

Operations Research

Applications of Linear Programming

Ling-Chieh Kung

Department of Information Management
National Taiwan University

Road map

- ▶ **Materials blending.**
- ▶ Linearizing maximum/minimum functions.
- ▶ AMPL.

Material blending

- ▶ In some situations, we need to determine not only products to produce but also **materials** to input.¹
- ▶ This is because we have some **flexibility** in making the products.
- ▶ For example, in making orange juice, we may use orange, sugar, water, etc. Different ways of **blending** these materials results in different qualities of juice.
- ▶ The goal is to save money (lower the proportion of expensive materials) while maintaining **quality**.

¹This example comes from Chapter 3 of *Operations Research: Applications and Algorithms* by Wayne L. Winston, 4th edition.

Material blending: the problem

- ▶ We blend materials 1, 2, and 3 to make products 1 and 2.
- ▶ The quality of a product, which depends on the proportions of these three materials, must meet the standard:
 - ▶ Product 1: at least 40% of material 1; at least 20% of material 2.
 - ▶ Product 2: at least 50% of material 1; at most 30% of material 3.
- ▶ At most 100 kg of product 1 and 150 kg of product 2 can be sold.
- ▶ Prices for products 1 and 2 are \$10 and \$15 per kg, respectively.
- ▶ Costs for materials 1 to 3 are \$8, \$4, and \$3 per kg, respectively.
- ▶ Amount of a product made equals the amount of materials input.
- ▶ We want to maximize the total profit.

Formulation: decision variables

- ▶ Probably our first attempt is to define the following: Let

x_1 = kg of product 1 produced,

x_2 = kg of product 2 produced,

y_1 = kg of material 1 purchased,

y_2 = kg of material 2 purchased, and

y_3 = kg of material 3 purchased.

- ▶ May we express the quality of each product? No!
- ▶ We need to specify the amount of material 1 used for product 1, the amount of material 1 used for product 2, etc.
- ▶ So we need to **redefine** our decision variables.

Formulation: decision variables

- ▶ How about this: Let

x_1 = kg of material 1 used for product 1,

x_2 = kg of material 1 used for product 2,

x_3 = kg of material 2 used for product 1,

x_4 = kg of material 2 used for product 2,

x_5 = kg of material 3 used for product 1, and

x_6 = kg of material 3 used for product 2.

- ▶ The definition is correct and precise, but **not easy to use**.
 - ▶ Similar to computer programming: give your variables reasonable names that allow people to know **what they are**.

Formulation: decision variables

- ▶ A more intuitive way of naming variables: Let

x_{11} = kg of material 1 used for product 1,

x_{12} = kg of material 1 used for product 2,

x_{21} = kg of material 2 used for product 1,

x_{22} = kg of material 2 used for product 2,

x_{31} = kg of material 3 used for product 1, and

x_{32} = kg of material 3 used for product 2.

- ▶ Or in a compact format:

x_{ij} = kg of material i used for product j , $i = 1, \dots, 3$, $j = 1, 2$.

Formulation: objective function

- ▶ Let's write down the total profit.
- ▶ Sales revenues depend on the amount of products we sell.
 - ▶ How many kg of product 1 may we sell? $x_{11} + x_{21} + x_{31}$ kg.
 - ▶ Similarly, we have $x_{12} + x_{22} + x_{32}$ kg of product 2.
- ▶ Material costs depend on the amount of materials we purchase.
 - ▶ Similarly, we need to buy $x_{11} + x_{12}$ kg of material 1, $x_{21} + x_{22}$ kg of material 2 and $x_{31} + x_{32}$ kg of material 3.
- ▶ The objective function is

$$\begin{aligned} & \max 10(x_{11} + x_{21} + x_{31}) + 15(x_{12} + x_{22} + x_{32}) \\ & \quad - 8(x_{11} + x_{12}) - 4(x_{21} + x_{22}) - 3(x_{31} + x_{32}) \\ & = \max 2x_{11} + 7x_{12} + 6x_{21} + 11x_{22} + 7x_{31} + 12x_{32}. \end{aligned}$$

Formulation: quality constraints

- ▶ To guarantee that at least 40% of product 1 are made by material 1?

$$\frac{x_{11}}{x_{11} + x_{21} + x_{31}} \geq 0.4.$$

- ▶ It is conceptually correct. However, it is **nonlinear**!
- ▶ Let's fix the nonlinearity by moving the denominator to the RHS:

$$x_{11} \geq 0.4(x_{11} + x_{21} + x_{31}).$$

Though equivalent, they are just different.

