Teaching, Learning & Applying Optimization

AMPL's Intuitive Modeling Meets the Python Ecosystem

Filipe Brandão, Robert Fourer

{fdabrandao,4er}@ampl.com

AMPL Optimization Inc.
www.ampl.com — +1 773-336-AMPL

INFORMS Sponsored Webinar

13 December 2023

Teaching, Learning, and Applying Optimization: AMPL's Intuitive Modeling Meets the Python Ecosystem

Optimization is part of any educational program in Operations Research or Analytics, but the curriculum must steadily evolve to remain relevant. Following an introductory example, this presentation takes you on a tour through new developments in the AMPL modeling language and system that have been changing the ways that large-scale optimization is taught and learned:

- ***A more natural approach to describing optimization problems.*** Students can write common logical conditions, "not-quite-linear" functions, and nonlinear functions the way they think about them, without having to learn complicated and error-prone reformulations.
- ***A Python-first alternative to learning AMPL and model-building.*** New teaching materials leverage the power of Jupyter notebooks and Google Colab to incorporate modern computing concepts and the vast Python ecosystem into the study of optimization.
- ***Faster, easier importing of data and exporting of results.*** The AMPL Python interface (amplpy) efficiently connects model sets and parameters to Python's native data structures and Pandas dataframes. An all-new spreadsheet interface reads and writes .xlsx and .csv files, with added support for two-dimensional spreadsheet tables.
- ***Streamlined application development.*** Python scripts are quickly turned into illustrative applications using amplpy, Pandas, and the Streamlit app framework.

These features are freely available for teaching, in convenient bundles of AMPL and popular solvers. The AMPL for Courses program provides full-featured, unlimited use by students and staff for the duration of your academic term. Courses can also take advantage of our Community Edition, size-limited demos, and short-term full-featured trials.

AMPL + Python for Teaching & Learning

Part 1: AMPL's intuitive modeling

- Background & motivation
  - Principles of algebraic model-based optimization
  - Writing optimization models like you think about them
- *Writing optimization models **more** like you think about them*
  - MP: an extended interface to solvers
  - Introduction: Multi-product flow with logical conditions
  - *Examples from users' questions and complaints*
  - Survey: Logical and "not linear" expressions now supported
- Using AMPL
  - Modeling environments: commands, scripts, APIs, ***amplpy***
  - Data interfaces: Spreadsheets, databases, ***Python***

AMPL + Python for Teaching & Learning

Part 1: AMPL's intuitive modeling

Part 2: AMPL meets the Python ecosystem

- A Python-first approach
  - Interfacing with Python using *amplpy*
  - AMPL in Jupyter notebooks
  - AMPL model colaboratory on Google Colab
- Enhancements to the AMPL Python interface
  - Installing AMPL and solvers as Python packages
  - Importing and exporting data naturally
    from/to Python data structures such as Pandas dataframes
  - Turning Python scripts into prescriptive analytics applications
    in minutes with Pandas, amplpy, and Streamlit

info@ampl.com

ampl.com — ampl.com/start-free-now/ — colab.ampl.com

Optimization in OR & Analytics

Given a recurring need to make many interrelated decisions

- Purchases, production and shipment amounts, assignments, . . .

Consistently make highly desirable choices

By applying ideas from mathematical optimization

- Ways of describing problems *(models)*
- Ways of solving problems *(algorithms)*

Model-Based Optimization

Steps

- *model:* Formulate a general description of a class of optimization problems
- *data:* Get values that define a scenario to be solved; combine them with the model to generate a problem instance
- *solver:* Apply algorithmic software to compute highly desirable decisions for the problem instance
- *results:* Analyze or deploy the solution

Independence

- *model* is independent of *data*
- *model* & *data* are independent of *solver*

Algebraic Model-Based Optimization

Mathematical model formulation

- *sets & parameters:* Description of the data required
- *decision variables:* Solution values to be determined
- *objective:* Function of the variables
  that one would like to minimize or maximize
- *constraints:* Conditions that the variables must satisfy
  to meet the requirements of the problem

Model-based optimization software

- *user's side:* Work with models and data
  to develop & implement optimization applications
- *solver's side:* Work with efficient data structures
  to apply mathematical optimization algorithms

AMPL: Write Optimization Models Like You Think About Them

Idea

- Design a computer modeling language
  that's a lot like mathematical model notation
- Build a system for working with models, data and solvers

Example

- Multi-Product Network Flow

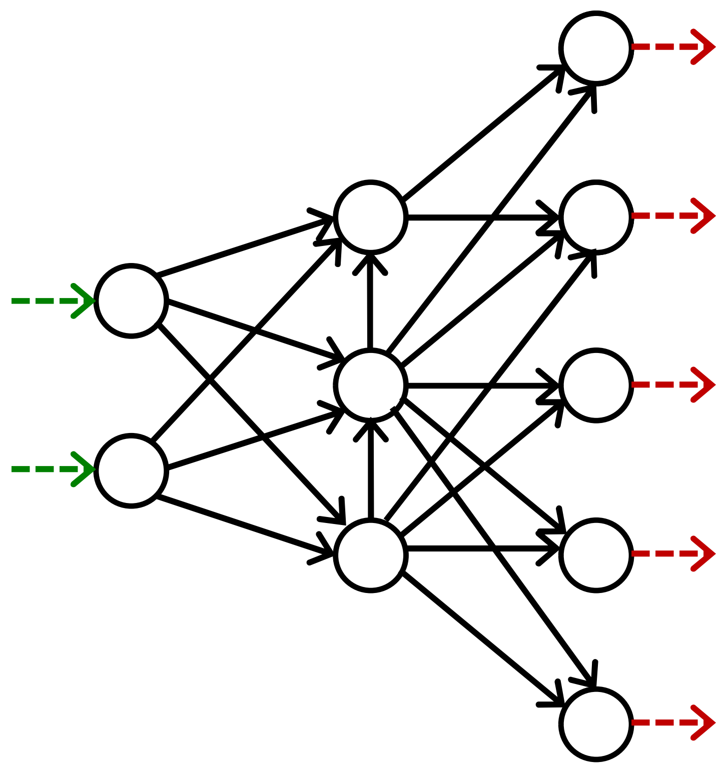
Example:
Multi-Product Network Flow

Motivation

- Ship products efficiently
  to meet demands

Context

- a transportation network
  - ∗ nodes ◯ representing cities
  - ∗ arcs ⟶ representing roads
- supplies - - -> at nodes
- demands - - -> at nodes
- capacities on arcs
- shipping costs on arcs



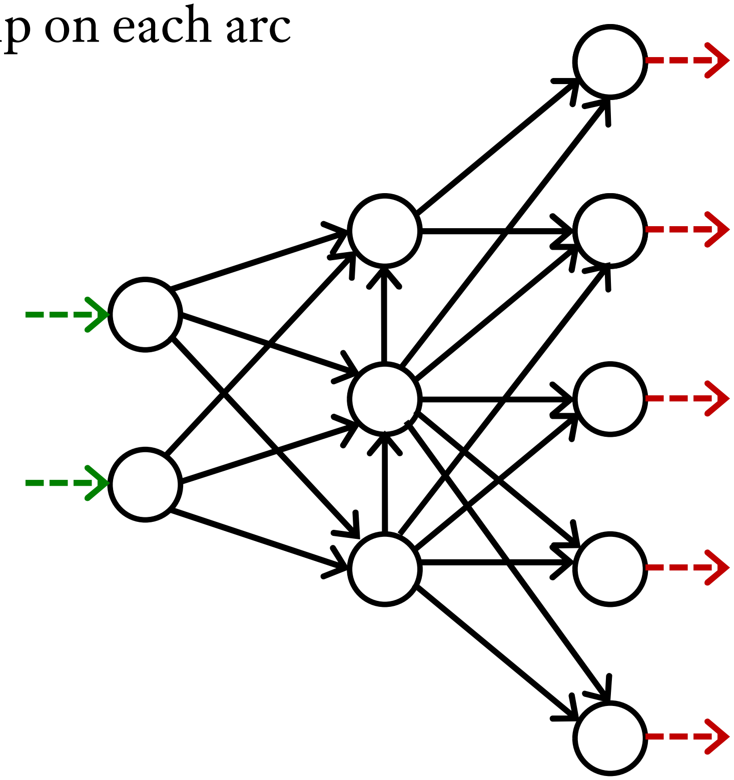
Example:
Multi-Product Network Flow

Decide

- how much of each product to ship on each arc

So that

- shipping costs are kept low
- shipments on each arc respect capacity of the arc
- supplies, demands, and shipments are in balance at each node



