AMPL Tutorial

Prof. Larry Snyder
Lehigh University

1/22/20

Outline

Introduction

The Centre County Problem: Explicit Form

The Centre County Problem: General Algebraic Form

Things That Can Go Wrong

Wyndor Glass Model

Other Things to Know

Your Turn

Outline

Introduction

The Centre County Problem: Explicit Form

The Centre County Problem: General Algebraic Form

Things That Can Go Wrong

Wyndor Glass Model

Other Things to Know

Your Turn

AMPL's Role

We write
optimization
models like
this:

$$\text{maximize} \quad \sum_{j=1}^{n} p_j x_j$$

$$\text{subject to} \quad \sum_{j=1}^{n} a_{ij} x_j \leq b_i \quad \forall i$$

$$x_j \leq d_j \delta_j \quad \forall j$$

$$x_j \geq 0 \quad \forall j$$

$$\delta_j \in \{0, 1\} \quad \forall j$$

AMPL's Role

We write optimization models like this:

maximize $\sum_{j=1}^{n} p_j x_j$

subject to $\sum_{j=1}^{n} a_{ij} x_j \leq b_i \quad \forall i$

$x_j \leq d_j \delta_j \quad \forall j$

$x_j \geq 0 \quad \forall j$

$\delta_j \in \{0, 1\} \quad \forall j$

The solver (algorithm) needs them to look like this:

$c = [2.5, 2.0, 1.7, 0, 0, 0]$

$A = \begin{bmatrix} 1 & 2 & 1 & 0 & 0 & 0 \\ 0 & 1 & 3 & 0 & 0 & 0 \\ 1 & 0 & 0 & -30 & 0 & 0 \\ 0 & 1 & 0 & 0 & -10 & 0 \\ 0 & 0 & 1 & 0 & 0 & -15 \end{bmatrix}$
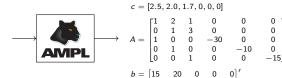
$b = \begin{bmatrix} 15 & 20 & 0 & 0 & 0 \end{bmatrix}'$

AMPL's Role

We write optimization models like this:

AMPL does the translating:

The solver (algorithm) needs them to look like this:

$$\text{maximize} \quad \sum_{j=1}^{n} p_j x_j$$

$$\text{subject to} \quad \sum_{j=1}^{n} a_{ij} x_j \le b_i \quad \forall i$$

$$x_j \le d_j \delta_j \quad \forall j$$

$$x_j \ge 0 \quad \forall j$$

$$\delta_j \in \{0, 1\} \quad \forall j$$

$\longrightarrow$

$\longrightarrow$

$c = [2.5, 2.0, 1.7, 0, 0, 0]$

$$A = \begin{bmatrix} 1 & 2 & 1 & 0 & 0 & 0 \\ 0 & 1 & 3 & 0 & 0 & 0 \\ 1 & 0 & 0 & -30 & 0 & 0 \\ 0 & 1 & 0 & 0 & -10 & 0 \\ 0 & 0 & 1 & 0 & 0 & -15 \end{bmatrix}$$

$b = \begin{bmatrix} 15 & 20 & 0 & 0 & 0 \end{bmatrix}'$

AMPL Blends Algebraic Notation and Computer Code

```ampl
param num_resources;                    # number of resources (M)
param num_products;                     # number of products (N)

set RESOURCES := 1..num_resources;      # set of resources
set PRODUCTS := 1..num_products;        # set of products

param fixed_cost {PRODUCTS};            # fixed cost (K_j)
param var_cost {PRODUCTS};              # variable cost (C_j)
param sales_price {PRODUCTS};           # sales price (p_j)
param sales_potential {PRODUCTS};       # sales potential (d_j)
param avail {RESOURCES};                # resource availability (b_i)
param needs {RESOURCES, PRODUCTS};      # num units of i needed for j (a_ij)

var ProduceAmt {PRODUCTS} >= 0;         # num units of product to produce (x_j)
var Produce {PRODUCTS} binary;          # produce product? (delta_j)

maximize Profit:
    sum {j in PRODUCTS} sales_price[j] * ProduceAmt[j]
    - sum {j in PRODUCTS} (fixed_cost[j] * Produce[j]
        + var_cost[j] * ProduceAmt[j]);

subject to Supply {i in RESOURCES}:
    sum {j in PRODUCTS} needs[i,j] * Produce[j] <= avail[i];

subject to LinkingAndSalesPotential {j in PRODUCTS}:
    Produce[j] <= sales_potential[j] * Produce[j];
```

