x)$ that PLX wants to maximize. The constraints are that the service level x must be a continuous variable between 0 and 100 (inclusive). The decision variable is the service level x. With this, we can write the optimization model as

$$\begin{aligned} \underset{x}{\text{maximize}} \quad & -\frac{(x-95)^2 \cdot (x+100)^2}{1000} + 10000 \\ \text{subject to} \quad & 0 \leq x \leq 100. \end{aligned} \tag{2.3}$$

As we will learn later, for optimization modeling purposes, we can solve an equivalent model that removes constants from the objective function and

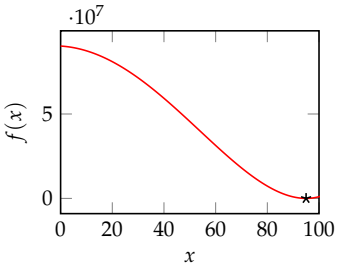

Figure 2.2: The objective function of the equivalent model equation (2.4).

turns it into a minimization problem (by multiplying the objective function by -1). For this case, the equivalent model is

$$\begin{aligned} \underset{x}{\text{minimize}} \quad & (x-95)^2 \cdot (x+100)^2 \\ \text{subject to} \quad & 0 \leq x \leq 100. \end{aligned} \tag{2.4}$$

This model is equivalent to the original model in equation (2.3), in the sense that the optimal solution x^* for this model will also be the optimal solution for the original model. See figure 2.2 for the objective function plot of the equivalent model (equation (2.4)).

b. Based on figure 2.1, the value of service level x that would lead to the highest profit for PLX is $x^* = 95$. This value is marked with a star in the plot and corresponds to the peak of the profit function $f(x)$ for x within the given range $[0, 100]$. The profit at this service level is $f(x^*) = 10000$ (million Rupiah).

An example of how to implement the model in Python using Pyomo is shown in figure 2.3 (for defining the decision variable x), figure 2.4 (for defining the objective function), and figure 2.5 (for defining the constraints). An example of how to solve this optimization model in Python using Pyomo and a nonlinear solver (IPOPT) is shown in figure 2.6. The full code is available in the course website.

```
1 import pyomo.environ as pyo
2
3 #initiate a model
4 model = pyo.ConcreteModel()
5
6 #set the decision variable x
7 model.x = pyo.Var(domain=pyo.Reals)
```

Figure 2.3: Defining the decision variable x for the PLX optimization model in Pyomo.

As illustrated in exercise 2.1, the optimization model is a compact representation of the problem that allows us to systematically define all the components of the problem. It is also a universal language that can be understood by both humans and computers. Having a model that is well-defined and structured allows us to write code to translate the model into a format that can be solved

```python
#objective function f(x) formulation
def f(x):
    return -((x - 95)**2 * (x + 100)**2)/1000 + 10000

#create an objective rule for pyomo
def obj_rule(model):
    return f(model.x)

#add the objective function to the model
model.obj = pyo.Objective(rule=obj_rule, sense=pyo.maximize)
```

Figure 2.4: Defining the objective function for the PLX optimization model in Pyomo.

```python
#feasibility constraints: x from 0 to 100
def x_geq_0(model):
    return model.x <= 100
def x_leq_100(model):
    return model.x >= 0

#add the constraints to the model
model.x_geq_0 = pyo.Constraint(rule=x_geq_0)
model.x_leq_100 = pyo.Constraint(rule=x_leq_100)
```

Figure 2.5: Defining the constraints for the PLX optimization model in Pyomo.

```python
#solve the model
solver=pyo.SolverFactory('ipopt', executable='/content/ipopt')
results = solver.solve(model)

#print the optimal solutions
fstar = pyo.value(model.obj)
xstar = pyo.value(model.x)

print(f"The optimal objective function value is {fstar}")
print(f"The optimal value of x is {xstar}")
```

Figure 2.6: Solving the PLX optimization model in Pyomo using IPOPT solver.

by optimization solvers. This is the essence of modern optimization modeling, where we use optimization modeling languages (e.g., Pyomo, GAMS, AMPL) to define the optimization model and optimization solvers (e.g., IPOPT, Gurobi, CPLEX) to solve the model. This separation of concerns between modeling and solving allows us to focus on the problem formulation and let the solvers handle the computational complexity of finding the optimal solution.

