Adding Optimization to Your Applications

Quickly and Reliably

1. A Guide to Model-Based Optimization

2. From Prototyping to Integration with AMPL

Robert Fourer

4er@ampl.com

AMPL Optimization Inc.

www.ampl.com — +1 773-336-AMPL

INFORMS Business Analytics Conference

Technology Workshop, 3 April 2022
Adding Optimization to Your Applications, Quickly and Reliably: From Prototyping to Integration with AMPL

Optimization is the most widely adopted technology of Prescriptive Analytics, but also the most challenging to implement:

• How can you prototype an optimization application fast enough to get results before the problem owner loses interest?
• How can you develop optimization-based procedures to get results you can use, within your time and resource requirements?
• How can you integrate optimization into your enterprise’s decision-making systems?

In this presentation, we show how AMPL gets you going without elaborate training, extra programmers, or premature commitments. We start by introducing model-based optimization, the key approach to streamlining the optimization modeling cycle and building successful applications today. Then we demonstrate how AMPL’s design of a language and system for model-based optimization is able to offer exceptional power of expression while maintaining ease of use.

The remainder of the presentation takes a single example through successive stages of the optimization modeling lifecycle:

• Prototyping in an interactive command environment.
• Development of optimization procedures via AMPL’s built-in scripting language.
• Integration through APIs to widely used programming languages including C++, C#, Java, and MATLAB, and featuring the popular data science languages Python and R.

Our example is simple enough for participants to follow its development through the course of this short workshop, yet rich enough to serve as a foundation for appreciating model-based optimization in practice.
Optimization
Mathematical optimization

From Wikipedia, the free encyclopedia

"Mathematical programming" redirects here. For the peer-reviewed journal, see Mathematical Programming.

"Optimization" and "Optimum" redirect here. For other uses, see Optimization (disambiguation) and Optimum (disambiguation).

In mathematics, computer science and operations research, mathematical optimization (alternatively spelled optimisation or mathematical programming) is the selection of a best element (with regard to some criterion) from some set of available alternatives. [1]

In the simplest case, an optimization problem consists of maximizing or minimizing a real function by systematically choosing input values from within an allowed set and computing the value of the function. The generalization of optimization theory and techniques to other formulations constitutes a large area of applied mathematics. More generally, optimization includes finding "best available" values of some objective function given a defined domain (or input), including a variety of different types of objective functions and different types of domains.

Contents [hide]
1 Optimization problems
2 Notation
  2.1 Minimum and maximum value of a function
  2.2 Optimal input arguments
3 History
4 Major subfields
  4.1 Multi-objective optimization
  4.2 Multi-modal optimization
5 Classification of critical points and extrema
  5.1 Feasibility problem
  5.2 Existence
  5.3 Necessary conditions for optimality
  5.4 Sufficient conditions for optimality
  5.5 Sensitivity and continuity of optima
  5.6 Calculus of optimization
6 Computational optimization techniques

Graph of a paraboloid given by \( z = f(x, y) = -(x^2 + y^2) + 4 \). The global maximum at \((x, y, z) = (0, 0, 4)\) is indicated by a blue dot.

Nelder-Mead minimum search of Simionescu's function. Simplices vertices are ordered by their value, with 1 having the lowest (best) value.
Mathematical Optimization

In general terms,

- Given an objective function of some decision variables
- Choose values of the variables to make the objective as large or as small as possible
- Subject to restrictions on the values of the variables

In practice,

- A paradigm for a very broad variety of decision problems
- A practical approach to making decisions
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)
Optimization in Practice

*Large numbers of decision variables*
  - Thousands to millions

*An objective function*
  - To be minimized or maximized

*Various constraint types*
  - 10-20 distinct types, many of each type
  - Few variables involved in each constraint

*Solved many times with different data*
  - Simple rules can’t capture all possibilities in advance
  - Number of “iterations” for each solve is hard to predict
Outline

1. Model-based optimization
   - Comparison of method-based and model-based approaches
   - Approaches to model-based optimization
   - Algebraic modeling languages: try AMPL
   - Ready-to-run solvers

2. From prototyping to integration

3. Case studies
**Example**: Balanced Assignment

**Motivation**
- meeting of employees from around the world

**Given**
- several employee categories
  - (title, location, department, male/female)
- a specified number of project groups

**Assign**
- each employee to a project group

**So that**
- the groups have about the same size
- the groups are as “varied” as possible with respect to all categories
**Balanced Assignment**

**Method-Based Approach**

**Define an algorithm to build a balanced assignment**

- Start with all groups empty
- Make a list of people (employees)
- For each person in the list:
  - Add to the group whose resulting “sameness” will be least

```plaintext
Initialize all groups G = { }

Repeat for each person p
   sMin = Infinity
   
   Repeat for each group G
   s = total "sameness" in G ∪ {p} 
   
   if s < sMin then
     sMin = s
     GMin = G 
   
   GMin = GMin ∪ {p}
```
Balanced Assignment

Method-Based Approach (cont’d)

Define a computable concept of “sameness”

- Sameness of a pair of people:
  - Number of categories in which they are the same
- Sameness in a group:
  - Sum of the sameness of all pairs of people in the group

Refine the algorithm to get better results

- Reorder the list of people
- Locally improve the initial “greedy” solution by swapping group members
- Seek further improvement through local search metaheuristics
  - What are the neighbors of an assignment?
  - How can two assignments combine to create a better one?
Balanced Assignment

Model-Based Approach

Formulate a “minimal sameness” model

- Define decision variables for assignment of people to groups
  * $x_{ij} = 1$ if person 1 assigned to group $j$
  * $x_{ij} = 0$ otherwise