Some Basics

- ▶ AMPL is case-sensitive
- ▶ AMPL ignores whitespace
- ▶ Your model will go into a text file (.mod and/or .dat)
- ▶ You will type commands like solve at the ampl: prompt
- ▶ Comments are denoted by a number sign (#)
- ▶ Every line must end with a semicolon (;)

Outline

Centre County Problem

▶ Centre County, PA is considering 4 potential community development projects:

Project	Daily Usage	Cost	Land Space (acres)
Park	600	$50,000	8
Basketball court	100	$20,000	0
Recreation center	300	$150,000	4
Swimming pool	500	$70,000	5

▶ Basketball court will be built in park
  ▶ No space needed
  ▶ But cannot build unless build park

Centre County Problem

- ▶ $200,000 from state
- ▶ 15 acres available
- ▶ Goal: select projects to max daily usage s.t. budget and land constraints

Source: Ravindran, Griffin, and Prabhu, *Service Systems Engineering and Management*, CRC Press, 2018.

Centre County Problem

$$\max Z = 600x_1 + 100x_2 + 300x_3 + 500x_4$$
$$\text{s.t.} \quad 50x_1 + 20x_2 + 150x_3 + 70x_4 \leq 200$$
$$8x_1 \quad\quad + 4x_3 + 5x_4 \leq 15$$
$$x_1 , \quad x_2 , \quad x_3 , \quad x_4 \leq 1$$
$$x_1 , \quad x_2 , \quad x_3 , \quad x_4 \geq 0$$

▶ (For now, we'll allow the decision variables to be continuous and ignore the "if basketball, then park" constraint)
▶ Open AMPL and create a file called centre.mod
▶ Type the following:

```
var x1 >= 0, <= 1;
var x2 >= 0, <= 1;
var x3 >= 0, <= 1;
var x4 >= 0, <= 1;

maximize Profit: 600 * x1 + 100 * x2 + 300 * x3 + 500 * x4;

subject to Budget: 50 * x1 + 20 * x2 + 150 * x3 + 70 * x4 <= 200;

subject to Space: 8 * x1 + 4 * x3 + 5 * x4 <= 15;
```

Centre County Problem

```
var x1 >= 0, <= 1;
var x2 >= 0, <= 1;
var x3 >= 0, <= 1;
var x4 >= 0, <= 1;

maximize Profit: 600 * x1 + 100 * x2 + 300 * x3 + 500 * x4;

subject to Budget: 50 * x1 + 20 * x2 + 150 * x3 + 70 * x4 <= 200;

subject to Space: 8 * x1 + 4 * x3 + 5 * x4 <= 15;
```

▶ Notice that:
  ▶ Every line ends with a semicolon
  ▶ Decision variables are declared using var
  ▶ The objective function is declared using maximize (or minimize)
  ▶ Constraints are declared using subject to (or shorten to subj to)
  ▶ The objective function and constraints each get a name (Profit, Budget, Space); names must be unique

Centre County Problem

```
var x1 >= 0, <= 1;
var x2 >= 0, <= 1;
var x3 >= 0, <= 1;
var x4 >= 0, <= 1;

maximize Profit: 600 * x1 + 100 * x2 + 300 * x3 + 500 * x4;

subject to Budget: 50 * x1 + 20 * x2 + 150 * x3 + 70 * x4 <= 200;

subject to Space: 8 * x1 + 4 * x3 + 5 * x4 <= 15;
```

▶ At the ampl: prompt, type:

```
ampl: model centre.mod;        <-- tells AMPL which model you want to solve
ampl: option solver cplex;     <-- tells AMPL which solver you want to use
ampl: solve;                   <-- tells AMPL to solve it!
```

▶ You should see:

```
CPLEX 12.8.0.0: optimal solution; objective 1320
1 dual simplex iterations (0 in phase I)
```

Congrats, You Just Solved Your First AMPL Model!

```
CPLEX 12.8.0.0: optimal solution; objective 1320
1 dual simplex iterations (0 in phase I)
```

▶ objective 1320 means the optimal objective function value is 1320

▶ Let's find out the values of the decision variables

▶ We ask AMPL for values using the display command:

```
ampl: display x1, x2, x3, x4;
x1 = 1
x2 = 1
x3 = 0.4
x4 = 1
```