Before we dive deeper into optimization modeling, let's first understand the intuition behind optimization and how we can identify the optimal solution through visualization.

2.2 *Identifying the Optimal Solution through Visualization*

In exercise 2.1, we can solve the model by visually inspecting the objective function and identifying the optimal solution. This is possible because we have a simple model that involves a 1-d decision variable x with a straightforward feasible region $\mathcal{X}$ and objective function $f(x)$, giving a single peak over the feasible region. In cases where the interplay between the objective function and constraint is more complex, we can have multiple peaks or valleys, or even has no clear peak or valley. Determining the optimal solution in these cases can be more involved.

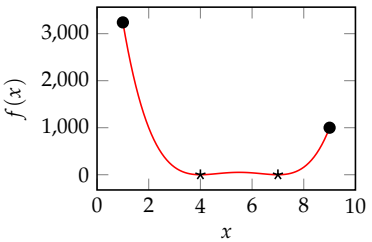

Figure 2.7: The objective function $f(x)$ with multiple optima ($x^* = 4$ and $x^* = 8$).

2.2.1 *Multiple Optimal Solutions*

In some cases, the optimization model may have multiple optimal solutions, where more than one value of the decision variable can minimize (or maximize) the objective function. This can happen when the objective function has multiple peaks or valleys over the feasible region, and the model can achieve the same objective function value for distinct decision variables. See figure 2.7 and figure 2.8 for examples.

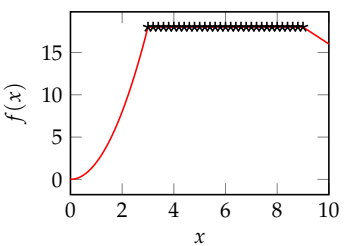

Figure 2.8: At the extreme, the objective function $f(x)$ can have infinitely many optima.

2.2.2 *Higher-Dimensional Decision Variables*

For models with 2D or 3D decision variables, we can still visualize the objective function (through contour plots[1] or surface plots[2]) and identify the optimal solution, although it may be more challenging to identify the optimal solution especially if the feasible region is complex (or simply large). As the problem gets more complex and involves larger dimensional decision variables, constraints,

[1] Contour plots are 2D plots that represent the objective function as a series of lines of equal value.

[2] Surface plots are 3D plots that represent the objective function in z dimension as a function of two decision variables x_1 and x_2.

and objectives, we need to develop more sophisticated optimization models and algorithms to solve them.

Exercise 2.2. Consider a simple optimization model with two decision variables x_1 and x_2 as follows:

$$\begin{aligned} \underset{x_1,x_2}{\text{minimize}} \quad & (x_1-2)^2 \cdot (x_2-3)^2 \cdot \left(x_1 x_2 - \frac{5}{2}\right) \\ \text{subject to} \quad & 1 \leq x_1 \leq 4, \\ & 1 \leq x_2 \leq 4. \end{aligned} \tag{2.5}$$

The objective function $f(x_1, x_2)$ is a simple polynomial function that represents the performance measure we want to optimize. The feasible region is defined by the constraints that x_1 and x_2 must be between 1 and 4. Figure 2.9 shows the surface plot of the objective function $f(x_1, x_2)$ and a set of lines that represent the feasible region. Based on this plot,

a. **Identify which region of the plot shows high and low objective function values!**

b. **Identify the optimal solution for this optimization model!**

c. There seems to be a few subregions that have lower objective function values than the marked optimal solution. **Why is the marked optimal solution still the best solution for this optimization model?**

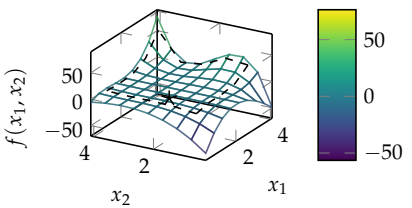

Figure 2.9: The surface plot for the equation (2.5).

