Modeling with the Gurobi-Python Interface

Juan Antonio Orozco Guzman

GUROBI OPTIMIZATION

The World's Fastest Solver

About the Speaker

Juan Antonio Orozco Guzman

Optimization Support Engineer

Gurobi Optimization

Modeling with the Gurobi-Python Interface

Introduction

GUROBI OPTIMIZATION

The World's Fastest Solver

Objectives

- Discuss the motivation for using Python in mathematical optimization (MO) applications
- Help you understand the importance of parameterizing a MO model
- Review some of the best practices for deploying MO models in Python

Agenda

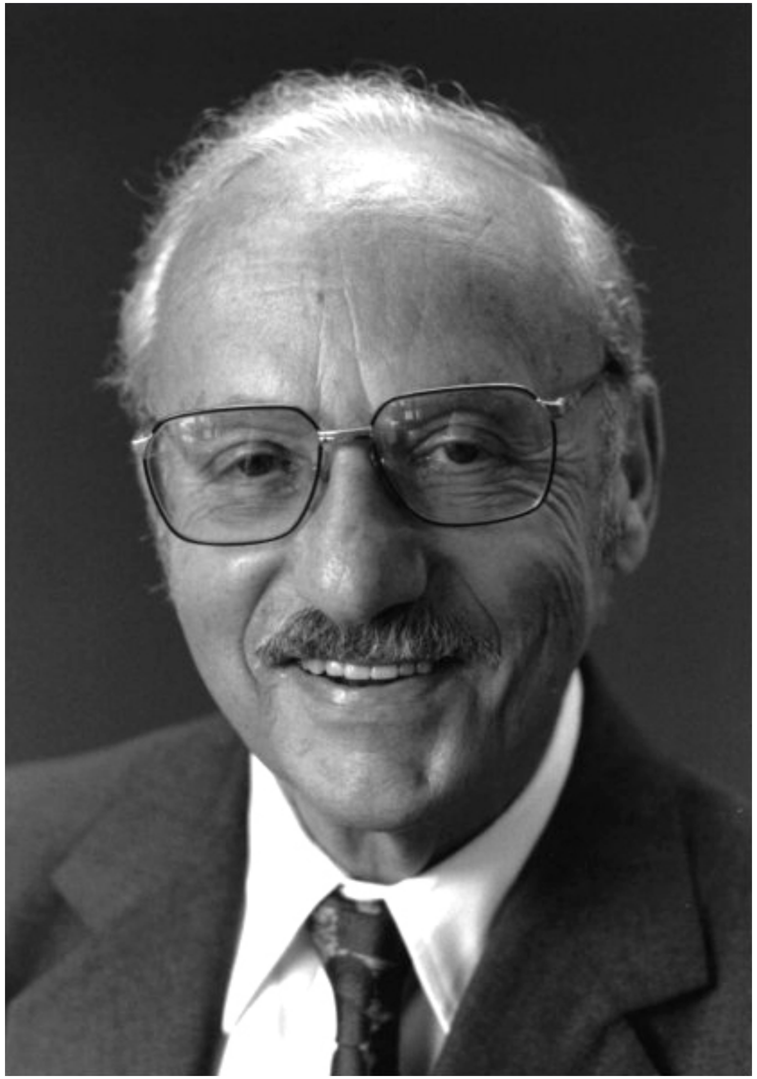
- Introduction
- Elements of a MO model
- Motivation for using Python
- Gurobi-Python interface
- Sparsity and best practices
- Conclusion

Modeling with the Gurobi-Python Interface

Introduction

What is Mathematical Optimization?

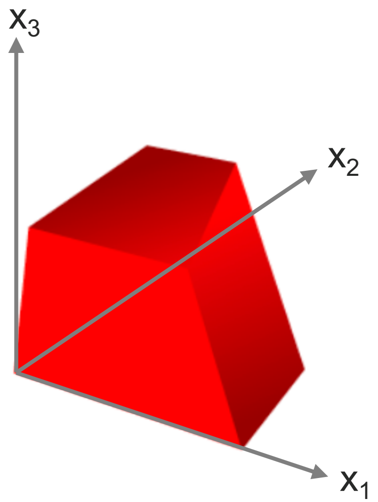
Its origins can be traced back to the invention of **linear programming** shortly after World War II in 1947 by **George Dantzig**.



What is Mathematical Optimization?