- ▶ We may (but are not required to) choose other format, such as

$$0.6x_{11} - 0.4x_{21} - 0.4x_{31} \geq 0 \quad \text{or} \quad 3x_{11} - 2x_{21} - 2x_{31} \geq 0.$$

Formulation: constraints

- ▶ In total we have four quality constraints:

- ▶ $x_{11} \geq 0.4(x_{11} + x_{21} + x_{31})$.

- ▶ $x_{21} \geq 0.2(x_{11} + x_{21} + x_{31})$.

- ▶ $x_{12} \geq 0.5(x_{12} + x_{22} + x_{32})$.

- ▶ $x_{13} \leq 0.3(x_{12} + x_{22} + x_{32})$.

- ▶ The demands are limited:

$$x_{11} + x_{21} + x_{31} \leq 100 \quad \text{and} \quad x_{12} + x_{22} + x_{32} \leq 150.$$

- ▶ The quantities are nonnegative:

$$x_{ij} \geq 0 \quad \forall i = 1, \dots, 3, j = 1, 2.$$

Formulation: the complete formulation

- ▶ The complete formulation is

$$\begin{aligned}
 \max \quad & 10(x_{11} + x_{21} + x_{31}) + 15(x_{12} + x_{22} + x_{32}) \\
 & - 8(x_{11} + x_{12}) - 4(x_{21} + x_{22}) - 3(x_{31} + x_{32}) \\
 \text{s.t.} \quad & x_{11} \geq 0.4(x_{11} + x_{21} + x_{31}), \quad x_{21} \geq 0.2(x_{11} + x_{21} + x_{31}), \\
 & x_{12} \geq 0.5(x_{12} + x_{22} + x_{32}) \quad x_{13} \leq 0.3(x_{12} + x_{22} + x_{32}) \\
 & x_{11} + x_{21} + x_{31} \leq 100, \quad x_{12} + x_{22} + x_{32} \leq 150 \\
 & x_{ij} \geq 0 \quad \forall i = 1, \dots, 3, j = 1, 2.
 \end{aligned}$$

- ▶ Some remarks:
 - ▶ We may need to redefine decision variables when it is necessary.
 - ▶ We may from time to time use multi-dimensional variables.
 - ▶ We need to **linearize** nonlinear constraints or objective functions, even if they look so similar.

Road map

- ▶ Materials blending.
- ▶ **Linearizing maximum/minimum functions.**
- ▶ AMPL.

Fair allocation: the problem

- ▶ Suppose that we want to allocate \$1000 to two persons in a **fair** way.
- ▶ We adopt the following measurement of fairness: The smaller the difference between the two amounts, the fairer the allocation is.
- ▶ Obviously the answer is to give each person \$500.
- ▶ May we formulate a linear program to solve this problem?

Fair allocation: the first attempt

- ▶ Let x_i be the amount allocated to person i , $i = 1, 2$.
- ▶ Is the following formulation correct?

$$\begin{array}{ll} \min & x_2 - x_1 \\ \text{s.t.} & x_1 + x_2 = 1000 \\ & x_i \geq 0 \quad \forall i = 1, 2. \end{array}$$

Fair allocation: the second attempt

- ▶ Let x_i be the amount allocated to person i , $i = 1, 2$.
- ▶ The following formulation is correct:

$$\begin{array}{ll} \min & |x_2 - x_1| \\ \text{s.t.} & x_1 + x_2 = 1000 \\ & x_i \geq 0 \quad \forall i = 1, 2. \end{array}$$

- ▶ However, the absolute function $|\cdot|$ is **nonlinear**!
- ▶ It is possible to linearize this problem as a linear program?

Linearizing the second attempt

- ▶ First, let w be the absolute difference: $w = |x_2 - x_1|$:

$$\begin{array}{ll} \min & w \\ \text{s.t.} & x_1 + x_2 = 1000 \\ & w = |x_2 - x_1| \\ & x_i \geq 0 \quad \forall i = 1, 2. \end{array}$$

- ▶ We may change this equality constraint to an inequality:

$$\begin{array}{ll} \min & w \\ \text{s.t.} & x_1 + x_2 = 1000 \\ & w \geq |x_2 - x_1| \\ & x_i \geq 0 \quad \forall i = 1, 2. \end{array}$$

Why?