Understanding contour plots and surface plots is crucial to help us identify the optimal solution and understand the behavior of the objective function over the feasible region. If you are curious, figure 2.10 shows the Pyomo implementation of the 2D optimization model in equation (2.5) and how to solve it using the IPOPT solver. Note that the formulation uses bounds on the decision variables x_1 and x_2 to define the feasible region, rather than explicitly defining the inequalities as constraints. The code is also available in the course website.

2.2.3 *Black-box Cases: When the Objective Function is Partially/Fully Unknown*

The intuition we have developed so far is based on the assumption that we have access to the objective function and can evaluate it at any given point. This is often

```
#initiate a model
model = pyo.ConcreteModel()

#define the set
model.I = pyo.Set(initialize=[1, 2])

#define the decision variables
model.x = pyo.Var(model.I, domain=pyo.Reals, bounds=(1, 4))

#define the objective function
def f(x):
    return (x[1]-2)**2 * (x[2]-3)**2 * (x[1]*x[2] -5/2)
#create an objective rule for pyomo
def obj_rule(model):
    return f(model.x)
#add the objective function to the model
model.obj = pyo.Objective(rule=obj_rule, sense=pyo.maximize)

#solve the model
solver=pyo.SolverFactory('ipopt', executable='/content/ipopt')
results = solver.solve(model)

#Display the optimal solutions
fstar = pyo.value(model.obj)
xstar = [pyo.value(model.x[i]) for i in model.I]
print(f"The optimal objective function value is {fstar}")
print(f"The optimal value of x is {xstar}")
```

Figure 2.10: Solving the 2D optimization model in Pyomo using IPOPT solver.

the case in simple optimization problems where the objective function is known and can be easily evaluated. However, in many real-world problems, the objective function is unknown, and we can only evaluate it at specific points (based on historical data) or even not at all. This is known as a *black-box optimization problem*, where the objective function is treated as a black box that we can query but do not know the exact form of the function.

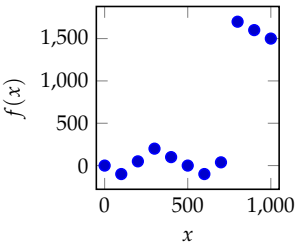

Figure 2.11: The objective function $f(x)$ for Qertamina's carbon capture and storage (CCS) project at different investment levels x.

Exercise 2.3. Qertamina is an oil and gas company that is looking to offset its carbon emissions by investing in carbon capture and storage (CCS) projects. The company has identified a potential CCS site in East Java, Indonesia. The company does not have the exact function of how much CO_2 can be captured based on the investment, but they have run a simulation model that can estimate the CO_2 capture for a given investment. The company has a budget of 1 billion Rupiah to invest in the CCS project. The goal is to maximize the net benefit of the project, which is the difference between the CO_2 capture economic value and the investment cost.

The model can be formulated as a black-box optimization problem, where the objective function $f(x)$ is the net benefit of the project, and the decision variable is the investment x amount. The feasible region is defined by the constraint that the investment must be between 0 and 1 billion Rupiah.

$$\begin{aligned} \underset{x}{\text{maximize}} \quad & f(x) = CO_2 \text{ capture economic value}(x) - x \\ \text{subject to} \quad & 0 \leq x \leq 1 \text{ billion Rupiah.} \end{aligned} \tag{2.6}$$

Qertamina has run a simulation model that estimates the CO_2 capture economic value for different investment levels $x \in \{0, 100, 200, \ldots, 1000\}$ million Rupiah. The plot is shown in figure 2.11.