GUROBI OPTIMIZATION

- **A declarative approach where the modeler:**
  - States all the **properties** associated with a solution.
  - Defines a **criterion** to guide the search.
- **Other algorithmic approaches specify the necessary steps to build and test a solution.**



Mathematical Optimization Modeling

Elements of a MO model

Example: Assignment Problem

GUROBI OPTIMIZATION

Maximize total matching score:

$$\max \sum_{i \in I} \sum_{j \in J} c_{i,j} x_{i,j}$$

At most, one person can be assigned to each job

$$s.t. \quad \sum_{i \in I} x_{i,j} \leq 1 \quad \forall j \in J$$

At most, one job can be assigned to each person

$$\sum_{j \in J} x_{i,j} \leq 1 \quad \forall i \in I$$

1 if person i is assigned to job j; 0 otherwise

$$x_{i,j} \in \{0,1\} \quad \forall i \in I, j \in J$$

Elements of an Assignment Problem

$$
\begin{aligned}
\max \quad & \sum_{i \in I} \sum_{j \in J} c_{i,j} \boxed{x_{i,j}} & & \\
s.t. \quad & \sum_{i \in I} \boxed{x_{i,j}} \leq 1 & & \forall j \in J \\
& \sum_{j \in J} \boxed{x_{i,j}} \leq 1 & & \forall i \in I \\
& \boxed{x_{i,j}} \in \{0,1\} & & \forall i \in I, j \in J
\end{aligned}
$$

Decision Variables

Elements of an Assignment Problem

$$\boxed{\max \quad \sum_{i \in I} \sum_{j \in J} c_{i,j} x_{i,j}}$$

Objective Function

$$
\begin{aligned}
s.t. \quad & \sum_{i \in I} x_{i,j} \quad \leq \quad 1 \quad \forall j \in J \\
& \sum_{j \in J} x_{i,j} \quad \leq \quad 1 \quad \forall i \in I \\
& x_{i,j} \in \{0,1\} \qquad \forall i \in I, j \in J
\end{aligned}
$$

Elements of an Assignment Problem

$$
\begin{aligned}
\max \quad & \sum_{i \in I} \sum_{j \in J} c_{i,j} x_{i,j} \\
s.t. \quad & \boxed{\sum_{i \in I} x_{i,j} \leq 1 \quad \forall j \in J} \\
& \boxed{\sum_{j \in J} x_{i,j} \leq 1 \quad \forall i \in I} \\
& \boxed{x_{i,j} \in \{0,1\} \quad \forall i \in I, j \in J}
\end{aligned}
$$

Constraints

Elements of an Assignment Problem

$$
\begin{aligned}
\max \quad & \sum_{i \in I} \sum_{j \in J} \boxed{c_{i,j}} x_{i,j} \\
s.t. \quad & \sum_{i \in I} x_{i,j} \leq \boxed{1} \quad \forall j \in J \\
& \sum_{j \in J} x_{i,j} \leq \boxed{1} \quad \forall i \in I \\
& x_{i,j} \in \{0,1\} \quad \forall i \in I, j \in J
\end{aligned}
$$

Data Coefficients

Elements of an Assignment Problem

$$
\begin{aligned}
\max \quad & \sum_{i \in \boxed{I}} \sum_{j \in \boxed{J}} c_{i,j} x_{i,j} \\
s.t. \quad & \sum_{i \in \boxed{I}} x_{i,j} \leq 1 \quad \forall j \in \boxed{J} \\
& \sum_{j \in \boxed{J}} x_{i,j} \leq 1 \quad \forall i \in \boxed{I} \\
& x_{i,j} \in \{0,1\} \quad \forall i \in \boxed{I}, j \in \boxed{J}
\end{aligned}
$$

Index Sets

Elements of an Assignment Problem

$$
\begin{array}{llcll}
 & \max \sum_{i \in I} \sum_{j \in J} c_{\boxed{i,j}} x_{\boxed{i,j}} & & & \\
s.t. & \sum_{i \in I} x_{\boxed{i,j}} & \leq & 1 & \forall j \in J \\
 & \sum_{j \in J} x_{\boxed{i,j}} & \leq & 1 & \forall i \in I \\
 & x_{\boxed{i,j}} \in \{0,1\} & & & \forall i \in I, j \in J
\end{array}
$$

