Modeling

Best practices and techniques

Outline

- Model framework
- What makes a model difficult
- Numerical issues in models
- Programming pitfalls
- Model debugging
- Advanced modeling

Model components

- Decision variables

- Constraints
  - $\mathbf{Ax} = \mathbf{b}$ (linear constraints)
  - $\mathbf{l} \leq \mathbf{x} \leq \mathbf{u}$ (bound constraints)
  - some x_h integral (integrality constraints)
  - some x_i lie within second order cones (cone constraints)
  - $\mathbf{x}^T\mathbf{Q}_j\mathbf{x} + \mathbf{q}_j^T\mathbf{x} \leq \beta_j$ (convex quadratic constraints)
  - some x_k in SOS (special ordered set constraints)

- Objective function
  - minimize $\mathbf{x}^T\mathbf{Q}\mathbf{x} + \mathbf{c}^T\mathbf{x} + \alpha$ (convex quadratic function)

- Many of these are optional

Many applications

- Industries:
  - Advertising and marketing
  - Aerospace and defense
  - Airlines and airports
  - Automotive
  - Biotech, medical and pharmaceutical
  - Chemical and petroleum
  - Energy and utilities
  - Financial services
  - Food and beverage
  - Government
  - Ground and sea transportation
  - Industrial automation and machinery
  - Metals, materials and mining
  - Pulp and paper
  - Retail
  - Semiconductor
  - Sports scheduling
  - Supply chain
  - Telecom

- Business Problems:
  - Inventory optimization
  - Production mix
  - Machine allocation
  - Fuel use minimization
  - Maintenance planning
  - Less-than-truckload (LTL) loading
  - Inventory stocking & reordering
  - Vendor selection
  - Shipment planning
  - Capital budgeting
  - Cash management
  - Revenue optimization
  - Portfolio optimization
  - Fund cloning
  - Bond management
  - Workforce scheduling
  - Office assignment

Presolve is your friend

- Collection of presolve reductions applied before algorithms
  - Reduces problem size
  - Tightens formulation
- Presolve is very effective and finds the obvious reductions
  - Users do not need to apply as many reductions as possible
- Limits to what presolve can do
  - Can't find reductions that aren't actually implied by the model
  - Users have better understanding of underlying problem being modeled

What makes a model difficult

- Size
- Frequency – a series of related models
- Integer variables
- Quadratic expressions
- Numerical scaling

Model size

- Models typically become large via copies: regions, products, time, …
- Reducing model size is an art
  - What should be modeled
  - What should be approximated
- Some constraints may be treated as “lazy”
  - Pulled into the model only when violated
- Gurobi is parallel by default; parallel MIP consumes memory
- Solver considerations
  - Have enough physical memory (RAM) to load & solve model in memory
  - Use 64-bits
  - Try compute server or cloud

Frequency: a series of related models

- Model may not be so easy when there are many to solve

- Improve solve times via warm starts
  - Automatic: modify a model in memory rather than create a new model
  - Manual
    - LP: basis and primal/dual starts
    - MIP: start vectors

- Sometimes warm starts hurt more than they help;
  try solving from scratch via concurrent

Modifying a model

- Change coefficients
  - Objective
  - RHS
  - Matrix
  - Bounds
- Change variable types: continuous, integer, etc.
- Add variables or constraints
- Delete variables or constraints

- For small changes, modifying a model is more efficient than creating a new model
  - Reuse existing model data
  - Automatically use prior solution as warm-start for new model if possible
    - Some changes will force solver to discard LP basis

Python example: modifying a model

```
m = read("afiro.mps")
m.optimize()

x02 = m.getVarByName("x02")
x02.LB = 1
x02.UB = 1
m.optimize()

      Solved in 1 iterations and 0.00 seconds

m.reset()
m.optimize()

      Solved in 6 iterations and 0.00 seconds
```

Integer variables

- In most cases, integer variables make a model more difficult
- General integer variables tend to be more difficult than binary (0–1)
- Things to consider
  - Which general integers are necessary
  - Can some variables be approximated

Quadratic expressions

- Quadratic expressions are much more complex than linear
  - Especially for constraints: quadratic constraints require the barrier method
- Quadratic is essential for some applications
  - Financial risk
  - Engineering
- Quadratic constraints should *never* be used for logical expressions
  - Ex: $x = 0$ or $y = 0$ should *not* be modeled by $x\ y = 0$
  - More about logical expressions later