a. **Given the available data, what is your best estimate of the optimal investment amount x^* for the CCS project?**

b. Suppose you can run a few more simulations to estimate the objective function at different investment levels but needs to be selective due to the high cost of running the simulation. **What strategy would you use to find the optimal investment amount x^*?**

Black-box optimization problems are common in many real-world applications, such as machine learning, engineering design, and financial modeling. In these cases, we need to develop optimization models and algorithms that can find the optimal solution without knowing the exact form of the objective function. This complication adds another layer of complexity to the optimization process, as we need to balance the exploration of the objective function space with the exploitation of the known information to find the best solution. An active area of research in optimization is to combine predictive modeling techniques with optimization algorithms to solve black-box optimization problems more efficiently and effectively. This is known as *surrogate-assisted optimization*, where we use a surrogate model (e.g., machine learning model) to approximate the objective function and guide the optimization process. This is beyond the scope of this chapter and will be covered in more advanced optimization courses.

2.3 *Systematic Approach to Modeling*

As the optimization problem becomes more complex and involves larger dimensional decision variables, and perhaps, of different types (e.g., continuous, integer, binary), the modeling process becomes a bit more challenging. It is useful to develop a systematic approach to modeling optimization problems to ensure that we capture all the essential components of the problem and represent them in a way that can be solved efficiently and effectively.

A systematic approach to modeling optimization problems involves the following steps:

1. *Define the problem*: Clearly define the problem and the goals of the optimization process. Identify the design specifications and performance measures that define the requirements and objectives of the problem. Be explicit in defining what you expect the model to answer.

2. *Identify the decision variables*: Identify the parameters that you can adjust to optimize the objective function. Often times, the decision variables consist of both the answer to the problem and some extra variables that can help us in the modeling and solving process. For example, many natural optimization problems can be modeled very simply as nonlinear optimization models. However, the solution process can be very difficult. By adding extra variables,

we can often simplify the solution process and turn the problem into a linear optimization models. We will discuss this in more detail in the next chapter.

3. *Formulate the objective function*: Define the objective function that represents the performance measure you want to optimize. This can be a mathematical function that quantifies the performance of the system or process based on the decision variables. The objective function should be a function of the decision variables and should be either minimized or maximized based on the problem requirements. Be cognizant of the trade-offs between different objectives and how they can be combined into a single objective function.

4. *Define the constraints*: Define the constraints that the solution must satisfy. These constraints can be related to the design specifications, physical limitations, or other requirements. Some constraints can be hard constraints that must be satisfied, while others can be soft constraints that can be violated to some extent. In some cases, the constraints can be formulated as a set of equations or inequalities that the decision variables must satisfy. In other cases, the constraints can be added as penalty terms to the objective function to guide the optimization process.

5. *Obtain the parameters and data*: Gather the parameters and data that are needed to define the model. This can include historical data, expert knowledge, experimental results, or other information that can help you define the objective function, constraints, and feasible region. The quality and quantity of the data can have a significant impact on the accuracy and reliability of the optimization model, so it is important to gather as much relevant information as possible.

6. *Verify and validate the model*: Verify that the model captures the logic of the system or process that you want it to represent. You might want to start with a small and simple model to verify that the model behaves as expected. Then, you can scale up the model to include more decision variables and constraints. It is often helpful to make the model modular and store the parameters in a data file that can be easily loaded into the model. This way, you can quickly test different scenarios and evaluate the model's performance. Once the full model is developed, validate the model by comparing the results to other possible solutions or benchmarks, analyzing the results, and interpreting the implications of the solutions. The goal of verification and validation is to ensure that the model is reliable and that the results are trustworthy.