- Specify valid assignments through constraints on the variables

- Formulate sameness as an objective to be minimized
  * Total sameness = sum of the sameness of all groups

Send to a ready-to-run solver

- Many excellent alternatives are available
- Broad problem classes are handled efficiently
- Special cases are recognized and exploited to advantage
  * zero-one variables like $x_{ij}$
Balanced Assignment

Model-Based Formulation

Given

- $P$ set of people
- $C$ set of categories of people
- $t_{ik}$ type of person $i$ within category $k$, for all $i \in P, k \in C$

and

- $G$ number of groups
- $g_{\text{min}}$ lower limit on people in a group
- $g_{\text{max}}$ upper limit on people in a group

Define

$$s_{i_1i_2} = |\{k \in C: t_{i_1k} = t_{i_2k}\}|, \text{ for all } i_1 \in P, i_2 \in P$$

sameness of persons $i_1$ and $i_2$
Model-Based Formulation (cont’d)

Determine
\[ x_{ij} \in \{0,1\} \quad \text{if person } i \text{ is assigned to group } j \]
\[ = 0 \text{ otherwise, for all } i \in P, j = 1, \ldots, G \]

To minimize
\[ \sum_{i_1 \in P} \sum_{i_2 \in P} s_{i_1i_2} \sum_{j=1}^{G} x_{i_1j} x_{i_2j} \]
\[ \text{total sameness of all pairs of people in all groups} \]

Subject to
\[ \sum_{j=1}^{G} x_{ij} = 1, \text{ for each } i \in P \]
\[ \text{each person must be assigned to one group} \]
\[ g_{\text{min}} \leq \sum_{i \in P} x_{ij} \leq g_{\text{max}}, \text{ for each } j = 1, \ldots, G \]
\[ \text{each group must be assigned an acceptable number of people} \]
Model-Based Solution

*Optimize with an off-the-shelf solver*

*Choose among many alternatives*

- Linearize and send to a mixed-integer linear solver
  - CPLEX, Gurobi, Xpress; CBC, MIPCL, SCIP
- Send quadratic formulation to a mixed-integer solver that automatically linearizes products involving binary variables
  - CPLEX, Gurobi, Xpress
- Send quadratic formulation to a nonlinear solver
  - Mixed-integer nonlinear: Knitro, BARON
  - Continuous nonlinear (might come out integer): MINOS, Ipopt, ...
Model-Based vs. Method-Based

Where is the work?

- **Method-based**: Programming an implementation of the method
- **Model-based**: Constructing a formulation of the model

Which should you prefer?

- For simple problems, any approach can seem pretty easy
- But real optimization problems are seldom simple . . .
Complications in Balanced Assignment

Client has trouble with “Total Sameness”
- Hard to relate to the goal of varied groups
- Minimize “total variation” instead
  - Sum over all types: most minus least assigned to any group

No employee should feel “isolated” within their group
- No group should have exactly one woman
- Every person should have a group-mate from the same location and of equal or adjacent rank

Room capacities are variable
- Different groups have different size limits
- Minimize “total deviation”
  - Sum over all types: greatest violation of target range for any group
**Balanced Assignment**

**Method-Based (cont’d)**

**Revise or replace the solution approach**

- Total variation objective is less suitable to a simple algorithm
- Isolation constraints are challenging to enforce

**Update or re-implement the method**

- Even small changes to the problem can necessitate major changes to the method and its implementation
Balanced Assignment

Model-Based (cont’d)

**Update the model**

- Replace the objective with “total variation”
- Add “isolation” constraints

**Re-run the solver**

- Total variation is actually easier
Balanced Assignment

Model-Based (cont’d)

To write new objective, add variables

\[ y_{kl}^{\text{min}} \] fewest people of category \( k \), type \( l \) in any group,
\[ y_{kl}^{\text{max}} \] most people of category \( k \), type \( l \) in any group,
for each \( k \in C \), \( l \in T_k = \bigcup_{i \in P} \{ t_{ik} \} \)

Add defining constraints

\[ y_{kl}^{\text{min}} \leq \sum_{i \in P : t_{ik} = l} x_{ij}, \text{ for each } j = 1, \ldots, G; \ k \in C, l \in T_k \]
\[ y_{kl}^{\text{max}} \geq \sum_{i \in P : t_{ik} = l} x_{ij}, \text{ for each } j = 1, \ldots, G; \ k \in C, l \in T_k \]

Minimize total variation

\[ \sum_{k \in C} \sum_{l \in T_k} (y_{kl}^{\text{max}} - y_{kl}^{\text{min}}) \]
Balanced Assignment

Model-Based (cont’d)

To express client requirement for women in a group, let

$$Q = \{i \in P : t_{i,m/f} = \text{female}\}$$

Add constraints

$$\sum_{i \in Q} x_{ij} = 0 \text{ or } \sum_{i \in Q} x_{ij} \geq 2, \text{ for each } j = 1, \ldots, G$$
Balanced Assignment

Model-Based (cont’d)

To express client requirement for women in a group, let

\[ Q = \{ i \in P : t_{i,m/f} = \text{female} \} \]

Define logic variables

\[ z_j \in \{0,1\} = 1 \text{ if any women assigned to group } j \]
\[ = 0 \text{ otherwise, for all } j = 1, \ldots, G \]

Add constraints relating logic variables to assignment variables

\[ z_j = 0 \Rightarrow \sum_{i \in Q} x_{ij} = 0, \]
\[ z_j = 1 \Rightarrow \sum_{i \in Q} x_{ij} \geq 2, \text{ for each } j = 1, \ldots, G \]
**Balanced Assignment**