Numerical issues

- Models are solved via a series of continuous (LP/QP) relaxations
- Computer is limited by numerical precision, typically doubles
  - In solving an LP or MIP, billions of numerical calculations can lead to an accumulation of numerical errors
- Typical causes of numerical errors
  - Scale: too large of a range of numerical coefficients
  - Rounding of numerical coefficients
    - Ex: Don't write 1/3 as 0.333

Understanding Big-M coefficients

- "Big-M" coefficients represent penalty values or logic

- Overly large big-M values can give slow performance or wrong answers
  - Optimal objective from Gurobi Optimizer: -1.47e+08
  - Optimal objective from other solver: -2.72e+07

Example: Wrong answer with Big-M

- $y \le 1000000\ x$
  x binary
  $y \ge 0$

- With default value of IntFeasTol (1e−5):
  - $x = 0.0000099999$, $y = 9.9999$ is integer feasible!
  - y can be positive without forcing x to 1
    - y is positive without incurring the expensive fixed charge on x

Consequence of numerical issues

```
Linear constraint matrix    : 25050 Constrs, 15820 Vars, 94874 NZs
Variable types              : 14836 Continuous, 984 Integer
Matrix coefficient range    : [ 0.00099, 6e+06 ]
Objective coefficient range : [ 0.2, 65 ]
Variable bound range        : [ 0, 5e+07 ]
RHS coefficient range       : [ 1, 5e+07 ]
```

- Big-M values create too large of a range of coefficients
- By reformulating the model, user got fast, reliable results

Manufacturing model

```
Set parameter presolve to value 2
Set parameter method to value 0
Set parameter feasibilitytol to value 1e-4
Set parameter optimalitytol to value 1e-4

Gurobi Optimizer version 5.6.0 build 12784 (mac64)
Copyright (c) 2013, Gurobi Optimization, Inc.

Read MPS format model from file model.mps.bz2
Reading time = 1.51 seconds
BLANK: 285152 rows, 400408 columns, 995654 nonzeros
Optimize a model with 285152 rows, 400408 columns and 995654 nonzeros
Presolve removed 262129 rows and 328110 columns
Presolve time: 3.83s
Presolved: 23023 rows, 72343 columns, 248922 nonzeros

Iteration    Objective       Primal Inf.    Dual Inf.      Time
       0    5.7099766e+07   5.165086e+06   3.834811e+10      5s
   15063    2.5332837e+09   3.348119e+09   1.214340e+14      6s

Solved in 15063 iterations and 6.00 seconds
Infeasible model
```

Default tolerances, same model & algorithms

```
Set parameter presolve to value 2
Set parameter method to value 0

Gurobi Optimizer version 5.6.0 build 12784 (mac64)
Copyright (c) 2013, Gurobi Optimization, Inc.

Read MPS format model from file model.mps.bz2
Reading time = 1.40 seconds
BLANK: 285152 rows, 400408 columns, 995654 nonzeros
Optimize a model with 285152 rows, 400408 columns and 995654 nonzeros
Presolve removed 262129 rows and 328110 columns
Presolve time: 3.81s
Presolved: 23023 rows, 72343 columns, 248922 nonzeros

Iteration    Objective       Primal Inf.    Dual Inf.      Time
       0    5.6936870e+07   5.164790e+06   3.761677e+10      5s
   20222    2.9106155e+11   0.000000e+00   5.669215e+12      6s
   24691    8.7897809e+10   0.000000e+00   2.022579e+12     10s
   29288    8.4257419e+10   0.000000e+00   0.000000e+00     15s

Solved in 29288 iterations and 15.14 seconds
Optimal objective  8.425741931e+10
Warning: unscaled dual violation = 4.61657e-06 and residual = 7.34255e-06
```

What happened?

- Coefficients are numerically difficult

```
Linear constraint matrix    : 285152 Constrs, 400408 Vars, 995654 NZs
  Matrix coefficient range     : [ 0.01, 75729.9 ]
  Objective coefficient range : [ 0.5, 100000 ]
  Variable bound range         : [ 0, 0 ]
  RHS coefficient range        : [ 1, 1.90109e+07 ]
```

- Ideally, this model should be reformulated

- Setting numerical tolerances is not the way to fix this model
  - (Not the same as termination criteria)

Numeric issues: objective function

- Avoid large spread for objective coefficients
  - Often arises from penalties

- Example: minimize $100000\ x + 5000\ y + 0.001\ z$
  - Coefficient on x is large relative to others

- If x takes small values, rescale x
  - Change scale from units to thousandths of units
  - Generally limited to continuous variables

- If x takes large values, use hierarchical objectives
  - Optimize terms sequentially
  - Value of previous term introduced as a constraint

Programming pitfalls

- Always check the solution status
- Always check for exceptions
- Don't be a lazy programmer!