Note that this systematic approach only provides a guideline to help you develop optimization models. The actual process of modeling an optimization problem can be more iterative and interactive, involving multiple stakeholders and experts in the field. It is also important to note that this guideline omits the process of solving the model. It can be the case that you have a good model but the solution process is too complex or too slow to be useful. In this case, you might need to simplify the model or develop more efficient algorithms to solve the model. We will dig into the solution process in the subsequent chapters. Our goal here is to provide you with a solid foundation in optimization modeling that you can build upon to solve more complex and challenging problems in your project.

2.4 *Production Planning Model Example*

Let's consider a few examples of models. We will start with simple models and gradually increase the complexity to illustrate the different components of the optimization process.

In this first example, we will consider a production planning problem where the goal is to determine the optimal production quantity for each product to maximize the total profit while satisfying budget and storage capacity constraints. The model is formulated as a linear optimization model, which is a common approach for production planning problems. In addition to the mathematical model, we will also implement the model in Python using the Pyomo library. This will help you understand how to translate the mathematical model into a computational model and solve it using optimization algorithms.

Sets

For this model, we have a set of products I that we want to produce. Each product has its own production capacity, profit per unit, and cost per unit. Let's assume for now that we have three products $I = \{1, 2, 3\}$, each with its own parameters (to be added in a bit). The implementation of the set I in Pyomo is shown in figure 2.12.

Parameters

- r_i: Revenue per unit of product i.

```
#initiate the model
model = pyo.ConcreteModel()

#define the set(s)
model.I = pyo.Set(initialize=[1, 2, 3])
```

Figure 2.12: Set definition in Pyomo.

- c_i: Cost per unit of product i.
- p_i: Production capacity of product i.
- b: Budget constraint for production.
- s: Storage capacity constraint.

Assuming we have loaded the data for each product into a dataframe say `df` and other scalar parameters into a dictionary say `params`, we can define the parameters for the model as shown in figure 2.13.

```
#define the parameters
model.r = pyo.Param(model.I, initialize=df['revenue'])
model.c = pyo.Param(model.I, initialize=df['cost'])
model.p = pyo.Param(model.I, initialize=df['production_capacity'])
model.b = pyo.Param(initialize=params['budget'])
model.s = pyo.Param(initialize=params['storage_capacity'])
```

Figure 2.13: Parameter definition in Pyomo.

Decision Variables

Our objective is to determine the production quantity of each product to maximize the total profit. Therefore, the decision variables for this model are:

- x_i: Production quantity of product i

The decision variables can be defined in Pyomo as shown in figure 2.14.

```
#define the decision variables
model.x = pyo.Var(model.I, domain=pyo.Integers)
```

Figure 2.14: Decision variable definition in Pyomo.

Objective Function

The objective function for this model is to maximize the total profit, which is calculated as the sum of the profit per unit of each product multiplied by the production quantity of the product at each time period. We can write the objective function as:

$$f(x) = \sum_{i \in I} (r_i - c_i) \cdot x_i, \tag{2.7}$$

where r_i is the revenue per unit of product i, c_i is the cost per unit of product i, and x_i is the production quantity of product i.

We can write this objective function as shown in figure 2.15. Then, we can

```
def f(model):
    return sum((model.r[i] - model.c[i]) * model.x[i] for i in model.I)
```

Figure 2.15: Objective function formulation

add the objective function and set it as a maximization problem as shown in figure 2.16. Note that the process of defining a function does not automatically

```
#add the objective function to the model
model.obj = pyo.Objective(rule=f, sense=pyo.maximize)
```

Figure 2.16: Objective function addition to the model

add the function to the model. We need to explicitly add the function to the model, so both processes are necessary.