**Model-Based (cont’d)**

*To express client requirement for women in a group, let*

\[ Q = \{ i \in P : t_{i,m/f} = \text{female} \} \]

**Define logic variables**

\[ z_j \in \{0,1\} = 1 \text{ if any women assigned to group } j \]
\[ = 0 \text{ otherwise, for all } j = 1, \ldots, G \]

**Linearize constraints relating logic variables to assignment variables**

\[ 2z_j \leq \sum_{i \in Q} x_{ij} \leq |Q| z_j, \text{ for each } j = 1, \ldots, G \]
To express client requirements for group-mates, let

\[ LR_{lr} = \{ i \in P : t_{i,\text{loc}} = l, t_{i,\text{rank}} = r \}, \text{ for all } l \in T_{\text{loc}}, r \in T_{\text{rank}} \]

\[ A_r \subseteq T_{\text{rank}}, \text{ set of ranks adjacent to rank } r, \text{ for all } r \in T_{\text{rank}} \]

Add constraints

\[ \sum_{i \in LR_{lr}} x_{ij} = 0 \text{ or } \sum_{i \in LR_{lr}} x_{ij} + \sum_{a \in A_r} \sum_{i \in LR_{la}} x_{ij} \geq 2, \]

for each \( l \in T_{\text{loc}}, r \in T_{\text{rank}}, j = 1, \ldots, G \)

Balanced Assignment
Balanced Assignment

Model-Based (cont’d)

To express client requirements for group-mates, let

\[ LR_{lr} = \{ i \in P : t_{i,loc} = l, t_{i,\text{rank}} = r \}, \text{ for all } l \in T_{\text{loc}}, r \in T_{\text{rank}} \]

\[ A_r \subseteq T_{\text{rank}}, \text{ set of ranks adjacent to rank } r, \text{ for all } r \in T_{\text{rank}} \]

Define logic variables

\[ w_{lrj} \in \{0,1\} \]

= 1 if group \( j \) has anyone from location \( l \) of rank \( r \)

= 0 otherwise, \text{ for all } l \in T_{\text{loc}}, r \in T_{\text{rank}}, j = 1, \ldots, G

Add constraints relating logic variables to assignment variables

\[ w_{lrj} = 0 \Rightarrow \sum_{i \in LR_{lr}} x_{ij} = 0, \]

\[ w_{lrj} = 1 \Rightarrow \sum_{i \in LR_{lr}} x_{ij} + \sum_{a \in A_r} \sum_{i \in LR_{la}} x_{ij} \geq 2, \]

for each \( l \in T_{\text{loc}}, r \in T_{\text{rank}}, j = 1, \ldots, G \)
Model-Based (*cont’d*)

**To express client requirements for group-mates, let**

\[ LR_{lr} = \{ i \in P : t_{i,loc} = l, \ t_{i,\text{rank}} = r \}, \text{ for all } l \in T_{\text{loc}}, r \in T_{\text{rank}} \]

\[ A_r \subseteq T_{\text{rank}}, \text{ set of ranks adjacent to rank } r, \text{ for all } r \in T_{\text{rank}} \]

**Define logic variables**

\[ w_{lrj} \in \{0,1\} = 1 \text{ if group } j \text{ has anyone from location } l \text{ of rank } r \]

\[ = 0 \text{ otherwise, for all } l \in T_{\text{loc}}, r \in T_{\text{rank}}, j = 1, \ldots, G \]

**Linearize constraints relating logic variables to assignment variables**

\[ w_{lrj} \leq \sum_{i \in LR_{lr}} x_{ij} \leq |LR_{lr}| \cdot w_{lrj}, \]

\[ \sum_{i \in LR_{lr}} x_{ij} + \sum_{a \in A_r} \sum_{i \in LR_{la}} x_{ij} \geq 2w_{lrj}, \]

\[ \text{for each } l \in T_{\text{loc}}, r \in T_{\text{rank}}, j = 1, \ldots, G \]
Method-Based Remains Popular for . . .

Applications of heuristic methods

- Simple heuristics
  - Greedy algorithms, local improvement methods
- Metaheuristics
  - Evolutionary methods, simulated annealing, tabu search, GRASP, . . .

Situations hard to formulate mathematically

- Intricate logical constraints
- Objectives computed by complex programs

Large-scale, intensive applications

- Routing huge fleets of delivery trucks
- Finding shortest routes in mapping apps
- Training huge neural networks

. . . and it appeals to programmers
Model-Based Has Become Common for . . .

Diverse industries

- Manufacturing, distribution, supply-chain management
- Air and rail operations, trucking, delivery services
- Medicine, medical services
- Refining, electric power flow, gas pipelines, hydropower
- Finance, e-commerce, . . .
Model-Based Has Become Common for . . .

*Diverse industries*

*Diverse fields*

- Operations research & management science
- Business analytics
- Engineering & science
- Economics
Model-Based Has Become Common for . . .

Diverse industries

Diverse fields

Diverse kinds of users

- Anyone who took an “optimization” class
- Anyone else with a technical background
- Newcomers to optimization

These have in common . . .