Subscripts

Elements of an Assignment Problem

$$
\begin{aligned}
\max \quad & \sum_{i \in I} \sum_{j \in J} c_{i,j} x_{i,j} \\
s.t. \quad & \sum_{i \in I} x_{i,j} \leq 1 \quad \forall j \in J \\
& \sum_{j \in J} x_{i,j} \leq 1 \quad \forall i \in I \\
& x_{i,j} \in \{0,1\} \quad \forall i \in I, j \in J
\end{aligned}
$$

Arithmetic Operators

Elements of an Assignment Problem

$$
\begin{aligned}
\max \quad & \sum_{i \in I} \sum_{j \in J} c_{i,j} x_{i,j} \\
s.t. \quad & \sum_{i \in I} x_{i,j} \ \boxed{\leq} \ 1 \quad \forall j \in J \\
& \sum_{j \in J} x_{i,j} \ \boxed{\leq} \ 1 \quad \forall i \in I \\
& x_{i,j} \in \{0,1\} \quad \forall i \in I, j \in J
\end{aligned}
$$

Constraint Operators

Elements of an Assignment Problem

$$
\begin{aligned}
\max \quad & \sum_{i \in I} \sum_{j \in J} c_{i,j} x_{i,j} \\
s.t. \quad & \sum_{i \in I} x_{i,j} \leq 1 \quad \boxed{\forall} j \in J \\
& \sum_{j \in J} x_{i,j} \leq 1 \quad \boxed{\forall} i \in I \\
& x_{i,j} \in \{0,1\} \quad \boxed{\forall} i \in I, j \in J
\end{aligned}
$$

For-all Operators

Elements of an Assignment Problem

$$
\begin{aligned}
\max \quad & \boxed{\sum_{i \in I} \sum_{j \in J}} c_{i,j} x_{i,j} \\
s.t. \quad & \boxed{\sum_{i \in I}} x_{i,j} \leq 1 \quad \forall j \in J \\
& \boxed{\sum_{j \in J}} x_{i,j} \leq 1 \quad \forall i \in I \\
& x_{i,j} \in \{0,1\} \quad \forall i \in I, j \in J
\end{aligned}
$$

Aggregate-sum Operators

Elements of a MO Model

- Decision variables
- Objective function
- Constraints
- Coefficients
- Sets and subscripts
- Operators:
  - Arithmetic (+, -, *, /)
  - Constraint ($\leq, \geq, =$)
  - For-all ($\forall$)
  - Aggregate-sum (Σ)

Mathematical Optimization Modeling

Motivation for Using Python

Desired Modeling Traits

Translate a math formulation into a machine-readable format:

- Code is **easy to write and maintain**
- Covers all solver and programming needs
- **Low overhead** as compared with solver alone

Why Python?

GUROBI OPTIMIZATION

- **Fast** and **powerful**; **Easy** to learn, use, and read
- **Extendible** with vast library of prewritten modules
  - Statistics
  - GUIs
  - Web development and web services
  - Data connections
  - File compression
  - Data encryption
- **In top-3** most popular programming languages (TIOBE index)
- Large community
- **Free** and **open-source** with license that is business-friendly

python™

Jupyter Notebook

- Web application to create and share documents that contain:
  - Live code
  - Explanatory text
  - Equations
  - Visualizations
- Excellent for **reproducible research**
- Free software with an active community
- Learn more at http://jupyter.org

Mathematical Optimization Modeling

Gurobi-Python Interface

Gurobi Modeling Objects

- `Model`: an optimization model
- `Var`: a decision variable
- `Constr`: a constraint
- Overloaded operators
  - Arithmetic
  - Constraint
- Aggregate sum operator (`quicksum`)
- `Tuplelists` and `multidicts`
- Python provides the rest:
  - Read data
  - Print out / write solutions
  - Define control and looping statements

Simple Python Example

```python
import gurobipy as gp
from gurobipy import GRB

model = gp.Model("mip1")

x = model.addVar(vtype=GRB.BINARY, name="x")
y = model.addVar(vtype=GRB.BINARY, name="y")
z = model.addVar(vtype=GRB.BINARY, name="z")

model.setObjective(x + y + 2*z, GRB.MAXIMIZE)

model.addConstr(x + 2*y + 3*z <= 4, name="Constraint1")
model.addConstr(x + y >= 1, name="Constraint2")

model.optimize()
```

$$
\begin{aligned}
Max \quad & x + y + 2z \\
s.t. \quad & x + 2y + 3z \leq 4 \\
& x + y \geq 1 \\
& x, y, z \in \{0,1\}
\end{aligned}
$$

