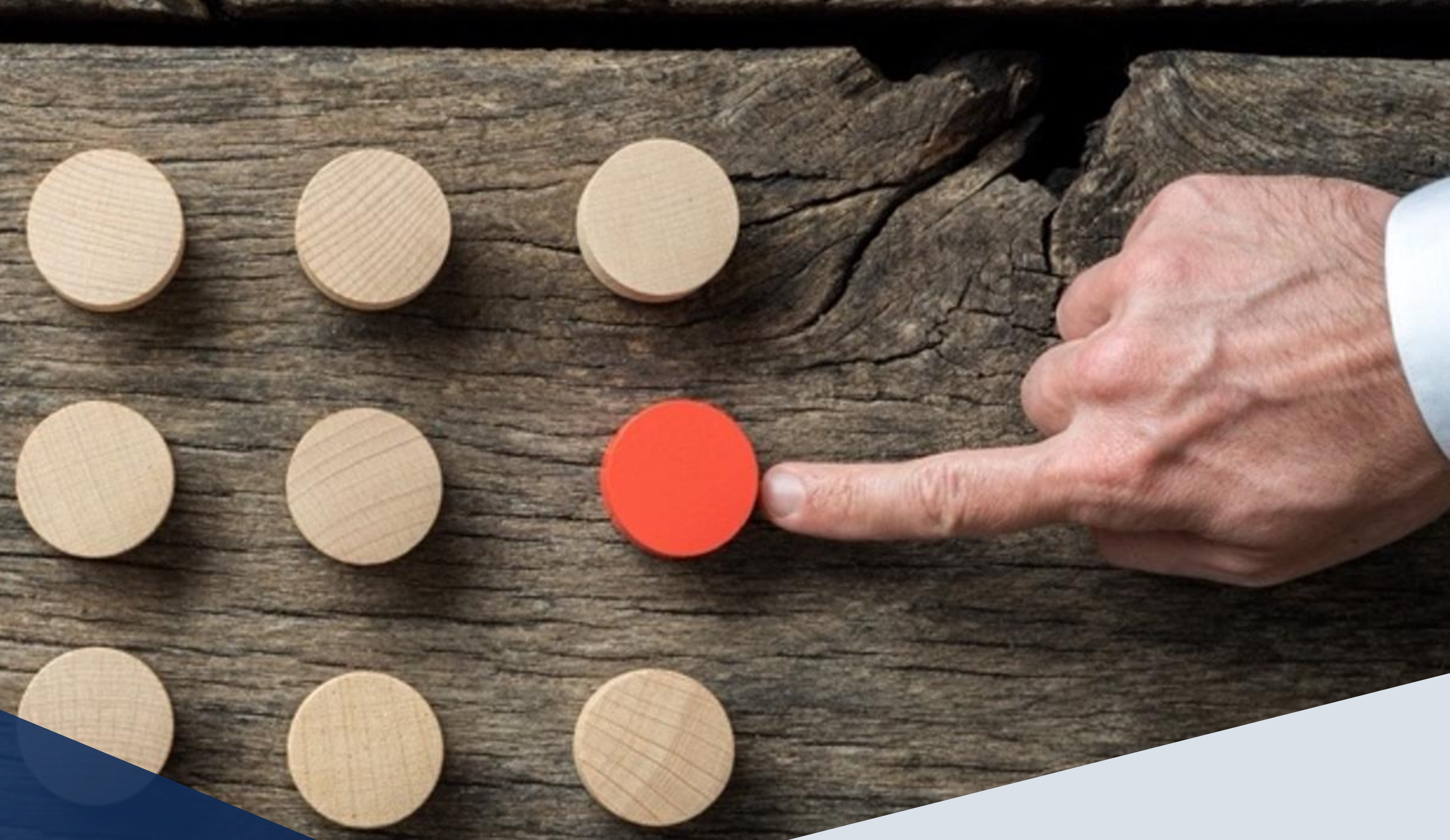


Gurobi Connect Chicago

Advanced Modeling

Rodrigo Fuentes
Sr. Technical Account Manager

GUROBI
OPTIMIZATION

Agenda

Gurobi Python Extensions

Advanced Modeling Features

- Range constraints
- Non-linear functions
- Special Ordered Sets
- General constraints
- Logical conditions
- Semi-continuous variables
- Selecting Big-M values

Multiple Objectives

GUROBI OPTIMIZATION

Gurobi Python Extensions

Python data structures and Gurobi extensions

- Python provides data structures that are well-suited for deploying optimization models:
  - Tuples
  - Lists
  - Sets
  - Dictionaries

- On top of that, gurobipy also offers a few data structures that allow you to build subsets from a collection of tuples efficiently:
  - Tuplelists
  - Tupledicts
  - Multidicts

Already available structures in python

GUROBI OPTIMIZATION

Tuple

A tuple is an ordered, compound grouping that cannot be modified once it is created. It is ideal for representing multi-dimensional subscripts.

```
mytuple = ('truck22','truck37') # create
print(mytuple[0]) # access
```

List

A list is an ordered group, so each item is indexed. Lists can be modified by adding, deleting or sorting elements.

```
mylist = ['truck22', 'truck37', 'truck53'] # create
print(mylist[1]) # access
```

Set

A set is an unordered group. As such, sets can only be modified by adding or deleting elements. Unlike lists, sets cannot have repeated elements.

```
myset = {'truck22', 'truck37', 'truck53'}
for item in myset:
   print(item)
```

Dictionary

A dictionary is a mapping from keys to values that is ideal for representing indexed data, e.g. cost, demand, capacity, etc. Typically strings, numbers, and tuples are used as keys, which should be unique.

```
demand = {
   'demand4': 20,
   'demand22': 40,
}
print(demand['demand22’])
```


Gurobipy Data Structures

GUROBI OPTIMIZATION

Tuplelist

This is Gurobi's extension of a Python list to efficiently build sub-lists from a list of tuples.

```
mytuplelist = gp.tuplelist([
   ("demand4", "truck22"), ("demand4", "truck37"), ("demand4", "truck53"),
   ("demand22", "truck22"),])
print(mytuplelist[2]) # = ('demand4', 'truck53') (tuple)
print(mytuplelist.select("demand4","*")) # Prints all tuples with 'demand4'
```