- Analysts inclined toward modeling; focus is
  - more on what should be solved
  - less on how it should be solved
- Good algebraic formulations for ready-to-run solvers
- Emphasis on fast prototyping and continued revision
Trends Favor Model-Based Optimization

Model-based approaches have spread
- Model-based metaheuristics ("Matheuristics")
- Solvers for SAT, planning, constraint programming

Ready-to-run optimization solvers have kept improving
- Solve the same problems faster and faster
- Handle broader problem classes
- Recognize special cases automatically

Optimization models have become easier to embed within broader methods
- Solver APIs that are model model-based
- APIs for optimization modeling systems
Approaches to Model-Based Optimization

*Translate between two forms of the problem*

- Modeler’s form
  - Symbolic description, easy for people to work with
- Solver’s form
  - Explicit data structure, easy for solvers to compute with

*Programming language approach*

- Write a *computer program* to generate the solver’s form

*Modeling language approach*

- Write the *model formulation* in a form that a computer can read and translate
Programming Language Approach

* Write a program to generate the solver’s form
  * Read data and compute objective & constraint coefficients
  * Send the solver the data structures it needs
  * Receive solution data structure for viewing or processing

* Some attractions
  * Ease of embedding into larger systems
  * Access to advanced solver features

* Serious disadvantages
  * Difficult environment for modeling
    * program does not resemble the modeler’s form
    * model is not separate from data
  * Very slow modeling cycle
    * hard to check the program for correctness
    * hard to distinguish modeling from programming errors
Modeling Language Approach

Use a computer language to describe the modeler’s form

- Write your model
- Prepare data for the model
- Let the computer translate to & from the solver’s form

Limited drawbacks

- Need to learn a new language
- Incur overhead in translation

Great advantages

- Faster modeling cycles
- More reliable modeling
- More maintainable applications

...even preferred by programmers
Approaches to Modeling Languages

Algebraic modeling languages

- Designed for “algebraic” formulations as seen in our model-based examples
- Excellent fit to many applications and many solvers

Executable approach

- Write a computer program . . .
  * that resembles an optimization model
  * that can be executed to drive a solver

Declarative approach

- Write a model description . . .
  * in a language specialized for optimization
  * that can be translated to the solver’s form
Example: Supply Chain Optimization

Executable approach: gurobipy
- Based on the Python programming language
  - Designed to look like algebraic notation
- Generates problems for the Gurobi solver

Declarative approach: AMPL
- Based directly on algebraic notation
  - Designed specifically for optimization
- Generates problems for Gurobi and other solvers
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 \((i,j) \in A\)
\[ s_{pj} \] supply/demand of product \( p \) at node \( j \), for each \( p \in P, j \in N \)
> 0 implies supply, < 0 implies demand
\[ c_{p(i,j)} \] cost per unit to ship product \( p \) on arc \((i,j)\), for each \( p \in P, (i,j) \in A \)
**Multi-Product Flow**

**Statements: Data**

**gurobipy**
- Assign values to Python lists and dictionaries

```python
products = ['Pencils', 'Pens']
nodes = ['Detroit', 'Denver', 'Boston', 'New York', 'Seattle']
arcs, capacity = multidict(
    ('Detroit', 'Boston'): 100,
    ('Detroit', 'New York'): 80,
    ('Detroit', 'Seattle'): 120,
    ('Denver', 'Boston'): 120,
    ('Denver', 'New York'): 120,
    ('Denver', 'Seattle'): 120 )
```

- Provide data later in a separate file

**AMPL**
- Define symbolic model sets and parameters

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

```AMPL
set PRODUCTS := Pencils Pens ;
set NODES := Detroit Denver Boston 'New York' Seattle ;
param: ARCS: capacity:  
    Boston 'New York' Seattle :=
    Detroit 100 80 120
    Denver 120 120 120 ;
```
Multi-Product Flow

Statements: Data (cont’d)

gurobipy

\[
\text{inflow} = \{
('Pencils', 'Detroit'): 50,
('Pencils', 'Denver'): 60,
('Pencils', 'Boston'): -50,
('Pencils', 'New York'): -50,
('Pencils', 'Seattle'): -10,
('Pens', 'Detroit'): 60,
('Pens', 'Denver'): 40,
('Pens', 'Boston'): -40,
('Pens', 'New York'): -30,
('Pens', 'Seattle'): -30
\}
\]

AMPL

\[
\text{param inflow \{COMMODITIES, NODES\};}
\]

\[
\text{param inflow (tr):}
\begin{array}{ccc}
\text{Pencils} & \text{Pens} \\
\text{Detroit} & 50 & 60 \\
\text{Denver} & 60 & 40 \\
\text{Boston} & -50 & -40 \\
\text{'New York'} & -50 & -30 \\
\text{Seattle} & -10 & -30 \\
\end{array}
\]

\]
Multi-Product Flow

Statements: Data (cont’d)

gurobipy

```python
cost = {
    ('Pencils', 'Detroit', 'Boston'): 10,
    ('Pencils', 'Detroit', 'New York'): 20,
    ('Pencils', 'Detroit', 'Seattle'): 60,
    ('Pencils', 'Denver', 'Boston'): 40,
    ('Pencils', 'Denver', 'New York'): 40,
    ('Pencils', 'Denver', 'Seattle'): 30,
    ('Pens', 'Detroit', 'Boston'): 20,
    ('Pens', 'Detroit', 'New York'): 20,
    ('Pens', 'Detroit', 'Seattle'): 80,
    ('Pens', 'Denver', 'Boston'): 60,
    ('Pens', 'Denver', 'New York'): 70,
    ('Pens', 'Denver', 'Seattle'): 30
}
```
Multi-Product Flow

Statements: Data (cont’d)

AMPL

```
param cost {COMMODITIES, ARCS} >= 0;

param cost {
    [Pencils,*,*] (tr) Detroit Denver :=
    Boston  10   40
    'New York'  20   40
    Seattle  60   30

    [Pens,*,*] (tr) Detroit Denver :=
    Boston  20   60
    'New York'  20   70
    Seattle  80   30
}
```
Multi-Product Flow