Multi-Product Flow

Formulation *(data)*

Given

P set of products

N set of network nodes

$A \subseteq N \times N$ set of arcs connecting nodes

and

u_{ij} capacity of arc from i to j, for each $(i, j) \in A$

s_{pj} net supply of product p at node j, for each $p \in P$, $j \in N$
> 0 implies supply, < 0 implies demand

c_{pij} cost per unit to ship product p on arc (i, j),
for each $p \in P$, $(i, j) \in A$

Multi-Product Flow

Formulation *(variables, objective, constraints)*

Determine

X_{pij} amount of commodity p to be shipped on arc (i, j),
for each $p \in P$, $(i, j) \in A$

to minimize

$\sum_{p \in P} \sum_{(i,j) \in A} c_{pij} X_{pij}$

total cost of shipments

Subject to

$\sum_{p \in P} X_{pij} \leq u_{ij}$, for all $(i, j) \in A$

total shipments must not exceed capacity

$\sum_{(i,j) \in A} X_{pij} + s_{pj} = \sum_{(j,i) \in A} X_{pji}$, for all $p \in P$, $j \in N$

shipments in plus net supply must equal shipments out

Multi-Product Flow

Model in AMPL

Symbolic data, variables, objective

```
set PRODUCTS;
set NODES;
param net_supply {PRODUCTS,NODES};

set ARCS within {NODES,NODES};
param capacity {ARCS} >= 0;

param var_cost {PRODUCTS,ARCS} >= 0;
var Flow {PRODUCTS,ARCS} >= 0;

minimize TotalCost:
   sum {p in PRODUCTS, (i,j) in ARCS} var_cost[p,i,j] * Flow[p,i,j];

subject to Capacity {(i,j) in ARCS}:
   sum {p in PRODUCTS} Flow[p,i,j] <= capacity[i,j];

subject to Conservation {p in PRODUCTS, j in NODES}:
   sum {(i,j) in ARCS} Flow[p,i,j] + net_supply[p,j] =
   sum {(j,i) in ARCS} Flow[p,j,i];
```

$$\sum_{(i,j)\in A} X_{pij} + s_{pj} = \sum_{(j,i)\in A} X_{pji}, \quad \text{for all } p \in P,\ j \in N$$

Writing Optimization Models
More Like You Think About Them

Motivation

- AMPL has logical and "not linear" expressions, but previous solver interfaces had very limited support for these
- Simple example: Multi-product flow with *logical conditions*
- Real examples and user complaints

Realization in AMPL

- An all-new AMPL-solver interface library supports more natural & direct ways of expressing models
- Allows many more expression types to be used freely in models
- Automatically converts to the forms required by each solver
  - using native solver extensions where available
  - automatically linearizing where necessary

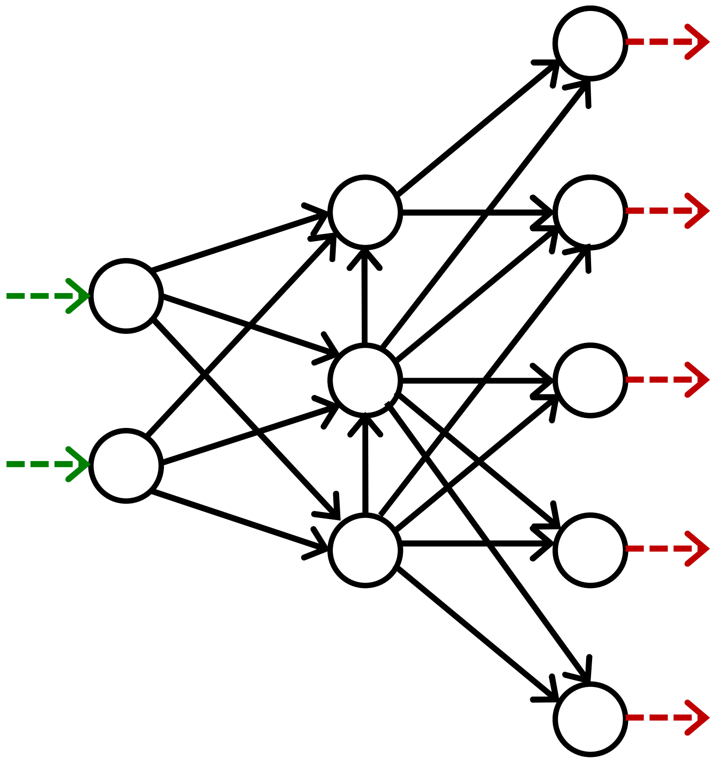
Example with conditions: Multi-Product Network Flow

Decide also

- whether to use each arc

So that

- variable costs plus fixed costs for shipping are kept low
- shipments are not too small
- not too many arcs are used



Formulating

Positive Shipments Incur Fixed Costs

How you think about it

```
param fix_cost {ARCS} >= 0;

minimize TotalCost:
   sum {p in PRODUCTS, (i,j) in ARCS} var_cost[p,i,j] * Flow[p,i,j] +
   sum {(i,j) in ARCS}
      if exists {p in PRODUCTS} Flow[p,i,j] > 0 then fix_cost[i,j];
```

How it would be linearized

```
param fix_cost {ARCS} >= 0;
var Use {ARCS} binary;

minimize TotalCost:
   sum {p in PRODUCTS, (i,j) in ARCS} var_cost[p,i,j] * Flow[p,i,j] +
   sum {(i,j) in ARCS} fix_cost[i,j] * Use[i,j];
```

Formulating

Shipments Can’t Be Too Small

How you think about it

```
subject to Shipment_Limits {(i,j) in ARCS}:
  sum {p in PRODUCTS} Flow[p,i,j] = 0 or
  min_ship <= sum {p in PRODUCTS} Flow[p,i,j] <= capacity[i,j];
```

How it would be linearized

```
subject to Min_Shipment {(i,j) in ARCS}:
   sum {p in PRODUCTS} Flow[p,i,j] >= min_ship * Use[i,j];

subject to Capacity {(i,j) in ARCS}:
   sum {p in PRODUCTS} Flow[p,i,j] <= capacity[i,j] * Use[i,j];
```

Formulating

Can't Use Too Many Arcs

How you think about it

```
subject to Limit_Used:
  atmost max_arcs {(i,j) in ARCS}
    (sum {p in PRODUCTS} Flow[p,i,j] > 0);
```

How it would be linearized

```
subject to Max_Used:
   sum {(i,j) in ARCS} Use[i,j] <= max_arcs;
```

Formulating

Linearization is Seldom That Simple!

```
subject to IfConstr {i in 1..card(veh)-1, j in i+1..card(veh):
         veh_ind[i] = veh_ind[j] and theory_time[i] <= theory_time[j]}:
   in_lane_veh[i] = in_lane_veh[j]
      ==> in_in_time[j] >= in_in_time[i] + l_veh/V;
```

```
minimize total_fuelcost:
   sum{(i,j) in A} sum{k in V} X[i,j,k] *
      ((if H[i,k] <= 300 then dMor[i,j] else
        if H[i,k] <= 660 then dAft[i,j] else
        if H[i,k] <= 901 then dEve[i,j]) * 5 +
       (if H[i,k] <= 300 then tMor[i,j] else
        if H[i,k] <= 660 then tAft[i,j] else
        if H[i,k] <= 901 then tEve[i,j]) * 0.0504);
```

```
subject to NoPersonIsolated
         {l in TYPES['loc'], r in TYPES['rank'], j in 1..numberGrps}:
   sum {i in LOCRANK[l,r]} Assign[i,j] = 0 or
   sum {i in LOCRANK[l,r]} Assign[i,j] +
     sum {a in ADJACENT[r]} sum {i in LOCRANK[l,a]} Assign[i,j] >= 2;
```

Motivation

Typical MIP User Complaint

Thank you so much for replying.
Let me show my "if-then" constraint in a more clear way as follows:

```
set veh := {1..16 by 1};

param veh_ind {veh};
param theory_time {veh};
param UP := 400000;

var in_lane_veh {veh} integer >=1, <=2;
var in_in_time {veh} >=0, <=UP;
```