Linearizing the second attempt

- ▶ Now, notice that $|x_2 - x_1| = \max\{x_2 - x_1, x_1 - x_2\}$ and

$$w \geq \max\{x_2 - x_1, x_1 - x_2\} \quad \Leftrightarrow \quad w \geq x_2 - x_1 \quad \text{and} \quad w \geq x_1 - x_2.$$

- ▶ Therefore, the linear program we want is

$$\begin{aligned} \min \quad & w \\ \text{s.t.} \quad & x_1 + x_2 = 1000 \\ & w \geq x_2 - x_1 \\ & w \geq x_1 - x_2 \\ & x_i \geq 0 \quad \forall i = 1, 2. \end{aligned}$$

- ▶ May we solve this LP and get the (500, 500) allocation?

Solving the linear program

- ▶ Consider the LP

$$\begin{aligned} \min \quad & w \\ \text{s.t.} \quad & x_1 + x_2 = 1000 \\ & w \geq x_2 - x_1 \\ & w \geq x_1 - x_2 \\ & x_i \geq 0 \quad \forall i = 1, 2. \end{aligned}$$

- ▶ The equality constraint means that $x_2 = 1000 - x_1$:

$$\begin{aligned} \min \quad & w \\ \text{s.t.} \quad & w \geq 1000 - 2x_1 \\ & w \geq 2x_1 - 1000 \\ & x_1 \geq 0. \end{aligned}$$

- ▶ Would you graphically solve the LP?

Linearizing constraints

- ▶ The technique we just applied can be generalized.
- ▶ When a **maximum** function is at the **smaller** side of an inequality:

$$y \geq \max\{x_1, x_2\} \quad \Leftrightarrow \quad y \geq x_1 \text{ and } y \geq x_2.$$

- ▶ y , x_1 , and x_2 can be variables, parameters, or a function of them:

$$\begin{aligned} y + x_1 + 3 &\geq \max\{x_1 - x_3, 2x_2 + 4\} \\ \Leftrightarrow y + x_1 + 3 &\geq x_1 - x_3 \text{ and } y + x_1 + 3 \geq 2x_2 + 4. \end{aligned}$$

- ▶ There may be more than two terms in the maximum function:

$$y \geq \max_{i=1, \dots, n} \{x_i\} \quad \Leftrightarrow \quad y \geq x_i \quad \forall i = 1, \dots, n.$$

Linearizing constraints

- ▶ A **minimum** function at the **larger** side can also be linearized.

$$y + x_1 \leq \min\{x_1 - x_3, 2x_2 + 4, 0\}$$
$$\Leftrightarrow y + x_1 \leq x_1 - x_3, y + x_1 \leq 2x_2 + 4, \text{ and } y + x_1 \leq 0.$$

- ▶ This technique **does not** apply to:
 - ▶ A maximum function at the larger side: $y \leq \max\{x_1, x_2\}$ is not equivalent to $y \leq x_1$ and $y \leq x_2$.
 - ▶ A minimum function at the smaller side: $y \geq \min\{x_1, x_2\}$ is not equivalent to $y \geq x_1$ and $y \geq x_2$.
 - ▶ A maximum or minimum function in an equality.

Linearizing constraints

- ▶ A **minimum** function at the **larger** side can also be linearized.

$$y + x_1 \leq \min\{x_1 - x_3, 2x_2 + 4, 0\}$$
$$\Leftrightarrow y + x_1 \leq x_1 - x_3, y + x_1 \leq 2x_2 + 4, \text{ and } y + x_1 \leq 0.$$

- ▶ This technique **does not** apply to:
 - ▶ A maximum function at the larger side: $y \leq \max\{x_1, x_2\}$ is not equivalent to $y \leq x_1$ and $y \leq x_2$.
 - ▶ A minimum function at the smaller side: $y \geq \min\{x_1, x_2\}$ is not equivalent to $y \geq x_1$ and $y \geq x_2$.
 - ▶ A maximum or minimum function in an equality.