Binary Variables

▶ Now let's make the variables binary instead of continuous

▶ Replace $>= 0$, $<= 1$ with binary:

```
var x1 binary;
var x2 binary;
var x3 binary;
var x4 binary;
```

▶ Let's also add the "if basketball, then park" constraint:

```
subject to IfBasketballThenPark: x2 <= x1;
```

Binary Variables

- ▶ When we change the .mod file, we must tell AMPL to reset
- ▶ Then re-load the model
- ▶ Then re-solve
- ▶ (We don't need to specify the solver again)

```
ampl: reset;
ampl: model centre.mod;
ampl: solve;
CPLEX 12.8.0.0: optimal integer solution; objective 1200
0 MIP simplex iterations
0 branch-and-bound nodes
```

Binary Variables

```
ampl: reset;
ampl: model centre.mod;
ampl: solve;
CPLEX 12.8.0.0: optimal integer solution; objective 1200
0 MIP simplex iterations
0 branch-and-bound nodes
```

▶ Again, let's find out the values of the variables:

```
ampl: display x1, x2, x3, x4;
x1 = 1
x2 = 1
x3 = 0
x4 = 1
```

▶ Yay, they're binary!
▶ (By the way, you can use the ↑ and ↓ keys to scroll through earlier commands)

Outline

A More General Algebraic Approach

▶ So far, so good

▶ But: This format would be a big pain if lots of variables

▶ The solution is to use **sets** to index our **parameters**, **decision variables**, **summations**, and **constraints**

▶ Let's omit the "if basketball, then park" constraint for now

▶ Create a new file called centrecounty.mod that contains:

```
set PROJECTS;                # set of projects (P)

param usage {PROJECTS};      # usage for project j (u_j)
param cost {PROJECTS};       # cost for project j (c_j)
param space {PROJECTS};      # space for project j (s_j)

var Select {PROJECTS} binary;  # select project j? (x_j)

maximize TotalUsage: sum {j in PROJECTS} usage[j] * Select[j];

subj to Budget: sum {j in PROJECTS} cost[j] * Select[j] <= 200;

subj to LandAvailable: sum {j in PROJECTS} space[j] * Select[j] <= 15;
```

A More General Algebraic Approach

```
set PROJECTS;                    # set of projects (P)

param usage {PROJECTS};          # usage for project j (u_j)
param cost {PROJECTS};           # cost for project j (c_j)
param space {PROJECTS};          # space for project j (s_j)

var Select {PROJECTS} binary;    # select project j? (x_j)

maximize TotalUsage: sum {j in PROJECTS} usage[j] * Select[j];

subj to Budget: sum {j in PROJECTS} cost[j] * Select[j] <= 200;

subj to LandAvailable: sum {j in PROJECTS} space[j] * Select[j] <= 15;
```

- ▶ Notice that:
    - ▶ Parameters and variables are indexed by the set; this is
      indicated with curly braces: param usage {PROJECTS}
    - ▶ In objective and constraints, use square brackets to index the
      parameters and variables: usage[p]
    - ▶ Text after the comment symbol (#) is ignored
    - ▶ Summation indices are also indicated with curly braces:
      sum {j in PROJECTS}

But Wait—There Are No Numbers!

```
set PROJECTS;                    # set of projects (P)

param usage {PROJECTS};          # usage for project j (u_j)
param cost {PROJECTS};           # cost for project j (c_j)
param space {PROJECTS};          # space for project j (s_j)

var Select {PROJECTS} binary;    # select project j? (x_j)

maximize TotalUsage: sum {j in PROJECTS} usage[j] * Select[j];

subj to Budget: sum {j in PROJECTS} cost[j] * Select[j] <= 200;

subj to LandAvailable: sum {j in PROJECTS} space[j] * Select[j] <= 15;
```

- ▶ We need some way to provide the **data** (**sets** and **parameter** values)
- ▶ The data go into a .dat file
- ▶ Separation of model and data is core to AMPL's philosophy

The Data File

▶ The .dat file specifies the items in each **set** and the values of each **parameter**

▶ Create a new file called centrecounty.dat:

```
set PROJECTS := park basketball rec pool;

param usage :=
    park          600
    basketball    100
    rec           300
    pool          500 ;

param cost :=
    park          50
    basketball    20
    rec           150
    pool          70 ;

param space :=
    park          8
    basketball    0
    rec           4
    pool          5 ;
```