Note that "in_lane_veh {veh}" are integer variables which equal 1 or 2,
and "in_in_time {veh}" are continuous variables.

```
subject to IfConstr {i in 1..card(veh)-1, j in i+1..card(veh):
  veh_ind[i] = veh_ind[j] and theory_time[i] <= theory_time[j]}:

    in_lane_veh[i] = in_lane_veh[j] ==> in_in_time[j] >= in_in_time[i] + l_veh/V;
```

When I run my program, there appears the following statement:

```
CPLEX 20.1.0.0: logical constraint _slogcon[1] is not an indicator constraint.
```

Motivation

Typical Response

To reformulate this model in a way that your MIP solver would accept,
you could define some more binary variables,

```
var in_lane_same {veh,veh} binary;
```

with the idea that in_lane_same[i,j] should be 1 if and only if in_lane_veh[i] = in_lane_veh[j].
Then the desired relation could be written as two constraints:

```
in_lane_veh[i] = in_lane_veh[j] ==> in_lane_same[i,j] = 1
in_lane_same[i,j] = 1 ==> in_in_time[j] >= in_in_time[i] + l_veh/V;
```

The second one is an indicator constraint, but you would just need
to replace the first one by equivalent linear constraints.

Given that in_lane_veh can only be either 1 or 2, those constraints could be

```
in_lane_same[i,j] >= 3 - in_lane_veh[i] - in_lane_veh[j]
in_lane_same[i,j] >= in_lane_veh[i] + in_lane_veh[j] - 3
```

Motivation

Typical Nonlinear User Complaint

So I tried out gurobi with the two commands I mentioned in my previous email, and I receive the message

```
Gurobi 9.0.2: Gurobi can't handle nonquadratic nonlinear constraints.
```

I went over the constraints, and it seems to me the only constraint that is nonquadratic nonlinear is

```
subject to A2 {t in 2..card(POS), i in PATIENTS}:
   sum {a in DONORS, b in PATIENTS, c in PATIENTS: ceil(a/2) = c}
      x[b,t] * x[c,t-1] * y[a,b] = 2 * x[i,t];
```

where x and y are binary variables.

Is this now sufficient for gurobi to solve if I only linearize one of the term on the LHS of this constraint (e.g. x[b, t]), while keeping the other two terms the same?

Motivation

Typical Response

*You are right, A2 has a cubic term x[b,t] * x[c,t-1] * y[a,b] that you will have to transform before you can get Gurobi to accept it.*

You can transform to quadratic by picking two of the three variables and replacing their product by a new variable. For example, if you define a new binary variable z[b,c,t] to replace x[b,t] * x[c,t-1], you can write

```
var z {t in 2..card(POS), b in PATIENTS, c in PATIENTS} binary;
subject to zDefn {t in 2..card(POS), b in PATIENTS, c in PATIENTS}:
   z[b,c,t] = x[b,t] * x[c,t-1];
```

Then write your constraint A2 as `z[b,c,t] * y[a,b] = 2 * x[i,t]`. There are two other possibilities, corresponding to the two other ways you can pick two of the three variables.

You can also linearize the cubic term directly. In that case, you would define a new binary variable z[a,b,c,t] to replace x[b,t] * x[c,t-1] * y[a,b], and you would add the following four constraints:

```
z[a,b,c,t] >= x[b,t] + x[c,t-1] + y[a,b] - 2
z[a,b,c,t] <= x[b,t]
z[a,b,c,t] <= x[c,t-1]
z[a,b,c,t] <= y[a,b]
```

Motivation

Typical *gurobipy* User Complaint

TypeError: unsupported operand type(s) for *: 'int' and 'GenExprMax'

Follow 2

Answered

Hi

I'm trying to solve a production problem. when the x change, it will cost a different additional cost. I need to compare the (x[i] -x[i-1]) with 0. how can I solve this.

```
production_change_cost = gp.quicksum(3 * gp.max_(0,(x[i] - x[i-1] for i in periods)) \
                                     + 0.8 * gp.max_(0,(x[i-1] - x[i] for i in periods)))
```

Motivation

Typical *gurobipy* Response

General constraints are meant to be used to define single constraints. It is not possible to use these constructs in other expressions, i.e., it is not possible to use gp.max_ in a more complex constraint other than y = gp.max_.

Moreover, as described in the documentation of the addGenConstrMax method, gp.max_ only accepts single variables as inputs. Thus, it is not possible to pass expressions `x[i] -x[i-1]`. To achieve what you want, you have to introduce additional auxiliary variables `aux[i] = x[i] -x[i-1]` and additional equality constraints z1 = gp.max_ and z2 = gp.max_

```
aux1 = mod.addVars(periods, lb=-GRB.INFINITY, name="auxvar1")
aux2 = mod.addVars(periods, lb=-GRB.INFINITY, name="auxvar2")
# are you sure that i-1 does not lead to a wrong key access?
m.addConstrs((aux1[i] = x[i]-x[i-1| for i in periods), name = "auxconstr1")
m.addConstrs((aux2[i] = x[i-1]-x[i| for i in periods), name = "auxconstr2")
z1 = m.addVar(lb = -GRB.INFINITY, n
z2 = m.addVar(lb = -GRB.INFINITY, n
m.addConstr(z1 = gp.max_(0,aux1),na
m.addConstr(z2 = gp.max_(0,aux2),name="maxconstr2")
[...]
production_change_cost = gp.quicksum(3 * z1 + 0.8 * z2)
```

```
  3 * max(0, {i in periods} x[i] - x[i-1]) +
0.8 * max(0, {i in periods} x[i-1] - x[i])
```

Formulating

Supported Extensions and Solvers

Operators and functions

- Conditional: `if-then-else`; `==>`, `<==`, `<==>`
- Logical: `or`, `and`, `not`; `exists`, `forall`
- Piecewise linear: `abs`; `min`, `max`; `<<breakpoints; slopes>>`
- Counting: `count`; `atmost`, `atleast`, `exactly`; `numberof`
- Comparison: `>`, `<`, `!=`; `alldiff`
- Complementarity: `complements`
- Nonlinear: `*`, `/`, `^`; `exp`, `log`; `sin`, `cos`, `tan`; `sinh`, `cosh`, `tanh`
- Set membership: `in`

Expressions and constraints

- High-order polynomials
- Second-order and exponential cones

Formulating

Extensions

Conditional operators

- **if** *constraint* **then** *var-expr1* [**else** *var-expr2*]
- *constraint1* ==> *constraint2* [**else** *constraint3*]
  constraint1 <== *constraint2*
  constraint1 <==> *constraint2*

```
minimize TotalCost:
   sum {j in JOBS, k in MACHINES}
      if MachineForJob[j] = k then cost[j,k];
```

```
subject to Multi_Min_Ship {i in ORIG, j in DEST}:
   sum {p in PROD} Trans[i,j,p] >= 1 ==>
      minload <= sum {p in PROD} Trans[i,j,p] <= limit[i,j];
```

Formulating

Extensions

Logical operators

- *constraint1* **or** *constraint2*
  constraint1 **and** *constraint2*
  not *constraint2*
- **exists** {*indexing*} *constraint-expr*
  forall {*indexing*} *constraint-expr*

```
subject to NoMachineConflicts
      {m1 in 1..nMach, m2 in m1+1..nMach, j in 1..nJobs}:
   Start[m1,j] + duration[m1,j] <= Start[m2,j] or
   Start[m2,j] + duration[m2,j] <= Start[m1,j];
```

```
subj to HostNever {j in BOATS}:
   isH[j] = 1 ==> forall {t in TIMES} H[j,t] = j;
```

Extensions

Piecewise-linear functions and operators

- `<<` *breakpoint-list* `;` *slope-list* `>>` *variable*
  `<<` *breakpoint-list* `;` *slope-list* `>>` `(`*variable* `,` *zero-point*`)`
- `abs(`*var-expr*`)`
  `min(`*var-expr-list*`)` `min {`*indexing*`}` *var-expr*
  `max(`*var-expr-list*`)` `max {`*indexing*`}` *var-expr*

```
maximize WeightSum:
   sum {t in TRAJ} max {n in NODE} weight[t,n] * Use[n];
```

```
minimize Total_Cost:
   sum {i in ORIG, j in DEST}
      <<{p in 1..npiece[i,j]-1} limit[i,j,p];
        {p in 1..npiece[i,j]} rate[i,j,p]>> Trans[i,j];
```