Linearizing the objective function

- ▶ When we **minimize a maximum function**:

$$\min \max\{x_1, x_2\} \quad \Leftrightarrow \quad \begin{array}{ll} \min & w \\ \text{s.t.} & w \geq x_1 \\ & w \geq x_2. \end{array}$$

- ▶ x_1 and x_2 can be variables, parameters, or a function of them.
 - ▶ There may be other constraints.
 - ▶ The objective function may contain other terms.
- ▶ Similarly, when we **maximize a minimum function**:

$$\begin{array}{ll} \max & \min\{x_1, x_2, 2x_3 + 5\} + x_4 \\ \text{s.t.} & 2x_1 + x_2 - x_4 \leq x_3. \end{array} \quad \Leftrightarrow \quad \begin{array}{ll} \max & w + x_4 \\ \text{s.t.} & w \leq x_1 \\ & w \leq x_2 \\ & w \leq 2x_3 + 5 \\ & 2x_1 + x_2 - x_4 \leq x_3. \end{array}$$

Linearizing the objective function

- ▶ This technique does not apply to:
 - ▶ Maximizing a maximum function.
 - ▶ Minimizing a minimum function.
- ▶ Finally, an **absolute function** is just a maximum function:

$$|x| = \max\{x, -x\}.$$

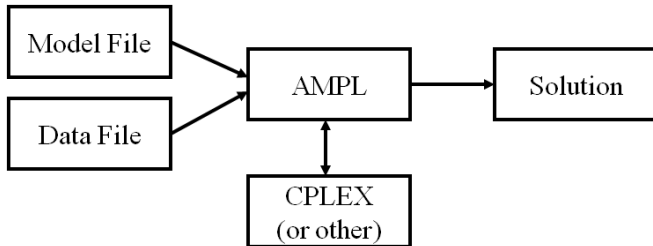
- ▶ Minimizing an absolute function can be linearized.
- ▶ An absolute function at the smaller side of an inequality can be linearized.

Road map

- ▶ Materials blending.
- ▶ Linearizing maximum/minimum functions.
- ▶ **AMPL.**

AMPL

- ▶ **AMPL** = “A Modeling Language for Mathematical Programming.”
- ▶ AMPL is an interface, and **Cplex** is a solver.



To obtain AMPL

- ▶ Office website: <http://ampl.com/>.
- ▶ To download a size-limited student version:
<http://ampl.com/try-ampl/download-a-free-demo/>.
- ▶ A typical package includes the components:
 - ▶ **ampl**: The console environment.
 - ▶ **cplex**: A solver for linear (fractional or integer) programs.
 - ▶ **minos**: A solver for nonlinear (fractional) programs.
 - ▶ **sw**: A more user-friendly console environment (MS Windows only) for “scrolling windows.”
- ▶ In this course, we use the licensed full education version.

The first example

- ▶ Consider our favorite LP

$$\begin{aligned} z^* = \max \quad & x_1 + x_2 \\ \text{s.t.} \quad & x_1 + 2x_2 \leq 6 \\ & 2x_1 + x_2 \leq 6 \\ & x_i \geq 0 \quad \forall i = 1, 2. \end{aligned}$$

- ▶ An optimal solution is $x^* = (2, 2)$. The associated $z^* = 4$.

The first example

- ▶ To use AMPL to solve this LP, all we need is a **model file**:

```
var x1;  
var x2;
```

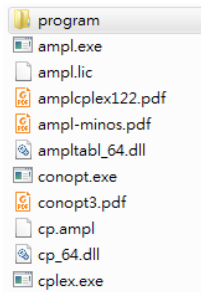
```
maximize profit: x1 + x2;
```

```
subject to resource_1: x1 + 2 * x2 <= 6;  
subject to resource_2: 2 * x1 + x2 <= 6;  
subject to nonneg_1: x1 >= 0;  
subject to nonneg_2: x2 >= 0;
```

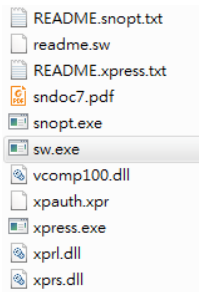
- ▶ Let's put these codes into a plain text file called "eg1.mod" and save this file in a "program" (or other name you prefer) folder.
- ▶ Let's try it first and explain the codes later.

The first example

Put your “eg1.mod” in the “program” folder.