Tupledict

This is Gurobi's extension of a Python dictionary for creating subsets of Gurobi variable objects. The keys of a tupledict are stored as a tuplelist, which enables the efficient creation of math expressions that contain a subset of matching variables by using methods like tupledict.select(), tupledict.sum() or tupledict.prod().

```
assign = model.addVars(expertise, name="assign") # returns a tupledict
supply = model.addConstrs((assign.sum(r, "*") == 1 for r in resources))
```


Gurobipy Data Structures

Multidict

As its name implies, multidict is a convenience function to define multiple dictionaries (each one with the same set of keys) and their indices in one statement. Specifically, the input consists of a Python dictionary with lists of the same length as the values associated with the keys. <u>The first output of this function is a tuplelist, and the rest are tupledicts.</u>

```
building_truck_spot_pairs, distance, scaled_demand = gp.multidict({
    ('demand4', 'truck11'): [112.7, 20],
    ('demand4', 'truck10'): [149.3, 15],
    ('demand22', 'truck11'): [155.7, 10]
})
print(scaled_demand.sum('*','truck11')) # = 30
```

```
building_truck_spot_pairs = [('demand4', 'truck11’), ('demand4', 'truck10’), ('demand22', 'truck11’)]          # tuplelist of keys to multidict

distance =  {('demand4', 'truck11'): 112.7,            #tupledict
             ('demand4', 'truck10'): 149.3,
             ('demand22', 'truck11'): 155.7 }

scaled_demand = {('demand4', 'truck11’): 20,
                 ('demand4', 'truck10'): 15,
                 ('demand22', 'truck11'): 10 }
```

1 dimensional item here, but could be tuple of arbitrary dimension

Gurobi tuplelist

GUROBI OPTIMIZATION

A Gurobi class to store lists of tuples

```
>>> import gurobipy as gp
>>> from gurobipy import GRB
Cities = ['A','B','C','D']
Routes = gp.tuplelist([('A','B'), ('A','C'), ('B','C'), ('B','D'), ('C','D')])
```

What makes it special: select() statement for efficient filtering

```
for c in Cities:                   # [('A','B'), ('A','C')]
  print(Routes.select(c,'*'))      # [('B','C'), ('B','D')]
                                   # [('C','D')]
                                   # []                 No routes with D as origin
```

The *tuplelist* is indexed to make select() efficient

Each select() call returns a tuplelist

Indexed variables: Model.addVars()

	0	1	2
0	(0,0)	(0,1)	(0,2)
1	(1, 0)	(1,1)	(1,2)

Using integers

```
x = model.addVars(2, 3, name="x")
# x[0,0], x[0,1], x[0,2], x[1,0], x[1,1], x[1,2]          (variable names)
```

Using lists of scalars

```
y = model.addVars(Cities, Cities, name="y")
# y[A,A], y[A,B], y[A,C], y[A,D], y[B,A], y[B,B], y[B,C], y[B,D]
# y[C,A], y[C,B], y[C,C], y[C,D], y[D,A], y[D,B], y[D,C], y[D,D]
```

Using a tuplelist

```
z = model.addVars(Routes, name="z") # z[A,B], z[A,C], z[B,C], z[B,D], z[C,D]
```

Using a generator expression

```
w = model.addVars((i for i in range(5) if i != 2), name="w")
# w[0], w[1], w[3], w[4]
# w  =  0: <gurobi.Var w[0]>, 1: <gurobi.Var w[1]>, 3: <gurobi.Var w[3]>, 4: <gurobi.Var w[4]>}
#  tupledict with indices as keys and variables as value
```


More about Model.addVars()

Automatically takes the cross-product of multiple indices

```
x = model.addVars(2, 3, name="x")
y = model.addVars(Cities, Cities, name="y")
```

addVars call with vs without provided “name”, generates names automatically, using the indices:

```
name="x":

{(0, 0): <gurobi.Var x[0,0]>,
 (0, 1): <gurobi.Var x[0,1]>,
 (0, 2): <gurobi.Var x[0,2]>,
 (1, 0): <gurobi.Var x[1,0]>,
 (1, 1): <gurobi.Var x[1,1]>,
 (1, 2): <gurobi.Var x[1,2]>}
```

```
no name specified:

{(0, 0): <gurobi.Var C0>,
 (0, 1): <gurobi.Var C1>,
 (0, 2): <gurobi.Var C2>,
 (1, 0): <gurobi.Var C3>,
 (1, 1): <gurobi.Var C4>,
 (1, 2): <gurobi.Var C5>}
```


Indexed constraints: Model.addConstrs()

GUROBI OPTIMIZATION

Recall

```
Routes = tuplelist([('A','B'), ('A','C'), ('B','C'), ('B','D'), ('C','D')])
```

Can use a generator expression to build constraints

```
x = model.addVars(Routes, name="x")
y = model.addVars(Routes, name="y")
model.addConstrs((x[i,j]+y[i,j] <= 2 for i,j in Routes), name="capacity")

{('A', 'B'): <gurobi.Constr capacity[A,B]>,
 ('A', 'C'): <gurobi.Constr capacity[A,C]>,
 ('B', 'C'): <gurobi.Constr capacity[B,C]>,
 ('B', 'D'): <gurobi.Constr capacity[B,D]>,
 ('C', 'D'): <gurobi.Constr capacity[C,D]>}
```

Summary: variable and constraint indices are keys in a tupledict, with the actual constraints and variables being the associated values

Operators and Python

Linear and quadratic expressions are used in constraints and the objective

- Basic (binary) mathematical operators (+ − × ÷)
- Aggregate sum operator (∑)

    Used alone as well as in products (dot product)

Aggregate sum: using tupledict.sum()

A tupledict of variables has a `sum()` function, using the same syntax as `tuplelist.select()`:

```
x = model.addVars(3, 4, vtype=GRB.BINARY, name="x")
model.addConstrs(x.sum(i,'*') <= 1 for i in range(3))
```

This generates the constraints:

```
x[0,0] + x[0,1] + x[0,2] + x[0,3] <= 1
x[1,0] + x[1,1] + x[1,2] + x[1,3] <= 1
x[2,0] + x[2,1] + x[2,2] + x[2,3] <= 1
```


Aggregate sum: using quicksum()

GUROBI OPTIMIZATION

Use generator expression inside a `quicksum()` function

```
obj = quicksum(cost[i,j] * x[i,j] for i,j in Routes)
```

Note:

`quicksum()` works just like Python's `sum()` function, but it is more efficient for optimization models.