Formulating

Extensions

Piecewise-linear functions and operators

- `<<` *breakpoint-list* `;` *slope-list* `>>` *variable*
  `<<` *breakpoint-list* `;` *slope-list* `>>` `(`*variable* `,` *zero-point*`)`
- **`abs`**`(`*var-expr*`)`
  `min``(`*var-expr-list*`)`   **`min`** `{`*indexing*`}` *var-expr*
  `max``(`*var-expr-list*`)`   **`max`** `{`*indexing*`}` *var-expr*

```
x = mod.addVars(periods)

production_change_cost = \
  gp.quicksum(3.0 * gp.max_(0,(x[i] - x[i-1] for i in periods)) \
            + 0.8 * gp.max_(0,(x[i-1] - x[i] for i in periods)))
```

```
var x {0..T} >= 0;

var production_change_cost =
   3.0 * max(0, {i in 1..T} x[i] - x[i-1]) +
   0.8 * max(0, {i in 1..T} x[i-1] - x[i]);
```

Formulating

Extensions

Counting operators

- **count** {*indexing*} (*constraint-expr*)
- **atmost** *k* {*indexing*} (*constraint-expr*)
  atleast *k* {*indexing*} (*constraint-expr*)
  exactly *k* {*indexing*} (*constraint-expr*)
- **numberof** *k* **in** (*var-expr-list*)

```
subject to Limit_Used:
  count {(i,j) in ARCS}
    (sum {p in PRODUCTS} Flow[p,i,j] > 0) <= max_arcs;
```

```
subj to CapacityOfMachine {k in MACHINES}:
   numberof k in ({j in JOBS} MachineForJob[j]) <= cap[k];
```

Formulating

Extensions

Comparison operators

- *var-expr1* **!=** *var-expr2*
  var-expr1 **>** *var-expr2*
  var-expr1 **<** *var-expr2*
- **alldiff(***var-expr-list***)**
  alldiff {*indexing***}** *var-expr*

```
subj to Different_Colors {(c1,c2) in Neighbors}:
   Color[c1] != Color[c2];
```

```
subject to OnePersonPerPosition:
   alldiff {i in 1..nPeople} Pos[i];
```

Formulating

Extensions

Complementarity operators

- *single-inequality1* **complements** *single-inequality2*
- *double-inequality* **complements** *var-expr*
  var-expr **complements** *double-inequality*

```
subject to Pri_Compl {i in PROD}:
   max(500.0, Price[i]) >= 0 complements
      sum {j in ACT} io[i,j] * Level[j] >= demand[i];
```

```
subject to Lev_Compl {j in ACT}:
   level_min[j] <= Level[j] <= level_max[j] complements
      cost[j] - sum {i in PROD} Price[i] * io[i,j];
```

Formulating

Extensions

Nonlinear expressions and operators

- *var-expr1* ***** *var-expr2*
  var-expr1 **/** *var-expr2*
  var-expr **^** *k*
- **exp**(*var-expr*) **log**(*var-expr*)
  sin(*var-expr*) **cos**(*var-expr*) **tan**(*var-expr*)

```
subj to Eq {i in J} :
   x[i+neq] / (b[i+neq] * sum {j in J} x[j+neq] / b[j+neq]) =
   c[i] * x[i] / (40 * b[i] * sum {j in J} x[j] / b[j]);
```

```
minimize Chichinadze:
     x[1]^2 - 12*x[1] + 11 + 10*cos(pi*x[1]/2)
   + 8*sin(pi*5*x[1]) - exp(-(x[2]-.5)^2/2)/sqrt(5);
```

Formulating

Extensions

Discrete variable domains

- **var** *varname* **{***indexing***}** **in** *set-expr* **;**

```
var Buy {f in FOODS} in {0,10,30,45,55};
```

```
var Ship {(i,j) in ARCS}
   in {0} union interval[min_ship,capacity[i,j]];
```

```
var Work {j in SCHEDS}
   in {0} union integer[least, max {i in SHIFT_LIST[j]} req[i]];
```

MP Interface

General use with MIP solvers

Read objectives & constraints from AMPL

- Store initially as linear coefficients + expression graphs
- Analyze trees to determine if linearizable

Generate linearizations

- Walk trees to build linearizations (flatten)
- Define auxiliary variables (usually zero-one)
- Generate equivalent constraints

Solve

- Send to solver through its API
- Convert optimal solution back to the original AMPL variables
- Write solution to AMPL

Special Alternatives in *Gurobi*

Apply our linearization (`count`)

- Use Gurobi's linear API

Have Gurobi linearize (`or`, `abs`)

- Simplify and "flatten" the expression tree
- Use Gurobi's "general constraint" API
  - **`addGenConstrOr`** (resbinvar, [binvars])
    tells Gurobi: resbinvar = 1 iff at least one item in [binvars] = 1
  - **`addGenConstrAbs`** (resvar, argvar)
    tells Gurobi: resvar = |argvar|

Send univariate nonlinearities to Gurobi (`log`)

- Replace by piecewise-linear approximations ***or***
  solve using generalized branching search (new in version 11!)
- Use Gurobi's "function constraint" API
  - **`addGenContstrLog`** (xvar, yvar)
    tells Gurobi: yvar = [piecewise-linear approximation of] log(xvar)

Formulating

Implementation Issues

Is an expression repeated?

- Detect common subexpressions

```
subject to Shipment_Limits {(i,j) in ARCS}:

sum {p in PRODUCTS} Flow[p,i,j] = 0 or
min_ship <= sum {p in PRODUCTS} Flow[p,i,j] <= capacity[i,j];
```

Is there an easy reformulation?

- Yes for min-max, no for max-max

```
minimize Worst_Rank:
  max {i in PEOPLE} sum {j in PROJECTS} rank[i,j] * Assign[i,j];
```

```
maximize Max_Value:
  sum {t in T} max {n in N} weight[t,n] * Value[n];
```

Formulating

Implementation Issues *(cont'd)*

Does an exact linearization exist?

- Yes if constraint set is "closed"
- No if constraint set is "open"

```
var Flow {ARCS} >= 0;
var Use {ARCS} binary;

subj to Use_Definition {(i,j) in ARCS}:
   Use[i,j] = 0 ==> Flow[i,j] = 0;
```

```
subj to Use_Definition {(i,j) in ARCS}:
   Flow[i,j] = 0 ==> Use[i,j] = 0 else Use[i,j] = 1;
```

Implementation Issues *(cont'd)*

Does an exact linearization exist?

- Yes if constraint set is "closed"
- No if constraint set is "open"

```
var Flow {ARCS} >= 0;
var Use {ARCS} binary;

subj to Use_Definition {(i,j) in ARCS}:
   Use[i,j] = 0 ==> Flow[i,j] = 0 else Flow[i,j] >= 0;
```

```
subj to Use_Definition {(i,j) in ARCS}:
   Use[i,j] = 0 ==> Flow[i,j] = 0 else Flow[i,j] > 0;
```

Formulating

Solver Efficiency Issues

Bounds on subexpressions

- Define auxiliary variables that can be bounded

```
var x {1..2} <= 2, >= -2;

minimize Goldstein-Price:
  (1 + (x[1] + x[2] + 1)^2
    * (19 - 14*x[1] + 3*x[1]^2 - 14*x[2] + 6*x[1]*x[2] + 3*x[2]^2))
* (30 + (2*x[1] - 3*x[2])^2
    * (18 - 32*x[1] + 12*x[1]^2 + 48*x[2] - 36*x[1]*x[2] + 27*x[2]^2));
```

```
var t1 >= 0, <= 25;   subj to t1def: t1 = (x[1] + x[2] + 1)^2;
var t2 >= 0, <= 100;  subj to t2def: t2 = (2*x[1] - 3*x[2])^2;

minimize Goldstein-Price:
  (1 + t1
    * (19 - 14*x[1] + 3*x[1]^2 - 14*x[2] + 6*x[1]*x[2] + 3*x[2]^2))
* (30 + t2
    * (18 - 32*x[1] + 12*x[1]^2 + 48*x[2] - 36*x[1]*x[2] + 27*x[2]^2));
```