Indexing and Subscripts in Python

- Python provides data structures that are well-suited for indexing and subscripts:
  - List
  - Dictionary
  - Tuple
- Gurobi also provides:
  - Tuplelist
  - Multidict

Python Tuple

- A fixed, compound grouping
  - `arc = ("CHI", "NYC")`
- A tuple cannot be modified once it is created
- Ideal for representing multi-dimensional subscripts
  - `("CHI", "NYC", "Prod1")`
  - `("CHI", "NYC", "Prod2")`
  - `("CHI", "NYC", "ProdN")`

Python List

- An ordered group
  - `cities = ["CHI", "NYC", "ATL", "MIA"]`
- Lists can be modified
  - Add, delete, sort elements
- Unlike sets, a list can have repeated elements
  - Python has a set type to represent sets

Python Dictionary

- A mapping from keys to values
  - `Cost[("CHI", "NYC")] = 100`
  - `Cost[("ATL", "NYC")] = 110`
- Keys can be basic values or tuples
- Ideal for representing indexed data

Gurobi Tuplelist

- Gurobi´s extension of lists
- Efficient for storing a list of tuples
  - `arcs = gp.tuplelist([("CHI", "NYC"), ("CHI", "ATL"), ("ATL", "MIA"), ("ATL", "NYC")])`
- `Select()` method finds matching subsets
  - `print(arcs.select("CHI", "*"))`

    `[("CHI", "NYC"), ("CHI", "ATL")]`
- `Select()` uses efficient indexing

Gurobi Multidict

- Convenience function to initialize dictionaries and their indices in one statement
  - ```
    cities, supply, demand = gp.multidict({
      "ATL": [100, 20],
      "CHI": [150, 50],
      "NYC": [20, 300],
      "MIA": [10, 200]})
    ```
- Index is returned as a tuplelist
    ```

The Diet Model

GUROBI OPTIMIZATION

$$\text{Min} \sum_{f \in Foods} cost_f \cdot buy_f$$

$$s.t. \quad \sum_{f \in Foods} nutrition\_value_{c,f} \cdot buy_f \geq \min\_nutrition_c \quad \forall c \in Categories$$

$$\sum_{f \in Foods} nutrition\_value_{c,f} \cdot buy_f \leq \max\_nutrition_c \quad \forall c \in Categories$$

$$buy_f \in \mathbb{R}^+$$

jupyter

Jupyter Notebook

Mathematical Optimization Modeling

Sparsity and Best Practices

Sparsity, Part 1

Mistake

```
availability = tuplelist([
  ("Amy", "Tue2"), ("Amy", "Wed3"),
  ("Amy", "Fri5"), ("Amy", "Sun7"),
...])

x = {}
for w in workers:
  for s in shifts:
    x[w,s] = model.addVar(ub=1,
obj=pay[w])
    if (w,s) not in availability:
      x[w,s].ub = 0
```

Overhead for both modeling and solving

Done Right

```
availability = tuplelist([
  ("Amy", "Tue2"), ("Amy", "Wed3"),
  ("Amy", "Fri5"), ("Amy", "Sun7"),
...])

x = {}
for w,s in availability:
  x[w,s] = model.addVar(ub=1,
obj=pay[w])
```

Variables represent only valid combinations of workers and shifts

Sparsity, Part 2

Mistake

```
# Constraint: assign exactly
shiftRequirements[s] workers to
shift s


model.addConstrs((shiftRequirements
[s] == gp.quicksum(x[w,s] for w in
workers if (w,s) in availability)
for s in shifts),
name="requirement")
```

Modeling step iterates over all workers, rather than only valid combinations

Done Right