Dot product

A tupledict of variables has a `prod()` function to compute the dot product.

If cost is a dictionary, then the following are equivalent:

```
obj = quicksum(cost[i,j] * x[i,j] for i,j in arcs)
```

```
obj = x.prod(cost)
```

Another example:

x = m.addVars([(1,2), (1,3), (2,3)])

coeff = dict([((1,2), 2.0), ((1,3), 2.1), ((2,3), 3.3)])

expr = x.prod(coeff) # LinExpr: 2.0 x[1,2] + 2.1 x[1,3] + 3.3 x[2,3]

expr = x.prod(coeff, '*', 3) # LinExpr: 2.1 x[1,3] + 3.3 x[2,3]

Takes the keys in coeff, multiplies the associated values by the values in x corresponding to the same keys.

Matrix API (since 9.0)

Support NumPy's ndarrays and scipy.sparse matrices as input

- More convenient if the underlying model is naturally expressed with matrices
- Faster because no modeling objects for individual linear expressions are created

Add matrix variables

```
x = model.addMVar(shape)
```

Add matrix constraints

```
model.addMConstr(A, x, sense, b)
```

Add regular constraints with overload operator

```
model.addConstr(A @ x <= b)
```

Set a quadratic objective function

```
model.setObjective(x @ Q @ x)
```


GUROBI OPTIMIZATION

Advanced Modeling Features

Background

Range constraints

Many models contain constraints like:

$$L \leq \sum_i a_i x_i \leq U$$

These can be rewritten as:

$$r + \sum_i a_i x_i = U$$
$$0 \leq r \leq U - L$$

The range constraint interface automates this for you

- **model.addRange(** $\sum_i a_i x_i$ **,** L**,** U**, “range0”)**

If you need to modify the range

- Retrieve the additional range variable, named `RgYourConstraintName`
- Modify the bounds on that variable

For full control, it may be easier to model this yourself.

- Useful for efficient deactivation/reactivation of constraints in the model
  - Set infinite bounds on r to deactivate

Background

General Constraints for popular logical expressions

- Absolute value
- Min/Max value
- And/Or over binary variables
- Indicator (if-then logic)

The General Constraint syntax is a safe way to implement model logic.

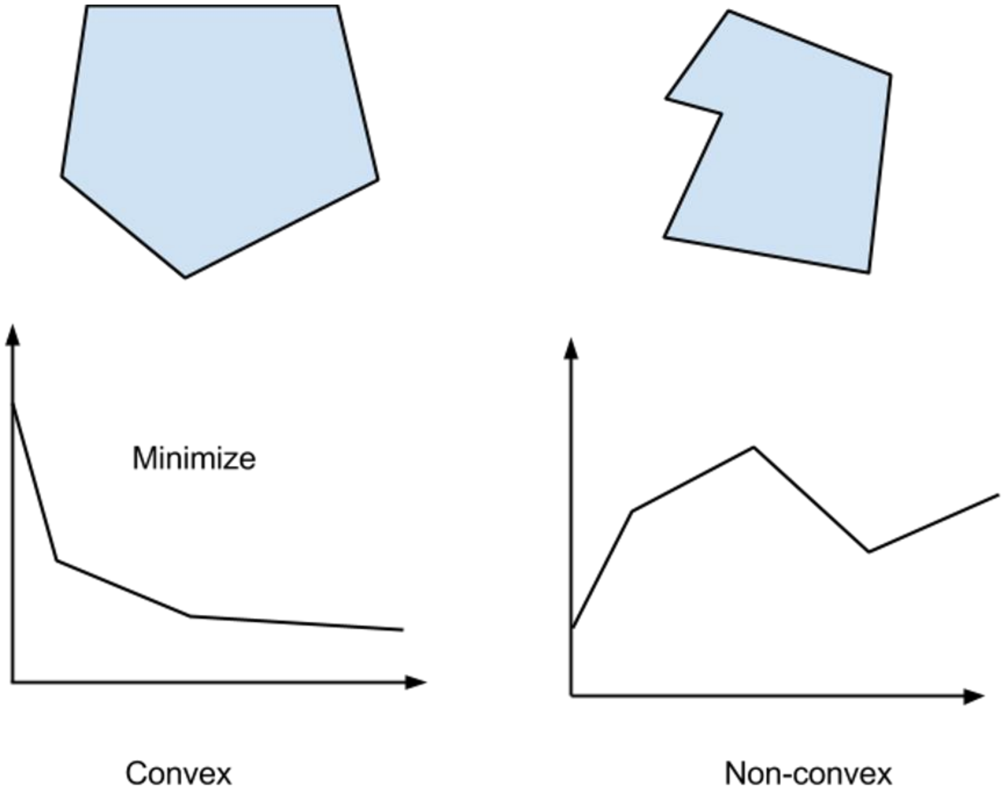
Coming Up:

- How do these logical expressions work?
- How to build models with complex logic?

Background

Convexity (or not)

Models with convex regions and convex functions are generally much easier to solve



Background

Special Ordered Sets

Special Ordered Set of type 1 – at most one variable may be ≠ 0 in a solution:

$$\mathrm{SOS1}\ (x_1, \cdots, x_n)$$

Special Ordered Set of type 2 – an <u>ordered</u> set where

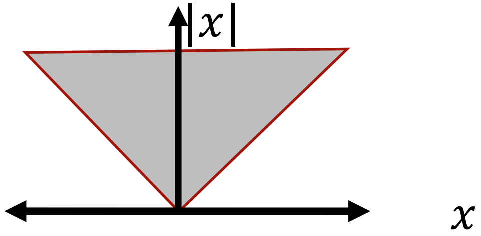
- At most two variables may be ≠ 0 in a solution
- Non-zero variables must be adjacent

$$\mathrm{SOS2}\ (x_1, \cdots, x_n)$$

Variables do not need to be integer.

Absolute value – Convex case

Simply substitute if absolute value function creates a convex model

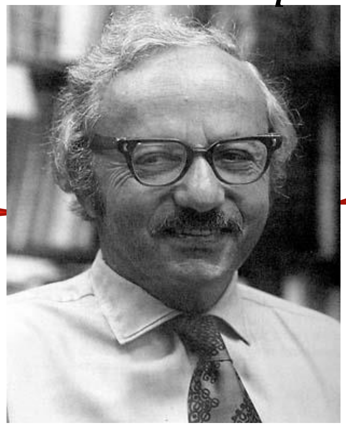


$\min |x|$
<linear constraints, some with x>
x free

$\min z$
z = $x_p + x_n$
x = $x_p - x_n$
<lin constraints, some with x>
x free, z, $x_p, x_n \geq 0$

What about x_p = 6, x_n = 4, x = 2, z = 10?