The Data File

```
set PROJECTS := park basketball rec pool;

param usage :=
    park          600
    basketball    100
    rec           300
    pool          500 ;

param cost :=
    park          50
    basketball    20
    rec           150
    pool          70 ;

param space :=
    park          8
    basketball    0
    rec           4
    pool          5 ;
```

▶ Notice that:
  ▶ Lines still end with semicolons
  ▶ Set elements can be strings (they can also be numbers)
  ▶ Declarations use the := symbol

Solving the Revised Model

▶ Now let's solve the revised model

```
ampl: reset;
ampl: model centrecounty.mod;
ampl: data centrecounty.dat;            <-- tells AMPL which data you want to use
ampl: solve;
CPLEX 12.8.0.0: optimal integer solution; objective 1200
0 MIP simplex iterations
0 branch-and-bound nodes
```

▶ We can display decision variables, just like before, even if
   they are indexed by sets:

```
ampl: display Select;
Select [*] :=
basketball  1
      park  1
      pool  1
       rec  0
;
```

▶ It's the same solution as before—not surprising, since it's the
   same model (and data)

Basketball Constraint

- ▶ Let's add the "if basketball then park" constraint back into the model
- ▶ Note that **set** elements that are strings must be enclosed in single quotes

```
subj to IfBasketballThenPark: Select['basketball'] <= Select['park'];
```

- ▶ (We already know that the solution will be the same, but let's solve it anyway)

```
ampl: reset;
ampl: model centrecounty.mod;
ampl: data centrecounty.dat;
ampl: solve;
CPLEX 12.8.0.0: optimal integer solution; objective 1200
0 MIP simplex iterations
0 branch-and-bound nodes
```

A More Compact Data Syntax

▶ When several parameters have the same index set, we can combine them into one table:

```
set PROJECTS := park basketball rec pool;

param:        usage   cost    space :=
park          600     50      8
basketball    100     20      0
rec           300     150     4
pool          500     70      5 ;
```

▶ White space (tabbing, etc.) doesn't matter, but nice alignment makes the table easier to read

A More Compact Data Syntax

▶ When several parameters have the same index set, we can combine them into one table:

```
set PROJECTS := park basketball rec pool;

param:        usage   cost    space :=
park          600     50      8
basketball    100     20      0
rec           300     150     4
pool          500     70      5 ;
```

▶ White space (tabbing, etc.) doesn't matter, but nice alignment makes the table easier to read

▶ Try the new data format and re-solve the model

▶ Did you get the same optimal solution?

Right-Hand Sides as Parameters

▶ The right-hand sides of our constraints are written as numbers

▶ It's good practice to avoid "hard-coding" *any* numbers in the .mod file

▶ Instead, declare them as parameters:

```
param budget;              # available budget
param land_avail;          # available land
```

▶ And replace the right-hand sides with those parameters:

```
subj to Budget: sum {j in PROJECTS} cost[j] * Select[j] <= budget;

subj to LandAvailable: sum {j in PROJECTS} space[j] * Select[j] <= land_avail;
```

▶ And declare the new parameters in the .dat file:

```
param budget := 200;
param land_avail := 15;
```

Right-Hand Sides as Parameters

▶ Let's solve the revised model:

```
ampl: reset;
ampl: model centrecounty.mod;
ampl: data centrecounty.dat;
ampl: solve;
CPLEX 12.8.0.0: optimal integer solution; objective 1200
0 MIP simplex iterations
0 branch-and-bound nodes
ampl: display Select;
Select [*] :=
basketball  1
      park  1
      pool  1
       rec  0
;
```

▶ Yay.

"For All" Constraints

▶ Suppose we want to add a constraint that says that the total spent on *each* project can be no more than 60:

$$c_j x_j \leq 60 \qquad \forall j \in P$$

▶ To implement the "for all," we *index the constraints*:

```
subj to MaxSpendPerProject {j in PROJECTS}: cost[j] * Select[j] <= 60;
```

"For All" Constraints

▶ Suppose we want to add a constraint that says that the total spent on *each* project can be no more than 60:

$$c_j x_j \le 60 \qquad \forall j \in P$$

▶ To implement the "for all," we *index the constraints*:

```
subj to MaxSpendPerProject {j in PROJECTS}: cost[j] * Select[j] <= 60;
```