Constraints

A few constraints need to be satisfied to ensure that the production plan is feasible and meets the requirements of the problem. The constraints for this model are:

- **Budget constraint**: The total cost of production must not exceed the budget constraint b. This can be written as:

$$\sum_{i \in I} c_i \cdot x_i \leq b, \tag{2.8}$$

  and can be added into our Pyomo mdoel as shown in figure 2.17.

```python
def budget_constraint(model):
    return sum(model.c[i] * model.x[i] for i in model.I) <= model.b
model.budget_constraint = pyo.Constraint(rule=budget_constraint)
```

Figure 2.17: Budget constraint formulation

- **Storage capacity constraint**: The total storage capacity used by the production must not exceed the storage capacity constraint s, which can be written as:

$$\sum_{i \in I} x_i \leq s. \tag{2.9}$$

Figure 2.18 shows how to add this constraint to the model.

```python
def storage_capacity(model):
    return sum(model.x[i] for i in model.I) <= model.s
model.storage_capacity = pyo.Constraint(rule=storage_capacity)
```

Figure 2.18: Storage capacity constraint formulation

- **Maximum production constraint**: The production quantity must be at most the production capacity for each product, written as:

$$x_i \leq p_i \quad \forall i \in I. \tag{2.10}$$

Figure 2.19 shows how to add this constraint to the model.

```python
def production_capacity(model, i):
    return model.x[i] <= model.p[i]
model.production_capacity = pyo.Constraint(model.I, rule=production_capacity)
```

Figure 2.19: Production capacity constraint formulation

- **Non-negativity constraint**: The production quantity must be non-negative:

$$x_i \geq 0 \quad \forall i \in I, \tag{2.11}$$

which is can be added to the model as shown in figure 2.20. Note that the non-negativity constraint can also be added as a domain constraint when defining the decision variables. We will use the domain constraint in the subsequent examples.

```
def nonnegative_domain(model, i):
    return model.x[i] >= 0
model.nonnegative_domain = pyo.Constraint(model.I, rule=nonnegative_domain)
```

Figure 2.20: Non-negativity constraint formulation

Here is the reasoning behind each constraint. The budget constraint equation (2.8) ensures that the total cost of production does not exceed the budget constraint b. The storage capacity constraint equation (2.9) ensures that the total storage capacity used by the production does not exceed the storage capacity constraint Cs. The minimum production constraint equation (2.10) ensures that the production quantity does not exceed the production capacity for each product. The non-negativity constraint equation (2.11) ensures that the production quantity is non-negative.

Optimization Model

Putting all the components together, we can write the optimization model for the production planning problem as:

$$\begin{aligned} \underset{x}{\text{maximize}} \quad & \sum_{i \in I} (r_i - c_i) \cdot x_i \\ \text{subject to} \quad & \sum_{i \in I} c_i \cdot x_i \leq b, \\ & \sum_{i \in I} x_i \leq s, \\ & x_i \leq p_i \quad \forall i \in I, \\ & x_i \geq 0 \quad \forall i \in I. \end{aligned} \tag{2.12}$$

Exercise 2.4. Consider a production planning problem for three products with the following data:

Product	Profit (Rp/unit)	Cost (Rp/unit)	Production capacity (units)
1	50 million	30 million	100
2	70 million	40 million	150
3	90 million	50 million	200

The budget constraint is $b = 10$ billion Rupiah, and the storage capacity constraint is $s = 400$ units. Solve the production planning problem and determine the optimal production quantity for each product.

The data for this problem is available at: Simple Production Planning Model Dataset[3].

[3] The dataset is available at https://docs.google.com/spreadsheets/d/e/2PACX-1vQzIHhg3mZq4eGXraXQZl07kduWMhwrnUqqs_gPT6qH_V1SWI3crMZMllxG6MX1sz3QJCFBjMt9tftr/pub?output=xlsx

Solution: One way to solve this is to load the data and use optimization software such as Python Pyomo package to model the linear optimization model equation (2.12) and solve it using a solver such as GLPK. An easy way to load the data into the optimization model is shown in figure 2.21. The next step is to build the model (see figure 2.22). Then, we can solve it using GLPK (see figure 2.23). Finally, we can extract the optimal solution and store it in an Excel file for further analysis (see figure 2.24).

```
#load the data
EXCEL_URL = 'https://docs.google.com/spreadsheets/d/e/2PACX-1
     vQzIHhg3mZq4eGXraXQZl07kduWMhwrnUqqs_gPT6qH_V1SWI3crMZMllxG6MX1sz3QJCFBjMt9tftr
     /pub?output=xlsx'

df = pd.read_excel(EXCEL_URL, sheet_name='data')
params = pd.read_excel(EXCEL_URL, sheet_name='params').to_dict(orient='list')

#convert the data to dictionary for easy access
params = {name_: value_ for name_, value_ in zip(params["name"], params["val"])}
```

Figure 2.21: Data loading for the production planning problem.