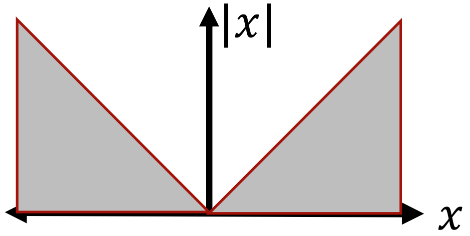
Dual argument: solution is feasible but not optimal, any optimal solution has either x_p or x_n = 0



Primal argument: 4 variables away from their bounds, but only room for three basic variables

Absolute value – Non-convex case

Use indicator variable and arbitrary big-M value to prevent both x_p and x_n positive



$\max |x|$

<linear constraints, some with x>

x free

$\max z$

$z = x_p + x_n$

$x = x_p - x_n = (x_p + \Delta) - (x_n + \Delta)$

<linear constraints, some with x>

$x_p \leq My$

$x_n \leq M(1 - y)$

$y \in \{0,1\},$

x free, z, $x_p, x_n \geq 0$

Can't rely on dual argument to force x_p or x_n to 0

Need some logic instead

Q: Any ideas on how to model this if no reasonable, finite big-M exists (ex: |x| can be infinite)?

Absolute value – SOS-1 constraint

GUROBI OPTIMIZATION

Use SOS-1 constraint to prevent both x_p and x_n positive

$\max |x|$

<constraints, some with x>

x free

- No big-M value needed
- Works for both convex and non-convex version
    - No branching needed in the convex case

$\max z$

$z = x_p + x_n$

$x = x_p - x_n$

$x_p, x_n \in$ SOS1

<constraints, some with x>

x free, z, $x_p, x_n \geq 0$

Q: Which formulation performs better?

Which LP relaxation is stronger?

$\max z$

$z = x_p + x_n$

$x = x_p - x_n$

$x_p \leq My$

$x_n \leq M(1-y)$

<constraints, some with x>

x free, $y \in \{0,1\}$, z, $x_p, x_n \geq 0$

Binary variables are better if modest M is known (or can be derived from model).

$\max z$

$z = x_p + x_n$

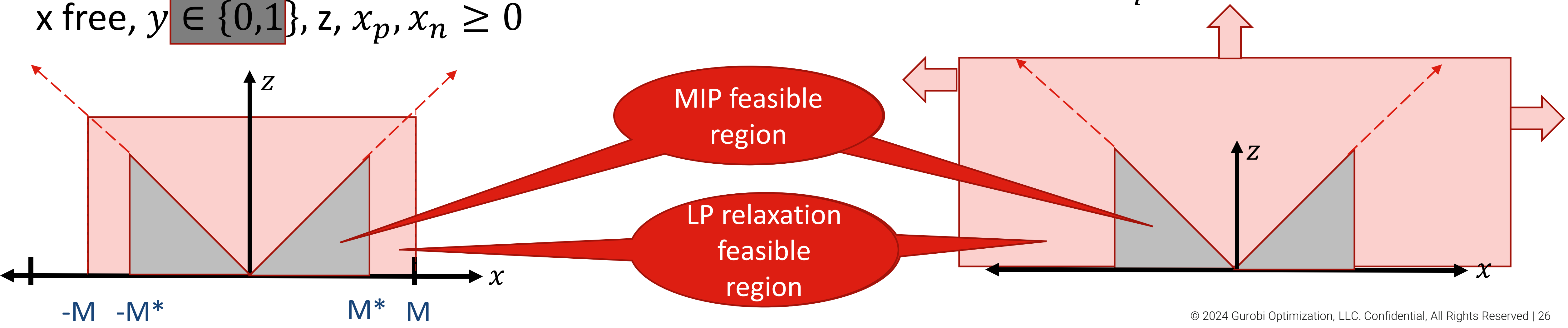
$x = x_p - x_n$

$x_p, x_n \in$ SOS-1

<constraints, some with x>

x free, z, $x_p, x_n \geq 0$

SOS is better, avoiding bad numerics, if M has to be arbitrarily large.



SOS constraints vs big-M representation

SOS constraints are handled with branching rules (not included in LP relaxation)

Gurobi will try to reformulate SOS constraints into a big-M representation during presolve

- Gurobi's presolve has very powerful bound strengthening features that can reduce the largest big-M
- User has control over this behavior with `PreSOS1BigM` and `PreSOS2BigM` parameters
- This sets a limit on the largest big-M necessary

SOS advantages

- Always valid (no reasonable M in some cases)
- Numerically stable; no large M values

Big-M advantages:

- LP relaxation is tighter
- Typically results in better performance for Gurobi's algorithms as long as M is relatively small
- LP relaxation weakens as M increases
- Formulations are equivalent if M is infinite

Piecewise linear functions

GUROBI OPTIMIZATION

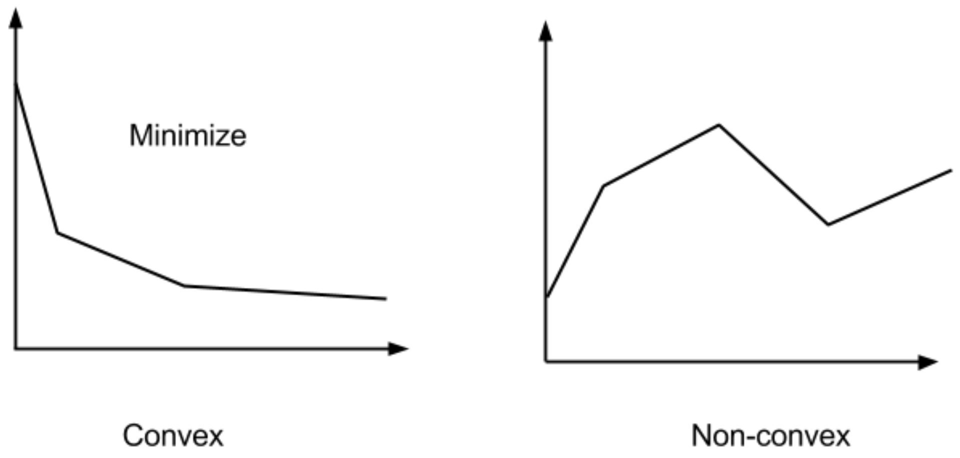
Generalization of absolute value functions

Convex case is "easy"

- Function represented by LP

Non-convex case is more challenging

- Function represented as MIP or SOS-2 constraints



Gurobi has an API for piecewise linear objectives (and constraints with 9.0)

- Built-in algorithmic support for the convex case (objective)
- Conversion to MIP is transparent to the user

Q: What are some potential applications?