▶ Or better yet:

```
param max_spend;              # max spend per project
...
subj to MaxSpendPerProject {j in PROJECTS}: cost[j] * Select[j] <= max_spend;
```

▶ And in centrecounty.dat:

```
param max_spend := 60;
```

"For All" Constraints

▶ Let's re-solve the model:

```
ampl: reset;
ampl: model centrecounty.mod;
ampl: data centrecounty.dat;
ampl: solve;
CPLEX 12.8.0.0: optimal integer solution; objective 700
0 MIP simplex iterations
0 branch-and-bound nodes
ampl: display Select;
Select [*] :=
basketball  1
      park  1
      pool  0
       rec  0
;
```

Outline

Introduction

The Centre County Problem: Explicit Form

The Centre County Problem: General Algebraic Form

Things That Can Go Wrong

Wyndor Glass Model

Other Things to Know

Your Turn

Dangling Subscripts

  ▶ AMPL is *great* at recognizing dangling subscripts!

```
set PROJECTS;                 # set of projects (P)

param usage {PROJECTS};       # usage for project j (u_j)
param cost {PROJECTS};        # cost for project j (c_j)
param space {PROJECTS};       # space for project j (s_j)

var Select {PROJECTS} binary; # select project j? (x_j)

maximize TotalUsage: usage[j] * Select[j];       <-- no sum!!!

...
```

  ▶ Then:

```
ampl: model centrecounty.mod;

centrecounty.mod, line 9 (offset 281):
    j is not defined
context:  maximize TotalUsage:  >>> usage[j] <<< * Select[j];
```

Dangling Subscripts

► Or:

```
subj to MaxSpendPerProject: cost[j] * Select[j] <= max_spend;          <-- no for-all!!!
```

```
ampl: model centrecounty.mod;

centrecounty.mod, line 23 (offset 722):
    j is not defined
context:   subj to MaxSpendPerProject:  >>> cost[j] <<<  * Select[j] <= max_spend;
```

Same Index in Sum and For-All

- ▶ AMPL also complains if you use the same index in a sum and a for-all:

```
subj to MaxSpendPerProject {j in PROJECTS}:
    sum {j in PROJECTS} cost[j] * Select[j] <= max_spend; <-- j in sum and for-all!!!
```

```
ampl: model centrecounty.mod;

centrecounty.mod, line 24 (offset 747):
    syntax error
context:  sum {j  >>> in  <<< PROJECTS} cost[j] * Select[j] <= max_spend;
```

- ▶ (The syntax error isn't that helpful, but the >>> in <<< is)

Misspellings

```
maximize TotalUsage: sum {j in PROJECTS} usage[j] * Slect[j];    <-- oops!
```

```
ampl: model centrecounty.mod;

centrecounty.mod, line 15 (offset 459):
    Slect is not defined
context:   maximize TotalUsage: sum {j in PROJECTS} usage[j] *  >>> Slect[j]; <<<
```

Misspellings

```
maximize TotalUsage: sum {j in PROJECTS} usage[j] * Slect[j];    <-- oops!
```

```
ampl: model centrecounty.mod;

centrecounty.mod, line 15 (offset 459):
    Slect is not defined
context:   maximize TotalUsage: sum {j in PROJECTS} usage[j] * >>> Slect[j]; <<<
```

▶ Don't forget that AMPL is case-sensitive:

```
maximize TotalUsage: sum {j in PROJECTS} usage[j] * select[j];   <-- should be Select!
```

```
ampl: model centrecounty.mod;

centrecounty.mod, line 15 (offset 459):
    select is not defined
context:   maximize TotalUsage: sum {j in PROJECTS} usage[j] * >>> select[j]; <<<
```

Missing Data

```
param:       usage    cost    space :=
park          600      50      8
basketball    100      20      0
rec           300     150      4 ;                 <-- no pool!
```

```
ampl: model centrecounty.mod;
ampl: data centrecounty.dat;
ampl: solve;
Error executing "solve" command:
error processing objective TotalUsage:
    no value for usage['pool']
```

▶ Notice that the error doesn't occur until you solve

Missing Semicolon

```
maximize TotalUsage: sum {j in PROJECTS} usage[j] * Select[j]    <-- no semicolon!
```

```
ampl: model centrecounty.mod;

centrecounty.mod, line 17 (offset 471):
    syntax error
context:  >>> subj  <<< to Budget: sum {j in PROJECTS} cost[j] * Select[j] <= budget;
```