Ignoring optimization status

Input

```
import sys
from gurobipy import *

m = read(sys.argv[1])
m.optimize()
for v in m.getVars():
  print v.VarName, v.X
```

Output – failure!

```
Model is infeasible
Best objective -, best bound -,
gap -
x#1#1
Traceback (most recent call
last):
  File "test.py", line 6, in
<module>
    print v.VarName, v.X
  File "var.pxi", line 62, in
gurobipy.Var.__getattr__ (../../
src/python/gurobipy.c:7027)
  File "var.pxi", line 129, in
gurobipy.Var.getAttr (../../src/
python/gurobipy.c:7738)
gurobipy.GurobiError: Unable to
retrieve attribute 'X'
```

Ways to manage solution status

- Check the Status attribute to see the result of the optimization

```
if m.Status == GRB.OPTIMAL:
  for v in m.getVars():
    print v.VarName, v.X
```

- Use SolCount attribute to see whether any solutions were found

```
if m.SolCount > 0:
  for v in m.getVars():
    print v.VarName, v.X
```

- Catch exceptions…

Catching exceptions

- Easy to test for exceptions in OO interfaces:

```
try:
  m = read(sys.argv[1])
  m.optimize()
  for v in m.getVars():
    print v.VarName, v.X
except GurobiError as e:
  print "Error:", e
```

- With C, test the return code for every call to the Gurobi API

- Don’t be sloppy – always test for exceptions!
  - Many requests to Gurobi support could be avoided by testing for exceptions and reviewing the exception values

Model debugging

- Basic concepts
  - Naming variables and constraints
  - Model files

- Advanced debugging
  - Covered during troubleshooting session

Naming variables and constraints

- Set the `VarName` and `ConstrName` attributes to meaningful values
  - `flow_Atlanta_Dallas` is more useful than x3615

- Don't reuse names for multiple constraints or variables
  - API doesn't care about the `VarName` or `ConstrName` attributes
  - Create unique, descriptive names to help with debugging

Model files

MPS format

- Machine-readable
- Full precision
- Order is preserved

- Best for testing

LP format

- Easy to read and understand
- May truncate some digits
- Order is not preserved

- Best for debugging

MPS format example

- Java:
  - `m.write("mymodel.mps");`

- Now, you can use this model file for any kind of tests
  - Command-line: `> gurobi_cl mymodel.mps`
  - Python

```
m = read("mymodel.mps")
m.optimize()
```

- MPS files are a great way to export models from other solvers too
  - Examples of how to do this on our website
  - Useful for performance comparisons

LP format example

```
Maximize
  x + y + 2 z
Subject To
 c0: x + 2 y + 3 z <= 4
 c1: x + y >= 1
Bounds
Binaries
 x y z
End
```

Advanced modeling techniques

- Background info
- Range constraints
- Special functions: absolute value, piecewise linear, min/max
- Logical conditions on binary variables
- Logical conditions on constraints
- Semi-continuous variables
- Selecting big-M values

Background info: why not automate this?