Formulating

Solver Efficiency Issues *(cont'd)*

Simplification of logic

- Replace an iterated **exists** with a **sum**

```
minimize TotalCost: ...
  sum {(i,j) in ARCS}
    if exists {p in PRODUCTS} Flow[p,i,j] > 0 then fix_cost[i,j];
```

```
minimize TotalCost: ...
  sum {(i,j) in ARCS}
    if sum {p in PRODUCTS} Flow[p,i,j] > 0 then fix_cost[i,j];
```

Formulating

Solver Efficiency Issues *(cont'd)*

Creation of common subexpressions

- Substitute a stronger bound from a constraint

```
subject to Shipment_Limits {(i,j) in ARCS}:
   sum {p in PRODUCTS} Flow[p,i,j] = 0 or
   min_ship <= sum {p in PRODUCTS} Flow[p,i,j] <= capacity[i,j];

minimize TotalCost: ...
   sum {(i,j) in ARCS}
      if sum {p in PRODUCTS} Flow[p,i,j] > 0
         then fix_cost[i,j];
```

```
minimize TotalCost: ...
   sum {(i,j) in ARCS}
      if sum {p in PRODUCTS} Flow[p,i,j] >= min_ship
         then fix_cost[i,j];
```

. . . consider automating all these improvements

Formulating

Solver Tolerance Issues

Solver tolerances are applied after automatic conversion

- Anomalous results are possible in rare circumstances

```
var x {1..2} >=0, <=100;

maximize Total:
   if x[1] <= 4.9999999 and x[2] >= 5.0000001
      then x[1] + x[2] else 0;

subj to con: x[1] = x[2];
```

```
ampl: solve;
Gurobi 10.0.2: optimal solution; objective 9.9999998

ampl: display x;
1  4.9999999
2  4.9999999

ampl: display Total;
Total = 0
```

Formulating

Solver Tolerance Issues *(cont'd)*

Warning added

- (but needs work)

```
var x {1..2} >=0, <=100;

maximize Total:
   if x[1] <= 4.9999999 and x[2] >= 5.0000001
      then x[1] + x[2] else 0;

subj to con: x[1] = x[2];
```

```
ampl: solve;
Gurobi 10.0.2: optimal solution; objective 9.9999998

------------ WARNINGS ------------
WARNING:  "Solution Check (Idealistic)“
     [ sol:chk:feastol=1e-06, :feastolrel=1e-06, :inttol=1e-05,
       :round='', :prec=‘’ ]
Objective value violations:
  - 1 objective value(s) violated,
        up to 1E+01 (abs)
Idealistic check is an indicator only, see documentation.
```

AMPL Modeling Environments

Native

- Interactive command line
- Model, data, and script ("run") files

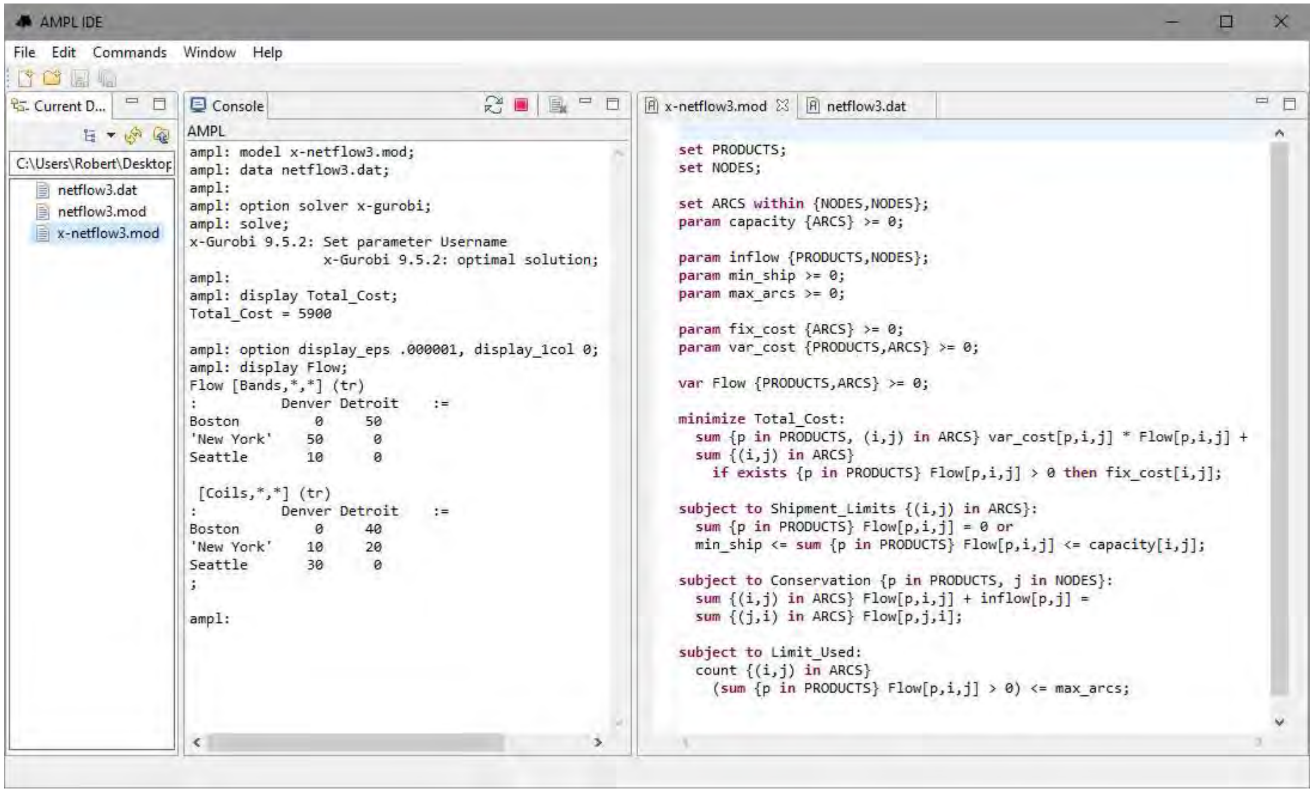
```
File  Edit  Help
sw: ampl
ampl: model x-netflow3.mod;
ampl: data netflow3.dat;

ampl: option solver gurobi;
ampl: solve;
Gurobi 11.0.0: optimal solution; objective 5900
10 simplex iterations
1 branching nodes
absmipgap=9.09495e-13, relmipgap=0

ampl: option display_eps .000001, display_1col 0;

ampl: display Flow;

Flow [Bands,*,*] (tr)
:           Denver Detroit    :=
Boston          0      50
'New York'     50       0
Seattle        10       0

 [Coils,*,*] (tr)
:           Denver Detroit    :=
Boston          0      40
'New York'     10      20
Seattle        30       0
;

ampl:
```

AMPL Modeling Environments

Native

IDEs

- ❖ AMPL IDE, Vscode



AMPL Modeling Environments

Native

IDEs

APIs

- C++, C#, Java, MATLAB, *Python*, R

amplpy

- Jupyter notebooks
- AMPL model colaboratory . . .