► Notice that the error actually points to the *next* line, because that's when AMPL realizes something is wrong

Missing Semicolon

▶ If you forget the semicolon on the command line, AMPL just
   prompts you for it, like a parent waiting for a "please":

```
ampl: model centrecounty.mod;
ampl: data centrecounty.dat;
ampl: solve
ampl?
```

▶ Then just type the ; and you are forgiven:

```
ampl? ;
CPLEX 12.8.0.0: optimal integer solution; objective 700
0 MIP simplex iterations
0 branch-and-bound nodes
```

Outline

Remember Wyndor Glass?

$$
\begin{array}{llrcrcl}
\text{maximize} & 3x_1 & + & 5x_2 & & \\
\text{subject to} & x_1 & & & \leq & 4 \\
& & & 2x_2 & \leq & 12 \\
& 3x_1 & + & 2x_2 & \leq & 18 \\
& x_1 & , & x_2 & \geq & 0
\end{array}
$$

- ▶ 2 products, 3 plants
- ▶ Objective maximizes total profit
- ▶ Constraints enforce plant capacities

Wyndor Glass in Algebraic Notation

▶ Sets:
  ▶ I = set of plants
  ▶ J = set of products
▶ Parameters:
  ▶ r_j = profit per batch of product j sold
  ▶ b_i = available hours at plant i
  ▶ a_{ij} = production time per batch of product j made at plant i
▶ Decision variables:
  ▶ x_j = number of batches of product j produced

$$\text{maximize} \quad \sum_{j \in J} r_j x_j$$
$$\text{subject to} \quad \sum_{j \in J} a_{ij} x_j \le b_i \quad \forall i \in I$$
$$x_j \ge 0 \qquad \forall j \in J$$

wyndor.mod

$$\text{maximize} \quad \sum_{j \in J} r_j x_j$$

$$\text{subject to} \quad \sum_{j \in J} a_{ij} x_j \leq b_i \quad \forall i \in I$$

$$x_j \geq 0 \qquad \forall j \in J$$

```
set PLANTS;                    # set of plants (I)
set PRODUCTS;                  # set of products (J)

param profit {PRODUCTS};       # profit per unit of product j (r_j)
param avail_hours {PLANTS};    # available hours at plant i (b_i)
param hours_per_batch {PLANTS,PRODUCTS};
                               # num hours at plant i per batch of product j (a_ij)

var Produce {PRODUCTS} >= 0;   # num batches of product j to produce (x_j)

maximize TotalProfit: sum {j in PRODUCTS} profit[j] * Produce[j];

subj to PlantCapacity {i in PLANTS}:
    sum {j in PRODUCTS} hours_per_batch[i,j] * Produce[j] <= avail_hours[i];
```

wyndor.dat

| | Production Time | | |
Plant	Product 1	Product 2	Available Hours
1	1	0	4
2	0	2	12
3	3	2	18
Profit per Batch (x1000)	$3	$5	

```
set PLANTS := 1 2 3;
set PRODUCTS := 1 2;

param profit :=
    1   3
    2   5 ;

param avail_hours :=
    1   4
    2   12
    3   18 ;

param hours_per_batch :       <-- syntax for 2-index parameters in .dat file
        1   2   :=
    1   1   0
    2   0   2
    3   3   2 ;
```

Solving the Model

```
ampl: model wyndor.mod;
ampl: data wyndor.dat;
ampl: solve;
CPLEX 12.8.0.0: optimal solution; objective 36
1 dual simplex iterations (0 in phase I)
ampl: display Produce;
Produce [*] :=
1   2
2   6
;
```

▶ Our old friend $x^* = (2, 6)$, $Z^* = 36$

Outline

Solvers

- ▶ AMPL can interface with *lots* of different **solvers**
- ▶ The solver is the code that actually does the optimization
- ▶ The AMPL input to the solver (`.mod` and `.dat` files) and the AMPL output from the solver (e.g., `display Select;`) are the same, regardless of solver
- ▶ For most purposes, CPLEX or Gurobi are sufficient
  - ▶ They can solve LPs, IPs, and other types of problems
  - ▶ But you need a license