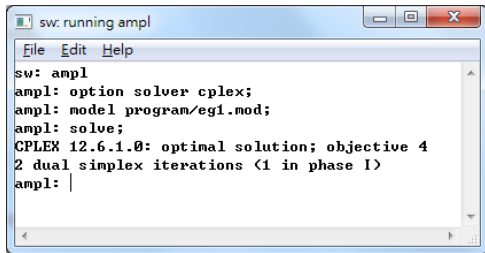
Open the console environment `sw` (or `ampl` in Mac).



The first example

- ▶ Type the following instructions one by one:

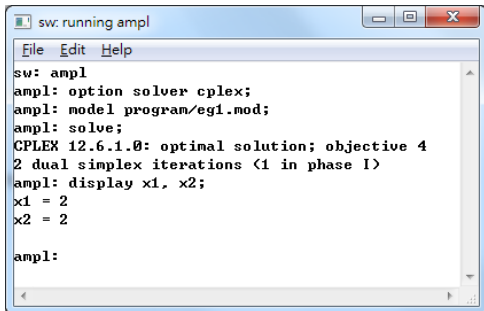
```
ampl
option solver cplex;
model program/eg1.mod;
solve;
```
- ▶ An optimal solution is found!
 - ▶ With the solver CPLEX.
 - ▶ The objective value of the optimal solution is 4.



```
sw: running ampl
File Edit Help
sw: ampl
ampl: option solver cplex;
ampl: model program/eg1.mod;
ampl: solve;
CPLEX 12.6.1.0: optimal solution; objective 4
2 dual simplex iterations (1 in phase I)
ampl: |
```

The first example

- ▶ To see the optimal solution, type
 display x1, x2;
- ▶ The values are displayed.
 - ▶ $x^* = (2, 2)$.



```
sw: running ampl
File Edit Help
sw: ampl
ampl: option solver cplex;
ampl: model program/eg1.mod;
ampl: solve;
CPLEX 12.6.1.0: optimal solution; objective 4
2 dual simplex iterations (1 in phase I)
ampl: display x1, x2;
x1 = 2
x2 = 2

ampl:
```

The first example: codes revisited

- ▶ Let's explain the codes in the model file.

```
var x1; # use "var" to declare variables
var x2; # each AMPL statement ends with a semicolon

maximize profit: x1 + x2; # name your objective function

subject to resource_1: x1 + 2 * x2 <= 6; # name each constraint
subject to resource_2: 2 * x1 + x2 <= 6;
subject to nonneg_1: x1 >= 0;
subject to nonneg_2: x2 >= 0;
```

- ▶ **Reserved words:** var, maximize, minimize, and subject to.
- ▶ Give all constraints and the objective function **distinct names**.
- ▶ Do not forget **colons and semicolons**.
- ▶ Use # to write **comments**.

The first example: make modifications

- ▶ Let's modify the code (and save the modified file):

```
subject to resource_1: x1 + 3 * x2 <= 6;
```

- ▶ Go back to the console and type

```
reset;  
model program/eg1.mod;  
solve;  
display x1, x2;
```

See how the optimal solution changes. Do not forget to **reset!**

- ▶ Remarks:
 - ▶ The file can be names with any extension file name as long as it is a plain-text file.
 - ▶ Be aware of the file path.
 - ▶ In Mac, use **absolute file paths** (for loading a file or a solver).

The second example

- ▶ Three products, four markets, different production costs and retail prices Find the production and sales plan to maximize profit.

Product	Market				Capacity
	1	2	3	4	
1	\$20 / \$30	\$40 / \$45	\$15 / \$30	\$30 / \$40	500
2	\$30 / \$35	\$25 / \$30	\$15 / \$35	\$20 / \$30	600
3	\$25 / \$40	\$35 / \$40	\$10 / \$20	\$25 / \$30	400

The mathematical model

- ▶ Variables: Let

x_{ij} = sales quantity of product i at market j , $i = 1, \dots, 3$, $j = 1, \dots, 4$.