AMPL Data Interfaces

Native

- Plain text format for AMPL sets & parameters

Spreadsheets

- Excel (*.xlsx*) format
- Comma-separated value (.csv) format

Databases

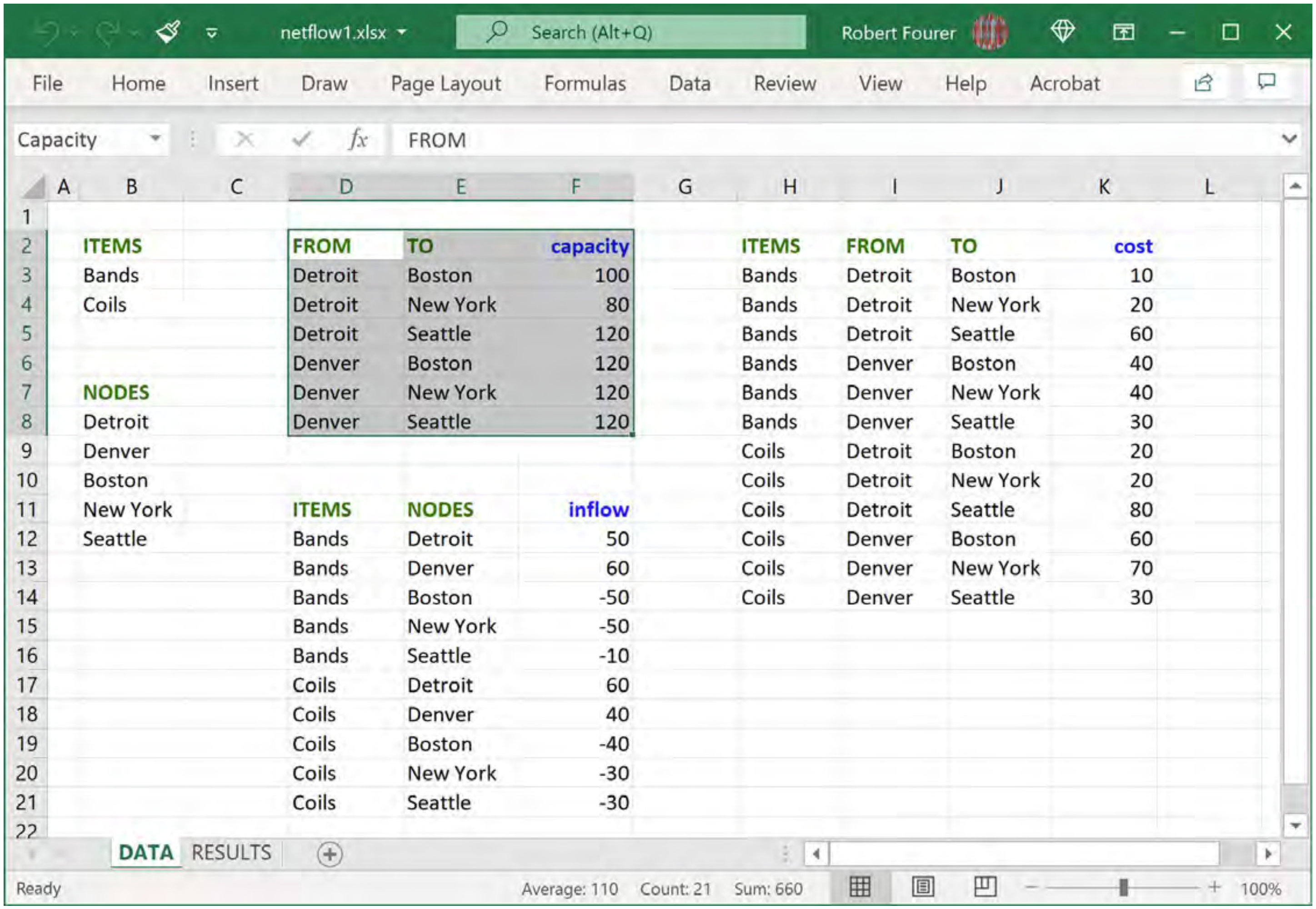
- Relational databases via ODBC interface

Python

- Lists, sets, tuples, dictionaries
- Pandas DataFrames . . .

Direct Spreadsheet Interface

"1D" spreadsheet ranges



	A	B	C	D	E	F	G	H	I	J	K	L
1												
2		ITEMS		FROM	TO	capacity		ITEMS	FROM	TO	cost	
3		Bands		Detroit	Boston	100		Bands	Detroit	Boston	10	
4		Coils		Detroit	New York	80		Bands	Detroit	New York	20	
5				Detroit	Seattle	120		Bands	Detroit	Seattle	60	
6				Denver	Boston	120		Bands	Denver	Boston	40	
7		NODES		Denver	New York	120		Bands	Denver	New York	40	
8		Detroit		Denver	Seattle	120		Bands	Denver	Seattle	30	
9		Denver						Coils	Detroit	Boston	20	
10		Boston						Coils	Detroit	New York	20	
11		New York		ITEMS	NODES	inflow		Coils	Detroit	Seattle	80	
12		Seattle		Bands	Detroit	50		Coils	Denver	Boston	60	
13				Bands	Denver	60		Coils	Denver	New York	70	
14				Bands	Boston	-50		Coils	Denver	Seattle	30	
15				Bands	New York	-50						
16				Bands	Seattle	-10						
17				Coils	Detroit	60						
18				Coils	Denver	40						
19				Coils	Boston	-40						
20				Coils	New York	-30						
21				Coils	Seattle	-30						
22												

Spreadsheet interface

Data Handling

Script (input)

```
model x-netflow3.mod;

table Products IN "amplxl" "netflow2.xlsx" "Items":
    PRODUCTS <- [ITEMS];

table Nodes IN "amplxl" "netflow2.xlsx":
    NODES <- [NODES];

table Capacity IN "amplxl" "netflow2.xlsx":
    ARCS <- [FROM,TO], capacity;

table Inflow IN "amplxl" "netflow2.xlsx":
    [ITEMS,NODES], inflow;

table Cost IN "amplxl" "netflow2.xlsx":
    [ITEMS,FROM,TO], cost;

load amplxl.dll;

read table Products; read table Nodes;
read table Capacity; read table Inflow; read table Cost;
```

Spreadsheet interface

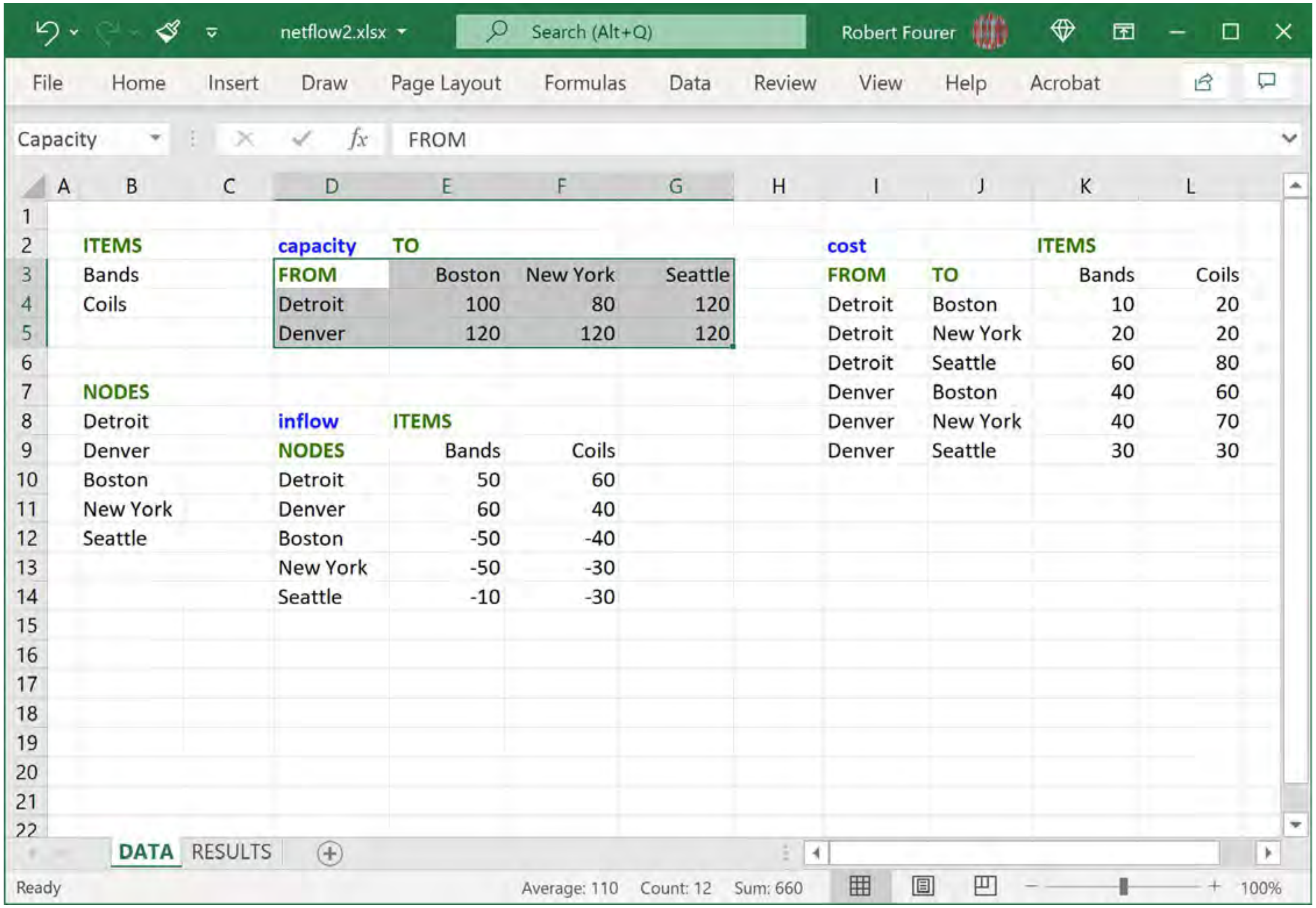
Data Handling

Script (input)

```
model x-netflow3.mod;

table Products IN "amplxl" "netflow2.xlsx" "Items":
    PRODUCTS <- [ITEMS];

table Nodes IN "amplxl" "netflow2.xlsx":
    NODES <- [NODES];

table Capacity IN "amplxl" "netflow2.xlsx" "2D":
    ARCS <- [FROM,TO], capacity;

table Inflow IN "amplxl" "netflow2.xlsx" "2D":
    [ITEMS,NODES], inflow;

table Cost IN "amplxl" "netflow2.xlsx" "2D":
    [ITEMS,FROM,TO], cost;

load amplxl.dll;

read table Products; read table Nodes;
read table Capacity; read table Inflow; read table Cost;
```