Sets of Numbers

- ▶ AMPL makes it easy to declare sets of numbers:
  - ▶ set PERIODS := 1..20 $\implies$ PERIODS = {1, 2, ..., 20}
  - ▶ set PERIODS := 5..50 by 5 $\implies$
    PERIODS = {5, 10, ..., 50}
- ▶ A common pattern is to declare the set *size* in the .mod file,
  declare the *set* based on the size also in the .mod file, and
  specify the size in the .dat file:

.mod:

```
param T;
set PERIODS := 1..T;
```

.dat:

```
param T := 20;
```

- ▶ **Note:** ".." notation can only be used in .mod files, not .dat

Sets of Numbers

▶ You can also use sets of numbers without explicitly declaring them as sets

.mod:

```
param cost {1..T};
var Produce {1..T};

minimize TotalCost: sum {1..T} cost[t] * Produce[t];
```

.dat:

```
param T := 20;
```

▶ (But I find it cleaner to declare the sets)

Parameter Conditions

- ▶ You can impose conditions on parameters in the .mod file
- ▶ These are not constraints! They are just validation rules

.mod:

```
param budget >= 0;
param land_avail > 0;
param is_open binary;
param num_staff integer;
param staffing_level >= 7;
```

.dat:

```
param budget := 200;
param land_avail := -15;
```

```
ampl: model centrecounty.mod;
ampl: data centrecounty.dat;
ampl: solve;
Error executing "solve" command:
error processing param land_avail:
    failed check: param land_avail = -15
        is not > 0;
```

Set Operations

- ▶ AMPL provides operators to work with sets:
  - ▶ union means $A \cup B$
  - ▶ inter means $A \cap B$
  - ▶ diff means $A \setminus B$
  - ▶ symdiff means symmetric difference: in A or B but not both
  - ▶ cross means cross product (Cartesian product): pairs (a, b) with $a \in A$, $b \in B$

```
set SUPPLIERS;
set FACTORIES;
set NODES = SUPPLIERS union FACTORIES;
set EXTERNAL_SUPPLIERS = SUPPLIERS diff FACTORIES;
set LINKS = SUPPLIERS cross FACTORIES;
```

.run Files

- ▶ If you find yourself repeating the same commands over and over again...
- ▶ ...use a .run file!
- ▶ A .run file is a script consisting of standard AMPL commands

wyndor.run:

```
reset;
model wyndor.mod;
data wyndor.dat;
option solver cplex;
solve;
display Produce;
```

- ▶ Then, run your file from the command line:

```
ampl: include wyndor.run;
```

.run Files

► You can also use for, let, printf, etc.

```
reset;
model wyndor.mod;
data wyndor.dat;
option solver cplex;

set HOURS_TO_TEST := avail_hours[3] .. avail_hours[3] + 10 by 2;
for {h in HOURS_TO_TEST} {
    let avail_hours[3] := h;
    solve;
    printf "avail_hours[3] = %2d ==> optimal profit = %6.2f\n", h, TotalProfit;
}
```

```
ampl: include wyndor.run;
CPLEX 12.8.0.0: optimal solution; objective 36
1 dual simplex iterations (0 in phase I)
avail_hours[3] = 18 ==> optimal profit =  36.00
<-- suppressing further CPLEX output... -->
avail_hours[3] = 20 ==> optimal profit =  38.00
avail_hours[3] = 22 ==> optimal profit =  40.00
avail_hours[3] = 24 ==> optimal profit =  42.00
avail_hours[3] = 26 ==> optimal profit =  42.00
avail_hours[3] = 28 ==> optimal profit =  42.00
```

.run Files

▶ if-then-else conditions, too:

```
...
for {h in HOURS_TO_TEST} {
    let avail_hours[3] := h;
    solve;
    if Produce[1] >= 3 then
        printf "avail_hours[3] = %2d ==> producing >= 3 batches of product 1\n", h;
    else
        printf "avail_hours[3] = %2d ==> producing <3 batches of product 1\n", h;
}
```

```
CPLEX 12.8.0.0: optimal solution; objective 36
1 dual simplex iterations (0 in phase I)
avail_hours[3] = 18 ==> producing <3 batches of product 1
<-- suppressing further CPLEX output... -->
avail_hours[3] = 20 ==> producing <3 batches of product 1
avail_hours[3] = 22 ==> producing >= 3 batches of product 1
avail_hours[3] = 24 ==> producing >= 3 batches of product 1
avail_hours[3] = 26 ==> producing >= 3 batches of product 1
avail_hours[3] = 28 ==> producing >= 3 batches of product 1
```