Piecewise linear functions – Applications

GUROBI OPTIMIZATION

Piecewise linear functions appear in models all the time

- Fixed costs in manufacturing due to setup
- Economies of scale when discounts are applied after buying a certain number of items
- …

Also useful when approximating non-linear functions

- More pieces provide for a better approximation
- Trade-off between performance and exactness

Examples:

- Unit commitment models in energy sector
- Pooling problem
- Trigonometric, logarithmic and other nonlinear univariate functions

Gurobi versions >= 9.0 can handle the nonconvex quadratic pooling constraints explicitly

Gurobi versions >= 11.0 can handle selected univariate nonlinear functions explicitly

Piecewise linear functions – API

Only need to specify function breakpoints

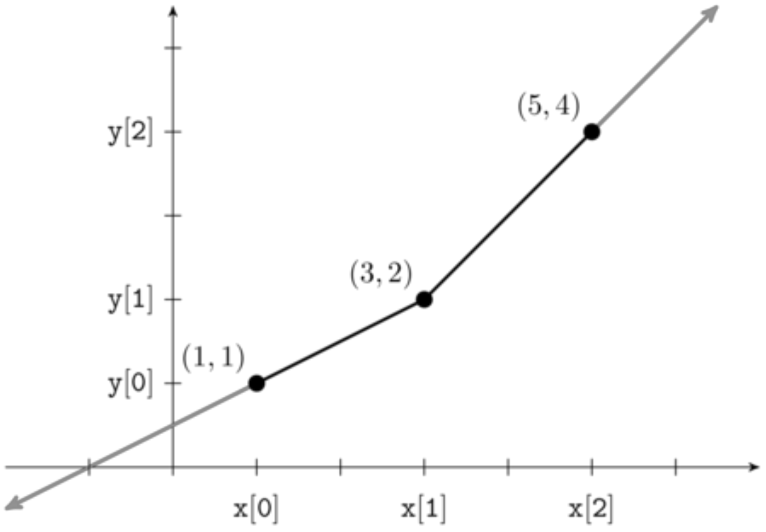
- No auxiliary variables or constraints necessary

Python example:

```
model.setPWLObj(x, [1, 3, 5], [1, 2, 4])
model.addGenConstrPWL(x, y, xpts, ypts, "gc")
```

x must be non-decreasing

- Repeat x value for a jump (or discontinuity)



Piecewise linear functions – SOS-2 constraint

Let (x_i, y_i) represent i^{th} point in piecewise linear function

To represent $y = f(x)$, use:

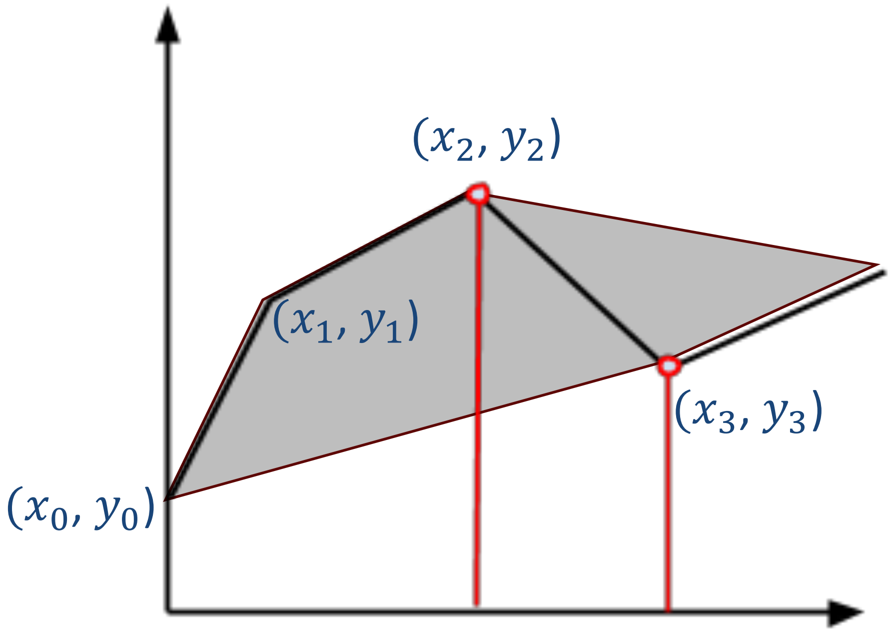
$$x = \sum_i \lambda_i x_i$$

$$y = \sum_i \lambda_i y_i$$

$$\sum_i \lambda_i = 1$$

$$\lambda_i \geq 0, \mathrm{SOS2}$$

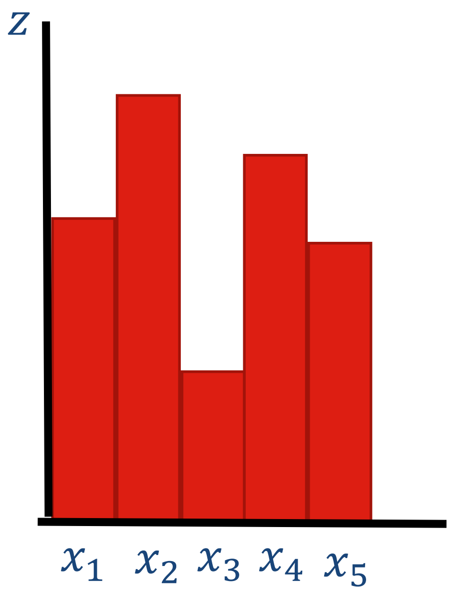
SOS-2 constraint is redundant if f is convex.