Formulation: Model

Determine

\[ X_{pij} \text{ amount of commodity } p \text{ to be shipped from node } i \text{ to node } j, \]
for each \( p \in P, (i, j) \in A \)

to minimize

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

total cost of shipping

subject to

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

total shipped on each arc must not exceed capacity

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

shipments in plus supply/demand must equal shipments out
Multi-Product Flow

Statements: Model
gurobipy

```python
m = Model('netflow')
flow = m.addVars(products, arcs, obj=cost, name="flow")
m.addConstrs(
    (flow.sum('*',i,j) <= capacity[i,j] for i,j in arcs), "cap")
m.addConstrs(
    (flow.sum(p,'*',j) + inflow[p,j] == flow.sum(p,j,'*')
     for p in products for j in nodes), "node")
```

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

Statements: Model

gurobipy

```python
m = Model('netflow')
flow = m.addVars(products, arcs, obj=cost, name="flow")
m.addConstrs(
    (flow.sum('*',i,j) <= capacity[i,j] for i,j in arcs), "cap")
m.addConstrs(
    (flow.sum(p,'*',j) + inflow[p,j] == flow.sum(p,j,'*')
     for p in products for j in nodes), "node")
```

for i,j in arcs:
    m.addConstr(sum(flow[p,i,j] for p in products) <= capacity[i,j],
               "cap[%s,%s]" % (i,j))

m.addConstrs(
    (quicksum(flow[p,i,j] for i,j in arcs.select('*',j)) + inflow[p,j] ==
     quicksum(flow[p,j,k] for j,k in arcs.select(j,'*'))
     for p in products for j in nodes), "node")
```
(Note on Summations)

**gurobipy quicksum**

```python
m.addConstrs(
    (quicksum(flow[p,i,j] for i,j in arcs.select('*',j)) + inflow[p,j] ==
     quicksum(flow[p,j,k] for j,k in arcs.select(j,'*'))
    for p in commodities for j in nodes), "node")
```

**quicksum (data)**

A version of the Python `sum` function that is much more efficient for building large Gurobi expressions (LinExpr or QuadExpr objects). The function takes a list of terms as its argument.

Note that while `quicksum` is much faster than `sum`, it isn't the fastest approach for building a large expression. Use `addTerms` or the `LinExpr()` constructor if you want the quickest possible expression construction.
Multi-Product Flow

Statements: Model (cont’d)

AMPL

```
var Flow {PRODUCTS, ARCS} >= 0;

minimize TotalCost:
    sum {p in PRODUCTS, (i,j) in ARCS} 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] + inflow[p,j] =
    sum {(j,i) in ARCS} Flow[p,j,i];
```

\[ \sum_{(i,j) \in A} X_{pij} + s_{pj} = \sum_{(j,i) \in A} X_{pji}, \text{ for all } p \in P, j \in N \]
Multi-Product Flow

Solution

gurobipy

```
m.optimize()

if m.status == GRB.Status.OPTIMAL:
    solution = m.getAttr('x', flow)
    for p in products:
        print('\nOptimal flows for %s:' % p)
        for i,j in arcs:
            if solution[p,i,j] > 0:
                print('%s -> %s: %g' % (i, j, solution[p,i,j]))
```

Solved in 0 iterations and 0.00 seconds
Optimal objective  5.500000000e+03

Optimal flows for Pencils:
Detroit -> Boston: 50
Denver -> New York: 50
Denver -> Seattle: 10

Optimal flows for Pens: ...
**Multi-Product Flow**

**Solution (cont’d)**

**AMPL**

```AMPL
ampl: model netflow.mod;
ampl: data netflow.dat;

ampl: option solver gurobi;
ampl: solve;

Gurobi 9.5.1: optimal solution; objective 5500
2 simplex iterations

ampl: display Flow;

Flow [Pencils,*,*]
  : Boston 'New York' Seattle :=
Denver   0 50 10
Detroit  50 0 0

[Pens,*,*]
  : Boston 'New York' Seattle :=
Denver   10 0 30
Detroit  30 30 0

;```

---

**Multi-Product Flow**

**Solution (cont’d)**

**AMPL**

```AMPL
ampl: model netflow.mod;
ampl: data netflow.dat;

ampl: option solver gurobi;
ampl: solve;

Gurobi 9.5.1: optimal solution; objective 5500
2 simplex iterations

ampl: display Flow;

Flow [Pencils,*,*]
  : Boston 'New York' Seattle :=
Denver   0 50 10
Detroit  50 0 0

[Pens,*,*]
  : Boston 'New York' Seattle :=
Denver   10 0 30
Detroit  30 30 0

;```
Multi-Product Flow

Solution (cont’d)

AMPL

```plaintext
AMPL: model netflow.mod;
AMPL: data netflow.dat;
AMPL: option solver cplex;
AMPL: solve;

CPLEX 20.1.0.0: optimal solution; objective 5500
0 dual simplex iterations (0 in phase I)
AMPL: display Flow;

Flow [Pencils,*,*]
: Boston 'New York' Seattle :=
Denver  0     50    10
Detroit 50     0     0