Outline

Your Turn: Fixed-Charge Problem

- ▶ N products we can manufacture
- ▶ Product j:
    - ▶ Incurs a fixed cost K_j if we manufacture it
    - ▶ Incurs a variable cost C_j per unit manufactured
    - ▶ Earns a profit p_j per unit sold
    - ▶ Has a demand potential d_j
- ▶ M raw materials
- ▶ Each unit of product j manufactured requires a_{ij} units of raw material i
- ▶ b_i units of raw material i are available
- ▶ Goal: select product mix to maximize net profit

Your Turn: Fixed-Charge Problem

$$\text{maximize} \quad \sum_{j=1}^{N} p_j x_j - \sum_{j=1}^{N} (K_j \delta_j + C_j x_j)$$

$$\text{subject to} \quad \sum_{j=1}^{N} a_{ij} x_j \leq b_i \qquad \forall i = 1, \ldots, M$$

$$x_j \leq d_j \delta_j \qquad \forall j = 1, \ldots, N$$

$$x_j \geq 0 \qquad \forall j = 1, \ldots, N$$

$$\delta_j \in \{0, 1\} \qquad \forall j = 1, \ldots, N$$

Source: Ravindran, Griffin, and Prabhu, *Service Systems Engineering and Management*, CRC Press, 2018.

Your Turn: Fixed-Charge Problem

▶ Use the following data:

Resource	Resources Needed (a_{ij})				# Available (b_i)
	Prod. 1	Prod. 2	Prod. 3	Prod. 4	
1	4	1	0	2	400
2	0	2	3	2	600
3	1	1	1	1	300
Fixed cost (K_j)	100	150	50	100	
Variable cost (C_j)	10	10	40	30	
Sales price (p_j)	22	30	18	45	
Sales potential (d_j)	100	75	140	60	

This Slide Intentionally Left Blank...

My .mod File

```
param num_resources;                    # number of resources (M)
param num_products;                     # number of products (N)

set RESOURCES := 1..num_resources;      # set of resources
set PRODUCTS := 1..num_products;        # set of products

param fixed_cost {PRODUCTS};            # fixed cost (K_j)
param var_cost {PRODUCTS};              # variable cost (C_j)
param sales_price {PRODUCTS};           # sales price (p_j)
param sales_potential {PRODUCTS};       # sales potential (d_j)
param avail {RESOURCES};                # resource availability (b_i)
param needs {RESOURCES , PRODUCTS};     # units of resource needed for product (a_ij)

var ProduceAmt {PRODUCTS} >= 0;         # num units of product to produce (x_j)
var Produce {PRODUCTS} binary;          # 0/1 if we produce/don't produce (delta_j)

maximize Profit:
    sum {j in PRODUCTS} sales_price[j] * ProduceAmt[j]
    - sum {j in PRODUCTS} (fixed_cost[j] * Produce[j] + var_cost[j] * ProduceAmt[j]);

subject to Supply {i in RESOURCES}:
    sum {j in PRODUCTS} needs[i,j] * ProduceAmt[j] <= avail[i];

subject to LinkingAndSalesPotential {j in PRODUCTS}:
    ProduceAmt[j] <= sales_potential[j] * Produce[j];
```

My .dat File

```
param num_resources := 3;
param num_products := 4;

param:   fixed_cost  var_cost     sales_price sales_potential :=
    1    100         10           22          100
    2    150         10           30          75
    3    50          40           18          140
    4    100         30           45          60 ;

param avail :=
    1    400
    2    600
    3    300 ;

param needs :
        1   2   3   4    :=
    1   4   1   0   2
    2   0   2   3   2
    3   1   1   1   1 ;
```

My AMPL Transcript

```
ampl: model fixedcharge.mod;
ampl: data fixedcharge.dat;
ampl: solve;
CPLEX 12.8.0.0: optimal integer solution; objective 2665
0 MIP simplex iterations
0 branch-and-bound nodes
absmipgap = 4.54747e-13, relmipgap = 1.70637e-16
ampl: display Produce;
Produce [*] :=
1  1
2  1
3  0
4  1
;

ampl: display ProduceAmt;
ProduceAmt [*] :=
1  51.25
2  75
3   0
4  60
;
```

▶ So: We produce products 1, 2, and 4, with
$x^* = (51.25, 75, 0, 60)$ and an optimal profit of 2665