At this point, you shall have a basic understanding of how to model and solve optimization problems using Pyomo and other optimization software. The process involves loading the data, defining the sets, parameters, decision variables, objective function, and constraints, solving the model using an optimization solver, and extracting and analyzing the results. This process can be applied to more complex optimization problems involving multiple decision variables, constraints, and objectives. The key is to break down the problem into its essential components and represent them in a way that can be solved efficiently and effectively.

Given the small size of the problem, you can verify the results manually to ensure that the model is working correctly. Check that the total cost of production does not exceed the budget constraint, the total storage capacity used by the

```
#initiate the model
model = pyo.ConcreteModel()

#define the set(s)
model.I = pyo.Set(initialize=[1, 2, 3])

#define the decision variables
model.x = pyo.Var(model.I, domain=pyo.Reals)

#define the parameters
model.r = pyo.Param(model.I, initialize=df['revenue'])
model.c = pyo.Param(model.I, initialize=df['cost'])
model.p = pyo.Param(model.I, initialize=df['production_capacity'])
model.b = pyo.Param(initialize=params['budget'])
model.s = pyo.Param(initialize=params['capacity'])

#load the functions into objective and constraints
model.obj = pyo.Objective(rule=objective, sense=pyo.maximize)
model.budget_constraint = pyo.Constraint(rule=budget_constraint)
model.storage_capacity = pyo.Constraint(rule=storage_capacity)
model.production_capacity = pyo.Constraint(model.I, rule=production_capacity)
model.nonnegative_domain = pyo.Constraint(model.I, rule=nonnegative_domain)
```

Figure 2.22: Building the production planning problem model.

```
#solve the model
solver = pyo.SolverFactory('glpk', executable='/usr/bin/glpsol')
result = solver.solve(model)
```

Figure 2.23: Solving the production planning problem using GLPK.

```
#display the results
print("Status:", result.solver.status)
print("Termination condition:", result.solver.termination_condition)
print("Optimal value:", pyo.value(model.obj))

print("Optimal production quantity:")
for i in model.I:
    print(f"Product {i}: {pyo.value(model.x[i])}")

#store the results in an Excel file
results = pd.DataFrame({
    "Product": list(model.I),
    "Production Quantity": [pyo.value(model.x[i]) for i in model.I]})
results.to_excel("production_planning_results.xlsx", index=False)
```

Figure 2.24: Extracting and displaying the solution for the production planning problem.

production does not exceed the storage capacity constraint, and the production quantity does not exceed the production capacity for each product. You can also analyze the results to understand the trade-offs between different products and how the production plan maximizes the total profit while satisfying the constraints. This will help you gain insights into the problem and understand the implications of the optimal solution, before moving on to larger and more complex model.

Exercise 2.5. Check your solution for Exercise 2.4 and verify that the model is working correctly. Specifically, check whether:

a. The total cost of production does not exceed the budget constraint.
b. The total storage capacity used by the production does not exceed the storage capacity constraint.
c. The production quantity does not exceed the production capacity for each product.
d. The production plan maximizes the total profit while satisfying the constraints.

In the following exercise, you will apply the same model to a slightly larger problem (using the Online Marketplace Import Dataset[4]). This will help you understand how to scale up the model and solve more complex optimization problems using the same approach.