[Pens,*,*]
: Boston 'New York' Seattle :=
Denver 10     0     30
Detroit 30    30    0
```

Algebraic Modeling Languages

Executable

Concept

- Create an algebraic modeling language inside a general-purpose programming language
- Redefine operators like + and <= to return constraint objects rather than simple values

Advantages

- Complete application development in one language
- Direct access to advanced solver features

Disadvantages

- Programming languages are not designed for describing models
  - Constraint descriptions can be awkward
  - Special methods may be required for efficiency
- Modeling and programming bugs are hard to separate
Algebraic Modeling Languages

Declarative

Concept
- Design a language for describing optimization models
- Connect to external applications via . . .
  * extensions for scripting and data transfer
  * APIs for programming languages

Disadvantages
- Adds a system between application and solver

Advantages
- Designed for building and using optimization models
  * Streamlines model building and processing
  * Promotes validation and maintenance of models
- Not specific to one programming language or solver
Algebraic Modeling Languages

Integration with Applications

gurobipy

- Everything can be developed in Python
- Part of the Gurobi package
  * Free solver-independent alternatives (Pyomo, PuLP, Python-MIP)

AMPL

- Prototypes can be developed in AMPL
  * Modeling language extended with loops, tests, assignments
- Application programming interfaces (APIs)
  for integrating AMPL with popular programming languages
  * C++, C#, Java, MATLAB, Python, R
Integration with Solvers

**gurobipy**
- Works closely with the Gurobi solver: callbacks during optimization, fast re-solves after problem changes
- Supports Gurobi’s extended expressions: min/max, and/or, if-then-else

**AMPL**
- Supports all popular solvers
- Extends to general nonlinear and logic expressions
  * Connects to nonlinear function libraries and user-defined functions
  * Automatically computes nonlinear function derivatives
  * Connects to global optimization and constraint programming solvers
**Executable**

**Advantages**
- Complete application development in one environment
- Direct access to advanced solver features

**Disadvantages**
- Programming languages are not designed for describing models
  - Constraint descriptions can be awkward
  - Model and data are mixed
  - Special methods may be required for efficiency
- Modeling and programming bugs are hard to separate

*Algebraic Modeling Languages*
Algebraic Modeling Languages

Declarative

Disadvantages

- Adds a system between application and solver

Advantages

- Focused on optimization modeling
  - Streamlined application prototyping, without programming
  - Faster processing, stronger validation, easier maintenance
- Not specific to one programming language
  - Scripting language extends the model statements with loops, tests, and assignments
  - APIs provide multiple programming language interfaces tailored to C++, C#, Java, MATLAB, Python, R
Balanced Assignment Revisited

**Given**

- \( P \) set of people
- \( C \) set of categories of people
- \( t_{ik} \) type of person \( i \) within category \( k \), for all \( i \in P, k \in C \)

**and**

- \( G \) number of groups
- \( g_{\text{min}} \) lower limit on people in a group
- \( g_{\text{max}} \) upper limit on people in a group

**Define**

- \( T_k = \bigcup_{i \in P} \{t_{ik}\} \), for all \( k \in C \)
  - set of all types of people in category \( k \)
Balanced Assignment Revisited *in AMPL*

*Sets, parameters*

```AMPL
set PEOPLE;   # individuals to be assigned
set CATEG;
param type {PEOPLE,CATEG} symbolic;
    # categories by which people are classified;
    # type of each person in each category

param numberGrps integer > 0;
param minInGrp integer > 0;
param maxInGrp integer >= minInGrp;
    # number of groups; bounds on size of groups

set TYPES {k in CATEG} = setof {i in PEOPLE} type[i,k];
    # all types found in each category
```
Balanced Assignment

Determine

\[ x_{ij} \in \{0,1\} = 1 \text{ if person } i \text{ is assigned to group } j \]
\[ = 0 \text{ otherwise, for all } i \in P, j = 1, \ldots, G \]

\[ y_{kl}^{\min} \] fewest people of category \( k \), type \( l \) in any group,

\[ y_{kl}^{\max} \] most people of category \( k \), type \( l \) in any group,

for each \( k \in C, l \in T_k \)

Where

\[ y_{kl}^{\min} \leq \sum_{i \in P: t_{ik} = l} x_{ij}, \text{ for each } j = 1, \ldots, G; k \in C, l \in T_k \]

\[ y_{kl}^{\max} \geq \sum_{i \in P: t_{ik} = l} x_{ij}, \text{ for each } j = 1, \ldots, G; k \in C, l \in T_k \]
Balanced Assignment in AMPL

Variables, defining constraints