Min/max functions – Convex case

It is easy to minimize the largest value (minimax) or maximize the smallest value (maximin):

$$\min \left\{ \max_i x_i \right\} \quad \Rightarrow \quad \begin{array}{l} \min z \\ z \geq x_i \;\; \forall i \end{array}$$



Example: minimizing completion time of last job in machine scheduling application

Min/max functions – Non-convex case

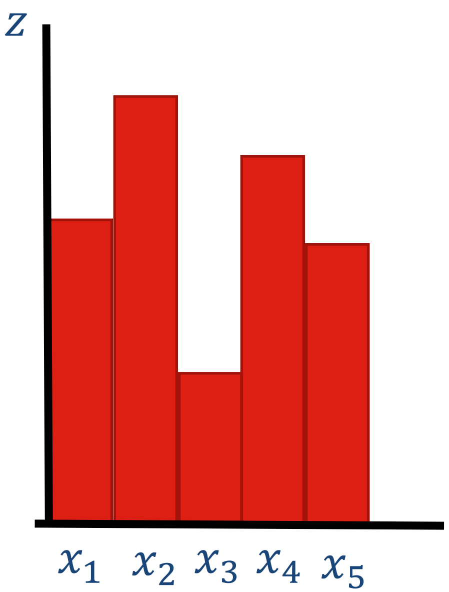
Harder to minimize the smallest value (minimin) or maximize the largest value (maximax)

- Use multiple indicator variables and a big-M value

$$\min \left\{ \min_i x_i \right\}$$

$$\min z$$
$$z \geq x_i - M(1 - y_i)$$
$$\sum_i y_i = 1$$
$$y_i \in \{0, 1\}$$

Only one of the constraints is binding



What is the smallest value of M that doesn't compromise the meaning of the model?

General Constraints for Logical Expressions

Function type	Function	Python syntax
Minimization	$y = \min\{x_1, x_2, x_3\}$	`addGenConstrMin(y, [x1,x2,x3])`
Maximization	$y = \max\{x_1, x_2, x_3\}$	`addGenConstrMax(y, [x1,x2,x3])`
Absolute value	$y = \lvert x \rvert$	`addGenConstrAbs(y, x)`

Note:
General constraints are also available for C, C++, Java, .NET, and R; we use Python syntax simply for illustration.

Logical conditions on binary variables

And

$x_1 = 1 \land x_2 = 1$ $\quad$ $x_1 + x_2 = 2$

Or

$x_1 = 1 \lor x_2 = 1$ $\quad$ $x_1 + x_2 \geq 1$

Exclusive Or (not both)

$x_1 = 1$ xor $x_2 = 1$ $\quad$ $x_1 + x_2 = 1$

At least/at most/counting

$x_i = 1$ for at least/most 3 i $\quad$ $\sum_i x_i \geq 3$

If-then (implication)

if $x_1 = 1$ then $x_2 = 1$ $\quad$ $x_1 \leq x_2$

Logical conditions – Variable result

And

$y = (x_1 \land x_2)$

$$y \leq x_1$$
$$y \leq x_2$$
$$y \geq x_1 + x_2 - 1$$

In other words,
$y = x_1 * x_2$

Or

$y = (x_1 \lor x_2)$

$$y \geq x_1$$
$$y \geq x_2$$
$$y \leq x_1 + x_2$$

Exclusive Or (not both)

$y = (x_1 \text{ xor } x_2)$

$$\left.\begin{aligned} y &\geq x_1 - x_2 \\ y &\geq x_2 - x_1 \end{aligned}\right\}$$

In other words,
$y = |x_1 - x_2|$
y true when x1 or x2 true

$$y \leq x_1 + x_2$$

y false when x1 and x2 false

$$y \leq 2 - x_1 - x_2$$

y false when x1 and x2 true

General Constraints for Logical Expressions

Function type	Condition	Python syntax
And	$y = (x_1 = 1 \ \wedge \ x_2 = 1)$	`addGenConstrAnd(y, [x1,x2])`
Or	$y = (x_1 = 1 \ \vee \ x_2 = 1)$	`addGenConstrOr(y, [x1,x2])`

Note:
General constraints are also available for C, C++, Java, .NET, and R; we use Python syntax simply for illustration.

Logical conditions on constraints

- Add indicator variables for each constraint
- Enforce logical conditions via constraints on indicator variables
- And constraints
  - Just add the individual constraints to the model

$$\sum_i a_{1i} x_i \leq b_1$$

and

$$\sum_i a_{2i} x_i \leq b_2$$

  - All other logical conditions require indicator variables

Logical conditions on inequalities – Or

Use indicator for the satisfied constraint, plus big-M value:

$$\sum_i a_{1i}x_i \leq b_1$$

or

$$\sum_i a_{2i}x_i \leq b_2$$

or

$$\sum_i a_{3i}x_i \leq b_3$$

$$\sum_i a_{1i}x_i \leq b_1 + \mathrm{M}(1 - \mathrm{y}_1)$$

$$\sum_i a_{2i}x_i \leq b_2 + \mathrm{M}(1 - \mathrm{y}_2)$$

$$\sum_i a_{3i}x_i \leq b_3 + \mathrm{M}(1 - \mathrm{y}_3)$$

$$y_1 + y_2 + y_3 \geq 1$$

$$y_1, y_2, y_3 \in \{0, 1\}$$

What is the smallest value of M that doesn't compromise the meaning of the model?

Logical conditions on equalities – Or

GUROBI OPTIMIZATION

- Add a free variable to each equality constraint to measure slack or surplus
- Use indicator variable to designate whether slack/surplus is zero

$$\sum_i a_{ki} x_i = b_k$$

$$\Rightarrow$$

$$\sum_i a_{ki} x_i + w_k = b_k$$

$$w_k \leq M(1 - y_k)$$ (Slack)

$$w_k \geq -M(1 - y_k)$$ (Surplus)

$$y_1 + \cdots + y_k \geq 1$$

$$y_1, \ldots, y_k \in \{0,1\}$$

Logical conditions on constraints – At least

At least constraints

- Generalizes the "or" constraint
- Use indicator or big M (if not too big) for the satisfied constraints
- Count the binding constraints via a constraint on indicator variables

Example:

- At least 4 constraints must be satisfied with

$$y_1 + y_2 + \cdots + y_k \geq 4$$

$$\sum_i a_{ki} x_i + w_k = b_k$$

$$w_k \leq M(1 - y_k)$$

$$w_k \geq -M(1 - y_k)$$

$$y_1 + \cdots + y_k \geq 4$$

$$y_1, \ldots, y_k \in \{0,1\}$$

Logical conditions on constraints – If-then

Indicator General Constraint represents if-then logic

- If $z = 1$ then $x_1 + 2x_2 - x_3 \geq 2$

```
addGenConstrIndicator(z, 1, x1+2*x2-x3 >= 2)
```

- The condition $(z = 1)$ must be a binary variable (z) and a value $(\mathbf{0}$ or $\mathbf{1})$

Semi-continuous variables

Many models have special kind of "or" constraint

$$x = 0 \;\; \vee \;\; 40 \leq x \leq 100$$

- This is a semi-continuous variable
- Semi-continuous variables are common in manufacturing, inventory, power generation, etc.
- Semi-integer variables are of similar form with an added integrality restriction

Two techniques for semi-continuous variables

1. **Add the indicator yourself:**

$$40y \leq x \leq 100y, y \in \{0,1\}$$

- Good performance but requires explicit upper bound on the semi-continuous variable

2. **Let Gurobi handle variables you designate as semi-continuous (or semi-integer)**

```
x = model.addVar(vtype=GRB.SEMICONT, lb=40, ub=100, name="x")
x = model.addVar(vtype=GRB.SEMIINT, lb=40, ub=100, name="x")
```

- practical option when upper bound is large or non-existent

Combined logical constraints

Limit on number of non-zero semi-continuous variables

Easy if you use indicator variables

$$40y_i \leq x_i \leq 100y_i$$
$$\sum_i y_i \leq 30$$

Need to model the logic yourself (instead of using semi-continuous variables), otherwise you cannot restrict the non-zero semi-variables.

Selecting big-M values

Want big-M as tight (small) as possible

Example:

$$x_1 + x_2 \leq 10 + My$$

$$\text{if } x_1, x_2 \leq 100 \text{ then } M = 190$$

Presolve will do its best to tighten big-M values.

Tight, constraint-specific big-M values are better than a giant one-size-fits-all big-M

- Too large big-M leads to poor performance and numerical problems
- Pick big-M values specifically for each constraint
- Can look at statistics of presolved model to assess how well Gurobi reduced the big M coefficients

What is the smallest value of M if $x_1, x_2 \leq 350$?

Selecting big-M values

GUROBI OPTIMIZATION

Want big-M as tight (small) as possible

Presolve will do its best to tighten big-M values.

But what if presolve doesn't provide any tighter big M values?

$$x_1 + x_2 \le My$$

<other constraints involving x_1, x_2>

$$x_1 + x_2 \ge 100000$$

Infeasible: M < 100000
Try again with rhs < 100000

Or run infeasibility finder; IIS may reveal group of constraints from which tighter M can be derived.

Feasible: Try again with rhs > 100000
Or solve a subproblem to determine M:

Maximize $x_1 + x_2$
s.t.
<all constraints in the model>

M = optimal subproblem objective value

New Optimization Based Bound Tightening in Gurobi 10.0 does this on selected individual variables

Helper Functions

General constraint helper functions specific to gurobipy to simplify construction of constraints:

```
max_(), min_(), abs_(), and_(), or_(), norm()
```

```
model.addConstr(x == abs_(y))
model.addConstr(x == or_(y,z,w))
```

Integrate general constraints with addConstrs():

```
X, Y, Gcons = model.addVars(10), model.addVars(10), {}
for i in X:
  Gcons[i] = model.addGenConstrMin(X[i],  [Y[i], 10], name="Gc")
```

```
X, Y = model.addVars(10), model.addVars(10)
Gcons = model.addConstrs((X[i] == min_(Y[i],10) for i in X), name="Gc")
```


Non-linear functions

General non-linear functions (of decision variables) are supported by Gurobi 9.0+

- e^x, a^x
- $\ln(x), \log_a(x)$
- $\sin(x), \cos(x), \tan(x)$
- x^a
- $ax^3 + bx^2 + cx + d$

```
addGenConstrExp(), addGenConstrExpA()
addGenConstrLog(), addGenConstrLogA()
addGenConstrSin(), addGenConstrCos(), addGenConstrTan()
addGenConstrPow()
addGenConstrPoly()
```

- **Starting with Gurobi 11.0, these nonlinear functions are supported directly as well as via piecewise linear approximation.**
  - **Use the FuncNonlinear parameter and attribute to configure direct support**

Non-convex quadratic functions are supported directly by Gurobi 9.0+

- Some special cases have been already supported prior to version 9.0 (e.g., products involving at least one binary decision variable)

Elementary, univariate functions

- Gurobi 11.0 can handle selected univariate, smooth constraints $f(x) = y$
  - Trigonometric, power functions, logarithms, exponentials, etc.
  - Use them as building blocks for more elaborate functions

Example: Suppose we want to model

$$f(\theta) = \overbrace{\sqrt{1+\theta^2}}^{v} + \overbrace{\ln\big(\overbrace{\theta + \sqrt{1+\theta^2}}^{w}\big)}^{z} \le 2, \ \theta \ge 0$$

Then we introduce auxiliary variables $u, v, w, z \ge 0$ and constraints as follows:

$$\mathrm{u} = 1 + \theta^2, \qquad u = v^2, \qquad w = \theta + v, \qquad z = \ln w$$

Then $f(\theta) \le 2$ can be represented as $v + z \le 2$

Multiple Objectives

Working with multiple objectives

GUROBI OPTIMIZATION

Real-world optimization problems often have multiple, competing objectives:

- Cost functions/Revenue
- Satisfaction/Target achievement
- Fairness
- KPIs
- Error/Feasibility
- Switching Configurations

Approaches to handling more than one objective function:

- Integration into a single objective function by using a weighted combination of multiple objectives
- Hierarchical approach: solve model for each objective but limit degradation of previous results
- … or a combination of both

Multi-Objective API

Easy definition of multiple objectives

```
model.setObjectiveN(LinExpr, index, ...)
```

Used with getMultiobjEnv() to set different parameters for different subproblems

Variant 1

```
coef1 = [0, 1, 2, 3, 4]
coef2 = [4, 3, 2, 1, 0]
x = model.addVars(5)

model.setObjectiveN(x.prod(coef1), 0, priority=2, weight=1)
model.setObjectiveN(x.prod(coef2), 1, priority=1, weight=1)
```