- ▶ Parameters: We denote the unit cost and price of product i at market j as C_{ij} and P_{ij} , respectively, and the capacity for product i as K_i .
- ▶ The mathematical model (an LP):

$$\begin{aligned} \max \quad & \sum_{i=1}^3 \sum_{j=1}^4 (P_{ij} - C_{ij})x_{ij} \\ \text{s.t.} \quad & \sum_{j=1}^4 x_{ij} \leq K_i \quad \forall i = 1, \dots, 3 \\ & x_{ij} \geq 0 \quad \forall i = 1, \dots, 3, j = 1, \dots, 4. \end{aligned}$$

Decoupling the data from a model

- ▶ To make our AMPL programs flexible and extendable, we should **decouple** the data from a model.
- ▶ To do this, we will prepare a **model file** and a **data file**.
 - ▶ The model file contains a conceptual model.
 - ▶ The data file contains the instance parameters.
- ▶ They should both be stored as plain-text files. The extension name does not matter.
- ▶ Be aware of file paths.
 - ▶ Name them as “eg2.mod” and “eg2.dat” and store them in the “program” folder.

The model file

```
param P;                # number of product
param M;                # number of market

param Capacity{i in 1..P};    # the capacity vector
param Cost{i in 1..P, j in 1..M}; # the cost matrix
param Price{i in 1..P, j in 1..M}; # the price matrix

var x{i in 1..P, j in 1..M};

maximize profit:        # use "sum" for summation
    sum{i in 1..P, j in 1..M} (Price[i, j] - Cost[i, j]) * x[i, j];

subject to productCapacity{i in 1..P}: # constraint indices
    sum{j in 1..M} x[i, j] <= Capacity[i];
subject to nonnegX{i in 1..P, j in 1..M}:
    x[i, j] >= 0;
```

The data file

```
param P := 3;
param M := 4;

param Capacity :=
  1 500
  2 600
  3 400;

param Cost:    1  2  3  4 :=
  1 20 40 15 30
  2 30 25 15 20
  3 25 35 10 25;

param Price:   1  2  3  4 :=
  1 30 45 30 40
  2 35 30 35 30
  3 40 40 20 30;
```

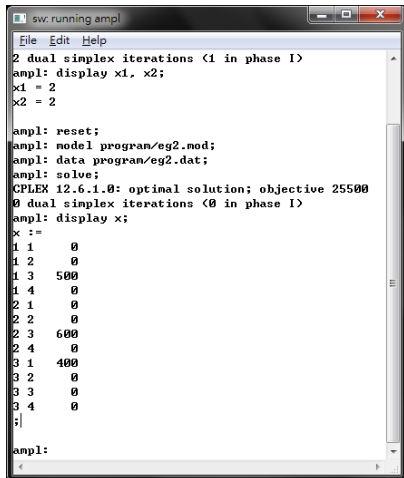
- ▶ The format does not matter.
- ▶ Reserved words: **param**.
- ▶ Parameter names must be consistent with those defined in the model file.
- ▶ Array and matrix lengths must be consistent with their limits.
- ▶ Be aware of those :=, ;, and : and the timing of using them.

Solving the second example

- ▶ Solve the second example by loading the model and data files.

```
reset;  
model program/eg2.mod;  
data program/eg2.dat;  
solve;  
display x;
```

- ▶ Do not forget to reset!



```
sw: running ampl  
File Edit Help  
2 dual simplex iterations (1 in phase I)  
ampl: display x1, x2;  
x1 = 2  
x2 = 2  
  
ampl: reset;  
ampl: model program/eg2.mod;  
ampl: data program/eg2.dat;  
ampl: solve;  
CPLEX 12.6.1.0: optimal solution; objective 25500  
0 dual simplex iterations (0 in phase I)  
ampl: display x;  
x :=  
1 1 0  
1 2 0  
1 3 500  
1 4 0  
2 1 0  
2 2 0  
2 3 600  
2 4 0  
3 1 400  
3 2 0  
3 3 0  
3 4 0  
:  
:  
ampl:
```

Some remarks

- ▶ The **default solver** in AMPL is MINOS.
 - ▶ You may choose to use MINOS by typing `option solver minos`.
 - ▶ MINOS can also solve LP.
 - ▶ CPLEX uses simplex-based methods while MINOS uses interior search methods (not covered in this course).
 - ▶ For solving LPs, CPLEX performs better.
 - ▶ MINOS cannot solve integer programs; CPLEX can.
- ▶ AMPL is **case-sensitive**.
- ▶ Try the AMPL instructions `show`; and `expand`; at home.
- ▶ Use `exit`; to exit the AMPL environment.
- ▶ The official AMPL book is freely available on the official website.