```AMPL
var Assign {i in PEOPLE, j in 1..numberGrps} binary;
    # Assign[i,j] is 1 if and only if
    # person i is assigned to group j
var MinType {k in CATEG, TYPES[k]};
var MaxType {k in CATEG, TYPES[k]};
    # fewest and most people of each type, over all groups
subj to MinTypeDefn {j in 1..numberGrps, k in CATEG, l in TYPES[k]}:
    MinType[k,l] <= sum {i in PEOPLE: type[i,k] = l} Assign[i,j];
subj to MaxTypeDefn {j in 1..numberGrps, k in CATEG, l in TYPES[k]}:
    MaxType[k,l] >= sum {i in PEOPLE: type[i,k] = l} Assign[i,j];
    # values of MinTypeDefn and MaxTypeDefn variables
    # must be consistent with values of Assign variables
```

\[ y_{kl}^{\text{max}} \geq \sum_{i \in P : t_{ik} = l} x_{ij}, \text{ for each } j = 1, \ldots, G; \ k \in C, l \in T_k \]
Balanced Assignment

Minimize

$$\sum_{k \in C} \sum_{l \in T_k} (y_{kl}^{\max} - y_{kl}^{\min})$$

sum of inter-group variation over all types in all categories

Subject to

$$\sum_{j=1}^{G} x_{ij} = 1, \text{ for each } i \in P$$

each person must be assigned to one group

$$g_{\min} \leq \sum_{i \in P} x_{ij} \leq g_{\max}, \text{ for each } j = 1, \ldots, G$$

each group must be assigned an acceptable number of people
Balanced Assignment in AMPL

Objective, assignment constraints

minimize TotalVariation:
   sum {k in CATEG, l in TYPES[k]} (MaxType[k,l] - MinType[k,l]);
   # Total variation over all types

subj to AssignAll {i in PEOPLE}:
   sum {j in 1..numberGrps} Assign[i,j] = 1;
   # Each person must be assigned to one group

subj to GroupSize {j in 1..numberGrps}:
   minInGrp <= sum {i in PEOPLE} Assign[i,j] <= maxInGrp;
   # Each group must have an acceptable size

\[ g^{\text{min}} \leq \sum_{i \in P} x_{ij} \leq g^{\text{max}}, \text{ for each } j = 1, \ldots, G \]
Balanced Assignment

Define also

\[
Q = \{i \in P : t_{i,m/f} = \text{female}\}
\]

\[
LR_{lr} = \{i \in P : t_{i,loc} = l, t_{i,rank} = r\}, \text{ for all } l \in T_{loc}, r \in T_{rank}
\]

\[
A_r \subseteq T_{rank}, \text{ for all } r \in T_{rank}
\]

Subject to also

\[
\sum_{i \in Q} x_{ij} = 0 \text{ or } \sum_{i \in Q} x_{ij} \geq 2, \text{ for each } j = 1, \ldots, G
\]

no group may have only one woman assigned

\[
\sum_{i \in LR_{lr}} x_{ij} = 0 \text{ or } \sum_{i \in LR_{lr}} x_{ij} + \sum_{a \in A_r} \sum_{i \in LR_{la}} x_{ij} \geq 2,
\]

for each \(l \in T_{loc}, r \in T_{rank}, j = 1, \ldots, G\)

for each person in each location, there must be

at least one other person of the same or an adjacent rank
Balanced Assignment in AMPL

Complicating constraints

```
set WOMEN = {i in PEOPLE: type[i,'m-f'] = 'F'};
subj to Min2WomenInGroupLO {j in 1..numberGrps}:
    sum{i in WOMEN} Assign[i,j] = 0 or sum{i in WOMEN} Assign[i,j] >= 2;

\sum_{i \in Q} x_{ij} = 0 \text{ or } \sum_{i \in Q} x_{ij} \geq 2, \text{ for each } j = 1,...,G
```

```
set LOCRANK {l in TYPES['loc'], r in TYPES['rank']} =
    {i in PEOPLE: type[i,'loc'] = l and type[i,'rank'] = r};
set ADJACENT {r in TYPES['rank']} within TYPES['rank'] diff {r};
subj 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;
```
**Balanced Assignment**

**Modeling Language Data**

210 people, 4 categories

- 18 types, 12 groups, 16-19 people/group

<table>
<thead>
<tr>
<th>A</th>
<th>B</th>
<th>C</th>
<th>D</th>
<th>E</th>
<th>F</th>
<th>G</th>
<th>H</th>
<th>I</th>
<th>J</th>
<th>K</th>
<th>L</th>
<th>M</th>
<th>N</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td></td>
<td></td>
<td>B</td>
<td>P</td>
<td>d</td>
<td>m</td>
<td>r</td>
<td>nGrps</td>
<td>mGrp</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>CATEG</td>
<td>PEOPLE</td>
<td>dept</td>
<td>loc</td>
<td>mf</td>
<td>rank</td>
<td>numberGrps</td>
<td>minGrp</td>
<td>maxGrp</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>dept</td>
<td>BIW</td>
<td>NNE</td>
<td>Peoria</td>
<td>M</td>
<td>Assistant</td>
<td>12</td>
<td>16</td>
<td>19</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>loc</td>
<td>KRS</td>
<td>WSW</td>
<td>Springfield</td>
<td>F</td>
<td>Assistant</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>mf</td>
<td>TLR</td>
<td>NNW</td>
<td>Peoria</td>
<td>F</td>
<td>Adjunct</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>rank</td>
<td>VAA</td>
<td>NNW</td>
<td>Peoria</td>
<td>M</td>
<td>Deputy</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>7</td>
<td></td>
<td>JRT</td>
<td>NNE</td>
<td>Springfield</td>
<td>M</td>
<td>Deputy</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>8</td>
<td></td>
<td>AMR</td>
<td>SSE</td>
<td>Peoria</td>
<td>M</td>
<td>Deputy</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>9</td>
<td></td>
<td>MES</td>
<td>NNE</td>
<td>Peoria</td>
<td>M</td>
<td>Consultant</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>10</td>
<td></td>
<td>JAD</td>
<td>NNE</td>
<td>Peoria</td>
<td>M</td>
<td>Adjunct</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>11</td>
<td></td>
<td>MJR</td>
<td>NNE</td>
<td>Springfield</td>
<td>M</td>
<td>Assistant</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>12</td>
<td></td>
<td>JRS</td>
<td>NNE</td>
<td>Springfield</td>
<td>M</td>
<td>Assistant</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>13</td>
<td></td>
<td>HCN</td>
<td>SSE</td>
<td>Peoria</td>
<td>M</td>
<td>Deputy</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>14</td>
<td></td>
<td>DAN</td>
<td>NNE</td>
<td>Springfield</td>
<td>M</td>
<td>Adjunct</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>15</td>
<td></td>
<td>CWT</td>
<td>NNE</td>
<td>Springfield</td>
<td>M</td>
<td>Adjunct</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>16</td>
<td></td>
<td>DCN</td>
<td>NNE</td>
<td>Peoria</td>
<td>M</td>
<td>Adjunct</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Adding Optimization to Your Applications, Quickly and Reliably

INFORMS Business Analytics Conference — 3 April 2022
**Balanced Assignment**

**Modeling Language Script**

*Read model & data, solve, write solution*