Direct Spreadsheet Interface

"2D" spreadsheet ranges



Data Handling

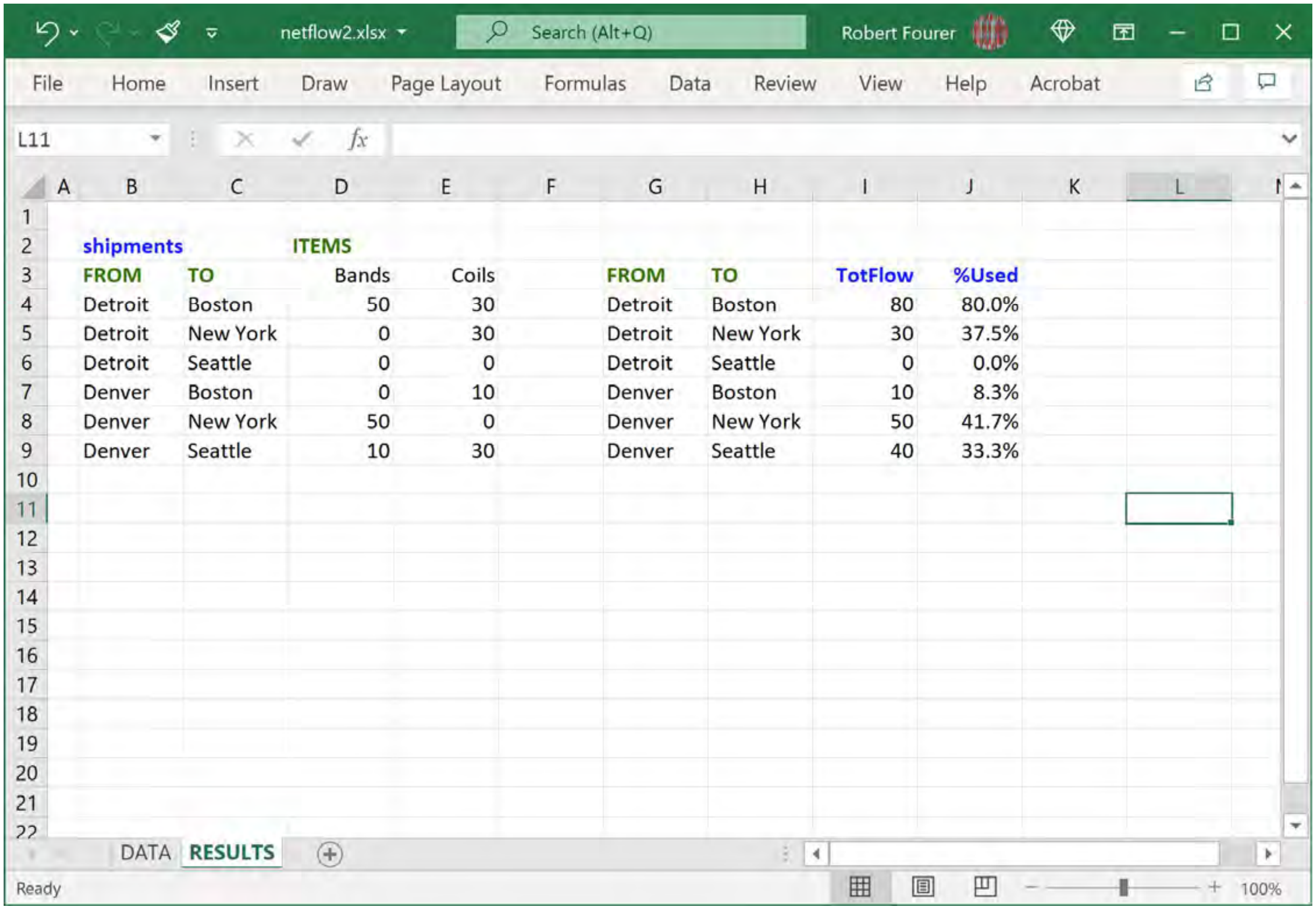
Script (output)

```
option solver gurobi;
solve;

table Results OUT "amplxl" "netflow1.xlsx" "2D":
    [ITEMS,FROM,TO], Flow;

table Summary OUT "amplxl" "netflow1.xlsx":
    {(i,j) in ARCS} -> [FROM,TO],
    sum {p in PRODUCTS} Flow[p,i,j] ~ TotFlow,
    sum {p in PRODUCTS} Flow[p,i,j] / capacity[i,j] ~ "%Used";

write table Results;
write table Summary;
```