```
# Constraint: assign exactly
shiftRequirements[s] workers to
shift s


model.addConstrs((shiftRequirements
[s] == gp.quicksum(x[w,s] for w,s
in availability.select("*",s)) for
s in shifts), name="requirement")
```

Iterates over only valid combinations of workers and shifts

Exploiting Sparsity

- Use tuplelists to specify valid combinations
- Create variables and data as dictionaries that are indexed by these tuplelists
- Use `tuplelist.select()` to efficiently iterate over valid combinations

Best Practices for Python Models

- Sparsity
- List comprehension and generator syntax
- `gp.quicksum()`
- Model / data separation and external sources of data
  - dietmodel.py (module with logic for model deployment)
  - diet2.py (data hard-coded in a separate script)
  - diet3.py (data read from a SQLite database)

Mathematical Optimization Modeling

Conclusion

Modeling Languages vs Gurobi-Python Interface

- **Concise, readable syntax for building models**
- **Interative model and data manipulation**
- **Efficient handling of dense and sparse data**
- **Convenient connections to external data sources**

Modeling Languages

- Specialized language for math programming
  - Syntax and primitives designed for optimization
  - Best practices built into the language
  - Error handling tailored for optimization

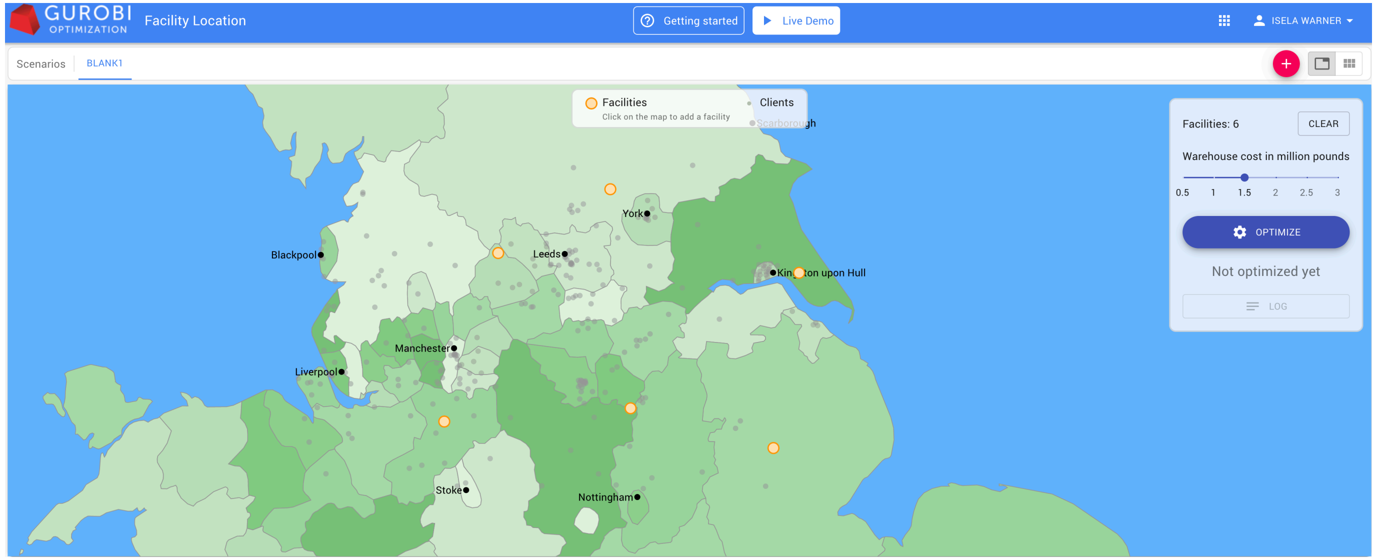
Gurobi-Python Interface

- Part of a general-purpose language
  - Greater flexibility
  - Designed with deployment in mind
  - Build complex programs in one language
  - Integrate easily with databases, web servers, etc.
  - Covers all features in Gurobi Optimizer
  - Large user community

Resources

- **Optimization Application Demos**
  - Cell Tower Coverage
  - Cutting Stock Problem with Multiple Master Rolls
  - Facility Location
  - Offshore Wind Farming
  - Resource Matching Optimization
  - The Traveling Salesman Problem
  - Workforce Scheduling

- **Jupyter Notebook Modeling Examples**
  - Cell Tower Coverage
  - Customer Assignment
  - Facility Location
  - HP Williams Modeling Examples
  - L0-Regression
  - Offshore Wind Farming
  - Standard Pooling
  - Traveling Salesman



Resources

- **Get a free 30-day trial of the Gurobi Optimizer**
  - www.gurobi.com/eval

- **Need help leveraging Optimization for your business?**
  - Contact us at info@gurobi.com

Thank You

GUROBI
OPTIMIZATION

The World's Fastest Solver