```apl
model BalAssign2022.mod;

table Categories IN "amplxl" "bal.xlsx": CATEG <- [CATEG];
table People IN "amplxl" "bal.xlsx": PEOPLE <- [PEOPLE];
table Types IN "amplxl" "bal.xlsx" "2D": [PEOPLE,CATEG], type;
table Groups IN "amplxl" "bal.xlsx": [], numberGrps, minInGrp, maxInGrp;
table Adjacent {r in TYPES['rank']} IN "amplxl" "bal.xlsx": ADJACENT[r] <- [(r)];

read table Categories; read table People;
read table Types; read table Groups; read table Adjacent;

option solver x-gurobi;
solve;

table Summary {k in CATEG} OUT "amplxl" "bal.xlsx" (k) "2D":
    {j in 1..numberGrps, l in TYPES[k]} -> [Group,Type],
    sum {i in PEOPLE: type[i,k] = l} Assign[i,j];

write table Summary;
```
Balanced Assignment

Modeling Language Execution

Load spreadsheet handler, execute script

```
ampl: load amplxl.dll;

ampl: include BalAssign2022.run;
```

Presolve eliminates 72 arithmetic and 144 logical constraints.

Adjusted problem:
2556 variables:
  2520 nonlinear variables
  36 linear variables

582 algebraic constraints, all linear; 25224 nonzeros
  210 equality constraints
  360 inequality constraints
  12 range constraints

252 logical constraints
1 linear objective; 2 nonzeros.

x-Gurobi 9.5.1: optimal solution; objective 25
134242 simplex iterations
816 branching nodes

50.4 sec
### Balanced Assignment

**Modeling Language Results**

**Rank**

<table>
<thead>
<tr>
<th></th>
<th>A</th>
<th>B</th>
<th>C</th>
<th>D</th>
<th>E</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Group</td>
<td>Assistant</td>
<td>Adjunct</td>
<td>Deputy</td>
<td>Consultant</td>
</tr>
<tr>
<td>2</td>
<td>1</td>
<td>8</td>
<td>7</td>
<td>2</td>
<td>0</td>
</tr>
<tr>
<td>3</td>
<td>2</td>
<td>8</td>
<td>7</td>
<td>2</td>
<td>1</td>
</tr>
<tr>
<td>4</td>
<td>3</td>
<td>8</td>
<td>8</td>
<td>2</td>
<td>1</td>
</tr>
<tr>
<td>5</td>
<td>4</td>
<td>7</td>
<td>7</td>
<td>2</td>
<td>1</td>
</tr>
<tr>
<td>6</td>
<td>5</td>
<td>8</td>
<td>8</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>7</td>
<td>6</td>
<td>7</td>
<td>7</td>
<td>2</td>
<td>1</td>
</tr>
<tr>
<td>8</td>
<td>7</td>
<td>8</td>
<td>8</td>
<td>2</td>
<td>0</td>
</tr>
<tr>
<td>9</td>
<td>8</td>
<td>7</td>
<td>8</td>
<td>2</td>
<td>1</td>
</tr>
<tr>
<td>10</td>
<td>9</td>
<td>7</td>
<td>8</td>
<td>2</td>
<td>0</td>
</tr>
<tr>
<td>11</td>
<td>10</td>
<td>8</td>
<td>8</td>
<td>2</td>
<td>0</td>
</tr>
<tr>
<td>12</td>
<td>11</td>
<td>8</td>
<td>7</td>
<td>2</td>
<td>1</td>
</tr>
<tr>
<td>13</td>
<td>12</td>
<td>8</td>
<td>7</td>
<td>2</td>
<td>0</td>
</tr>
</tbody>
</table>

---

*Accessibility: Good to go*
Balanced Assignment

Modeling Language Results

Location

![Excel Spreadsheet showing Balanced Assignment results with columns for Group, Peoria, Springfield, Macomb, Urbana, Joliet, Carbondale, Cairo, and Evansville]
Solvers for Model-Based Optimization

Ready-to-run solvers for broad problem classes

Three widely used types

- “Linear”
- “Nonlinear”
- “Global”
“Linear” Solvers

Require objective and constraint coefficients

Linear objective and constraints

- Continuous variables
  - Primal simplex, dual simplex, interior-point
- Integer (including zero-one) variables
  - Branch-and-bound + feasibility heuristics + cut generation
  - Automatic transformations to linear:
    piecewise-linear expressions, logic in constraints, . . .

Quadratic extensions

- Convex elliptic objectives and constraints
- Convex conic constraints
- $x_j u_j$ terms, where $u_j$ is a zero-one variable
- General non-convex quadratic expressions
“Nonlinear” Solvers

Require function and derivative evaluations

Continuous variables, local optimality

- Smooth objective and constraint functions
  - Derivative computations handled by modeling language systems
- Variety of methods
  - Interior-point, sequential quadratic, reduced gradient

Some extend to integer variables
“Global” Solvers

*Require expression graphs (or equivalent)*

*Nonlinear expressions, global optimality*

- Substantially harder than local optimality
- Smooth nonlinear objective and constraint functions
- Continuous and integer variables
**Try AMPL!**

**New! Free AMPL Community Edition** ampl.com/ce
- Free AMPL and open-source solvers
  - no size or time limitations
- 1-month full-featured trials of commercial solvers
- Requires internet connection for validation

**Time-limited trials, size-limited demos** ampl.com/try-ampl

**Free AMPL for Courses** ampl.com/try-ampl/ampl-for-courses
- Full-featured, time limited

**Free AMPL web access**
- AMPL model colaboratory colab.ampl.com
- NEOS Server neos-server.org