Spreadsheet interface

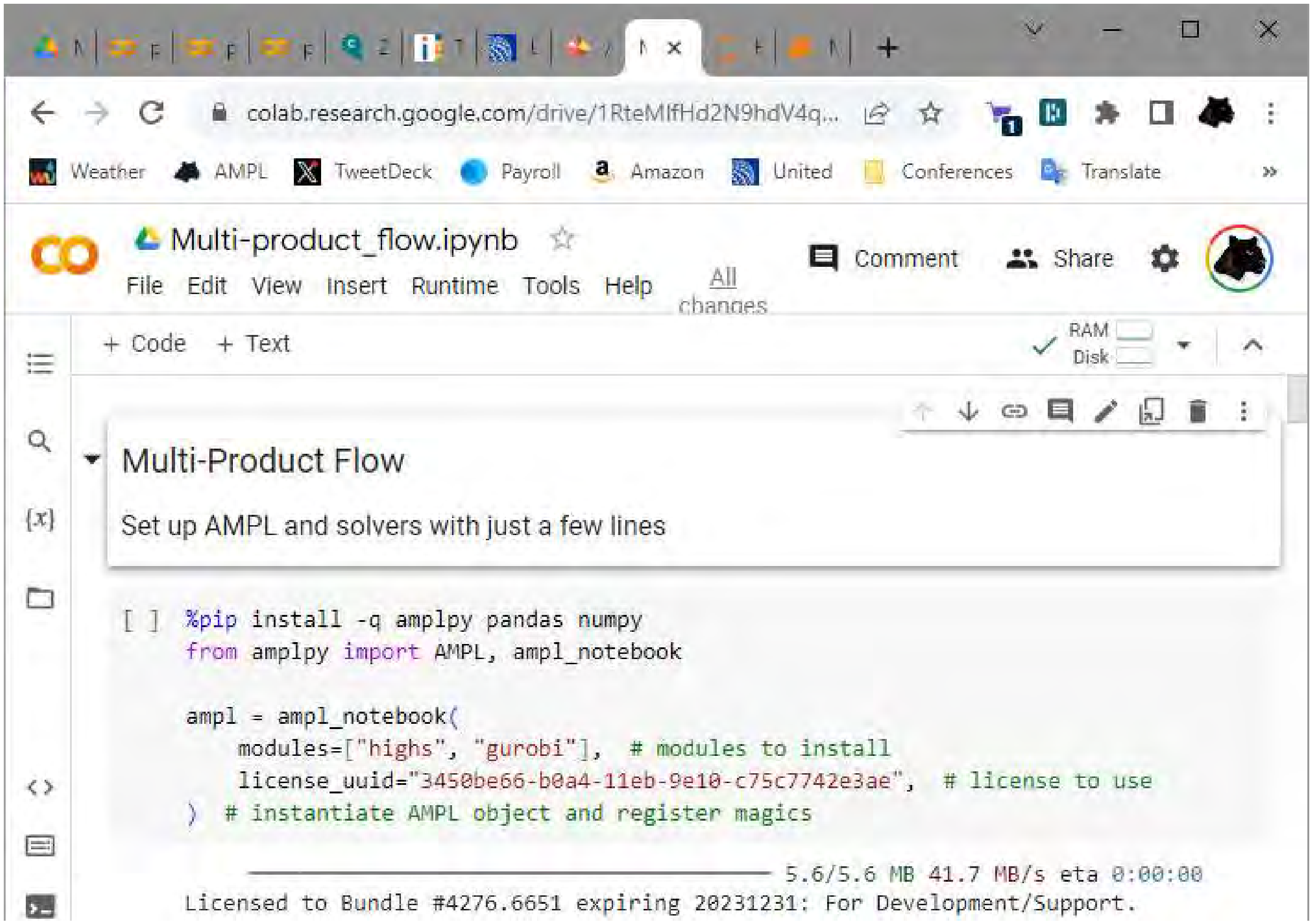
Data Results

“2D” spreadsheet range



shipments		ITEMS					
FROM	TO	Bands	Coils	FROM	TO	TotFlow	%Used
Detroit	Boston	50	30	Detroit	Boston	80	80.0%
Detroit	New York	0	30	Detroit	New York	30	37.5%
Detroit	Seattle	0	0	Detroit	Seattle	0	0.0%
Denver	Boston	0	10	Denver	Boston	10	8.3%
Denver	New York	50	0	Denver	New York	50	41.7%
Denver	Seattle	10	30	Denver	Seattle	40	33.3%

AMPL Python API *(in Google Colab)*



https://colab.research.google.com/drive/1RteMlfHd2N9hdV4q7luEf5X9ElgxeYR0?usp=sharing

Colaboratory

AMPL Model in Notebook Cell

colab.research.google.com/drive/1RteMIfHd2N9hdV4q...

\+ Code + Text All changes saved

RAM
Disk

AMPL model

Using new logical operators (no binary variables)

```
[ ] %%ampl_eval
    set PRODUCTS;
    set NODES;

    set ARCS within {NODES,NODES};
    param capacity {ARCS} >= 0;
```

```
    subject to Shipment_Limits {(i,j) in ARCS}:
        sum {p in PRODUCTS} Flow[p,i,j] = 0 or
        min_ship <= sum {p in PRODUCTS} Flow[p,i,j] <= capacity[i,j];

    subject to Conservation {p in PRODUCTS, j in NODES}:
        sum {(i,j) in ARCS} Flow[p,i,j] + inflow[p,j] =
        sum {(j,i) in ARCS} Flow[p,j,i];

    subject to Limit_Used:
        atmost max_arcs {(i,j) in ARCS}
            (sum {p in PRODUCTS} Flow[p,i,j] > 0);
```

Colaboratory

Python Data for the Model

colab.research.google.com/drive/1RteMlfHd2N9hdV4q...

\+ Code + Text All changes saved

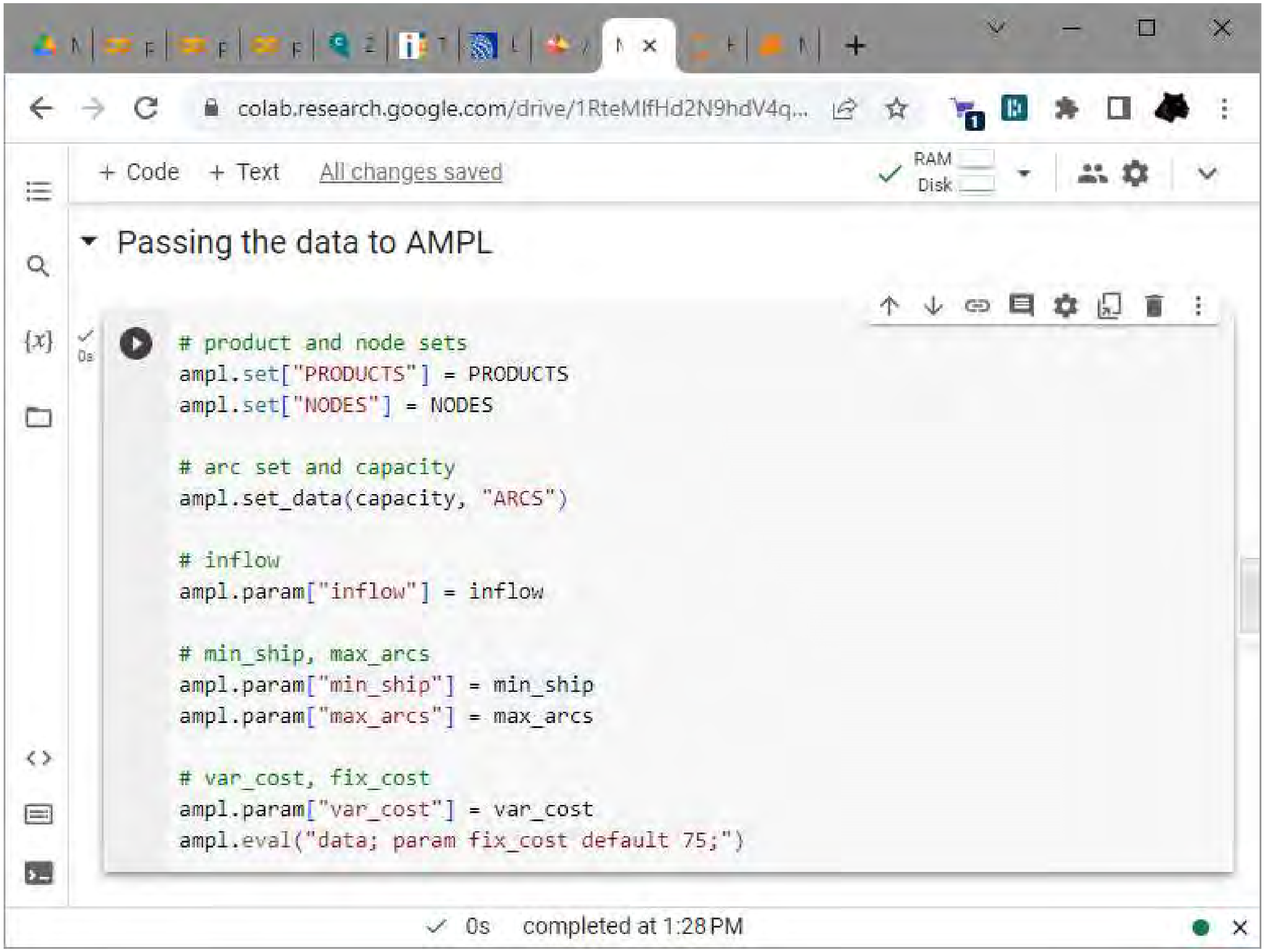
Data for an instance of the model

In a large-scale application, this would be read or derived from data sources external to the notebook.

```python
import pandas as pd

PRODUCTS = ["Bands", "Coils"]
NODES = ["Detroit", "Denver", "Boston", "New York", "Seattle"]

capacity = pd.DataFrame(
    [
        [100, 80, 120],
        [120, 120, 120],
    ],
    columns=["Boston", "New York", "Seattle"],
    index=["Detroit", "Denver"],
).stack().to_frame(name="capacity")

inflow = pd.DataFrame(
```

Colaboratory

Passing the Data to AMPL



Colaboratory

Invoking the Solver

```
[5] # solve
    ampl.option["solver"] = "gurobi"
    ampl.solve()

    # retrieve solution
    ampl.option["display_1col"] = "0"
    ampl.eval("display Flow;")

    ampl.var["Flow"].to_pandas()
```

```
Gurobi 10.0.3: optimal solution; objective 5900
10 simplex iterations
1 branching nodes

Flow [Bands,*,*] (tr)
:          Denver Detroit    :=
Boston        0      50
'New York'   50       0
Seattle      10       0

 [Coils,*,*] (tr)
:          Denver Detroit    :=
Boston        0      40
'New York'   10      20
Seattle      30       0
```