Variant 2

```
model.setObjectiveN(x[1]+2*x[2]+3*x[3]+4*x[4], 0, priority=2, weight=1)
model.setObjectiveN(4*x[0]+3*x[1]+2*x[2]+x[3], 1, priority=1, weight=1)
```


Enhanced Multi-Objective control

Enhanced termination control for multi-objective optimization

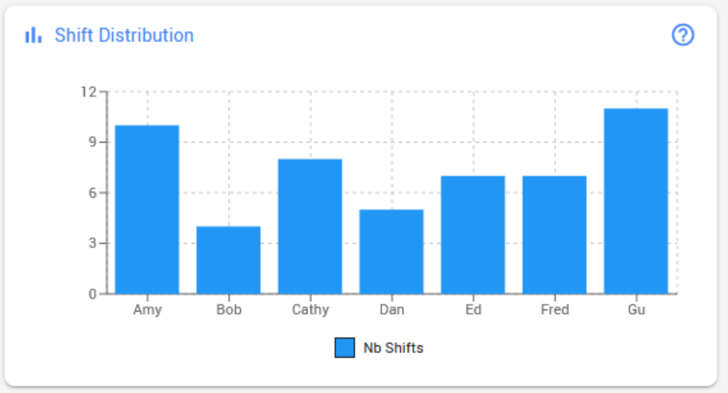
```
model.getMultiobjEnv()
```

```
env0 = model.getMultiobjEnv(0)
env1 = model.getMultiobjEnv(1)

env0.setParam('TimeLimit',100)
env1.setParam('TimeLimit',10)

model.optimize()
model.discardMultiobjEnvs()
```

- Allows for fine-grained control of each multi-objective optimization pass
  - Algorithmic choices
  - Termination criteria
- One optimization pass per objective priority level
  - Settings for additional objectives of same priority level are ignored

Note: MIP starts are only used for first priority level objective solve.

Example of multiple objectives



Workforce Scheduling Demo:

- First, minimize cost
- Then make the model “fair”

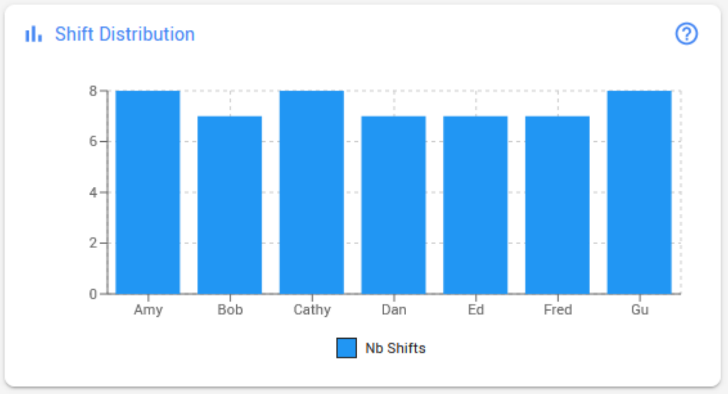
```
model.setObjectiveN(totSlack, index=index, priority=2,
                    reltol=extra_worker_trade_off, name='TotalSlack')
model.setObjectiveN(max_shift - min_shift, index=index,
                    priority=1, name='Fairness’)
```

Tolerances

Allow deviation from optimality

Gives “room” for finding lower-priority objectives

Could specify absolute instead of relative tolerance



Summary and Additional Learning

Takeaways

- Free variables can be expressed as difference of two nonnegative variables
- Big-Ms can pose challenges despite Gurobi's powerful bound strengthening
  - SOSs can help
  - IISs of reversed bounds can help
  - Optimization Based Bound Tightening can help
    - Gurobi parameter (OBBT)for individual variables
    - Create your own subproblem for more complex expressions
- Convex models usually are much easier to solve than nonconvex ones
- Gurobi continues to add support for more general constraints, facilitating the model development process
  - Before doing it yourself, check if there's an API for that
- Gurobipy modeling extensions (e.g. tupledict) enable simpler and faster code for extraction of subsets, an essential part of model building

GUROBI OPTIMIZATION

Check out our Modeling Examples in Jupyter Notebook:

www.gurobi.com/resource/modeling-examples-using-the-gurobi-python-api-in-jupyter-notebook/

Tutorial Example

Introductory

This modeling tutorial is at the introductory level, where we assume that you know P methods.

Example	Description
Intro to Mathematical Optimization Modeling	Learn the key components in the formu the Gurobi Optimizer to compute an op

Intermediate

These modeling examples are at the intermediate level, where we assume that you h you should know Python and be familiar with the Gurobi Python API.

Example	Description
Agricultural Pricing*	Determine the prices and demand for t of a country in order to maximize total from the sales of those products
Best Feature Selection for Forecasting	A linear regression problem that minim sum of squares subject to the constrai number of non-zero feature weights sh or equal to a given upper limit.
Customer Assignment	Address the optimal placement of facil of candidate locations) in order to mini between a company's facilities and its
Electrical Power Generation*	An electrical power generation problem unit commitment problem) by selecting of power stations to turn on in order to anticipated power demand over a 24-h
Factory Planning*	Solve a production planning problem a

Modeling Examples

*Problems from the fifth edition of Model Building in Mathematical Programming, by

Beginner

These modeling examples are at the beginner level, where we assume you know Pyt

Example	Description
Creating the Optimal Fantasy Basketball Lineup	Utilizes supervised machine learning to basketball players fantasy scores from and formulates an integer programmin the optimal lineup
3D Tic-Tac-Toe*	Arrange X's and O's on a three-dimensi board to minimize the number of comp diagonals.
Cell Tower Coverage	A simple covering problem that builds a towers to provide signal coverage to th of people possible.

Advanced

These modeling examples are at the advanced level, where we assume that you kno building mathematical optimization models. Typically, the objective function and/or Gurobi Python API.

Example	Description
Decentralization Planning*	For a given a set of departments of a c potential cities where these departmen determine the "best" location of each d order to maximize gross margins.
Farm Planning*	A production planning problem, where be made regarding which products to p which resources to use to produce thos

References

Model Building in Mathematical Programming, by H.P. Williams

Thank You

For more information: www.gurobi.com

Viva

GURORI SUMMIT

The Decision Intelligence Summit

September 2024 | Vegas

All In!

GUROBI OPTIMIZATION