[4] The dataset is available here: https://docs.google.com/spreadsheets/d/e/2PACX-1vSUL44wqRRtpUru5yzmEooxJdsnVuZhQyxksikX3Xe0RoaalSjwskPE52V8oKWF2Zn3V4xJ3KZ9GMTa/pub?output=xlsx

Exercise 2.6. Consider a larger problem with dataset available at: Online Marketplace Import Planning Dataset. The problem involves 1000 products from online shop marketplace. The goal is to determine the optimal import quantity for each product this year to maximize the total profit while satisfying budget, storage capacity, and import capacity constraints. The total budget is 5 billion Rupiah, the total storage capacity is 50,000 units. The import capacity for each product is given in the dataset.

a. **Solve the model and determine the optimal import quantity for each product!**
b. **What is the optimal total cost, total revenue, and total profit?**

c. **How many product types are imported to the maximum capacity? How many product types are imported but not to the maximum capacity?**
d. **What is the total import quantity for all products?**
e. **How much budget is left after importing the products?**
f. **How much import capacity remains?**
g. If you can either:

   - increase the budget by 10 million Rupiah, or
   - increase the import capacity by 10,000 units.

   Which one would you choose to maximize the total profit? Why?

2.5 *Summary*

In this chapter, we have introduced the basic concepts of optimization modeling and provided an overview of the optimization process. We have discussed the key components of an optimization model, including decision variables, objective function, constraints, and feasible region. We have also discussed the different types of optimization problems, such as linear optimization, nonlinear optimization, integer optimization, and black-box optimization. We have developed a systematic approach to modeling optimization problems and illustrated the process with a production planning example. We have also shown how to implement the optimization model in Python using the Pyomo library and solve it using optimization algorithms.

The key idea is to break down the problem into its essential components and represent them in a way that can be scaled up by loading the full-sized data, while working with the small-sized data during development and verification. This will help you understand the problem and develop an effective optimization model that can be solved efficiently and effectively. In the next chapter, we will discuss how to prepare the data for optimization modeling and what to consider when working with real-world data. This will help you understand how to gather, clean, and preprocess the data, especially if the data is known to be noisy. This will help you develop more robust and reliable optimization models that can be applied to real-world problems.

2.6 *End-of-Chapter Exercises*

Exercise 2.7. Consider a simple Rosenbrock function optimization problem with two decision variables x_1 and x_2:

$$\begin{aligned} \underset{x_1,x_2}{\text{minimize}} \quad & f(x_1,x_2) = (1-x_1)^2 + 100(x_2 - x_1^2)^2, \\ \text{subject to} \quad & -5 \leq x_1 \leq 5, \\ & -5 \leq x_2 \leq 5. \end{aligned} \tag{2.13}$$

Build an optimization model for this problem and solve it using IPOPT solver in Pyomo. **Determine the optimal solution for this optimization model!**

Exercise 2.8. Why is it important to start with a small and simple model when developing an optimization model? How can you scale up the model to handle larger and more complex problems?

Exercise 2.9. Your colleague is solving the import planning problem for an online marketplace with 1000 products (exercise 2.6), but found that the profit and cost data are not updated correctly (it still used the data from the previous year). The IT department has sent the updated data in a new Excel file. Given that the solution is needed urgently, your colleague claims that the solution is still optimal and the results are still valid, especially since not much has changed in the data by visual inspection. **What is your opinion on this matter? What would you do in this situation?**

Exercise 2.10. You are re-using the production planning model for a new project, but when loading the data, you found that the data contains missing values, inconsistent format (some thousand separators use commas, while others use periods). You also found that some products have negative production capacity values. **How would you handle these issues when loading the data into the optimization model?**

Exercise 2.11. The sales team wants you to automate the production planning process for a new product line that will be launched next year. However,

they want you to consider the uncertainty in the revenue and cost data. They have provided you with historical data for the revenue and cost of similar products, but they are not sure how accurate the data is for the new product line. **How would you handle the uncertainty in the data when developing the optimization model?**