- We know that other solvers provide specialized syntax for advanced modeling

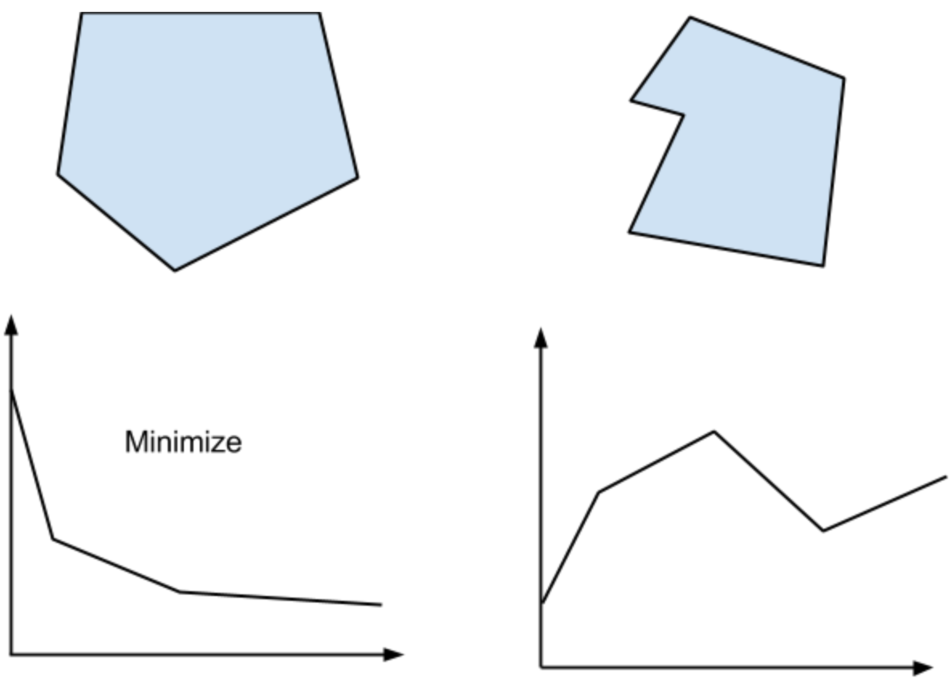
- Gurobi doesn't for several reasons
  - Makes the interface complicated and non-standard
  - "Training wheels" frequently leads to unsolvable models
  - We encourage you to construct models correctly

- So let's learn how to build advanced models efficiently

Background info: indicator variables & convexity

- Many advanced models are based on binary indicator variables
  - These variables indicate whether some condition holds

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



Convex

Non-convex

Background info: Special Ordered Sets

- Type 1: At most one variable in set may be nonzero

- Type 2: an *ordered* set where
  - At most 2 variables may be nonzero
  - Nonzero variables must be adjacent

- Variables need not be integer

Range constraints

- Many models contain constraints like:

$$L \le \sum_i a_i x_i \le U$$

- These can be written as:

$$r + \sum_i a_i x_i = U$$

$$0 \le r \le U - L$$

- The range constraint interface automates this for you
  - Semantic sugar coating
  - If you want to modify the range
    - Retrieve the additional range variable, named RgYourConstraintName
    - Modify the bounds on that variable
- For full control, it's easier to model this yourself

Absolute value: convex

- Simply substitute if absolute value function creates a convex model

$$\min |x|$$

$$\begin{aligned} &\min z \\ &z = x_p + x_n \\ &x = x_p - x_n \end{aligned}$$

Absolute value: non-convex

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

$$\max |x| \Rightarrow \begin{aligned} &\max z \\ &z = x_p + x_n \\ &x = x_p - x_n \\ &x_p \le My \\ &x_n \le M(1-y) \\ &y \in \{0,1\} \end{aligned}$$

Absolute value: SOS-1 constraint

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

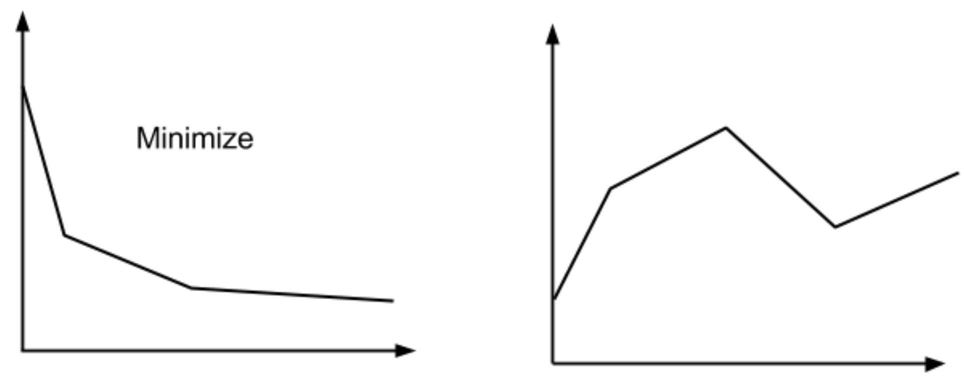
$$\max |x| \quad \Rightarrow \quad \begin{aligned} &\max z \\ &z = x_p + x_n \\ &x = x_p - x_n \\ &x_p, x_n \in \text{SOS-1} \end{aligned}$$

- No big-M value needed
- Works for both convex and non-convex functions
- Big-M version

Piecewise linear functions

- Generalization of absolute value functions

- Convex case: easy
  - Function represented by LP
- Non-convex case: more challenging
  - Function represented as MIP or SOS-2 constraints

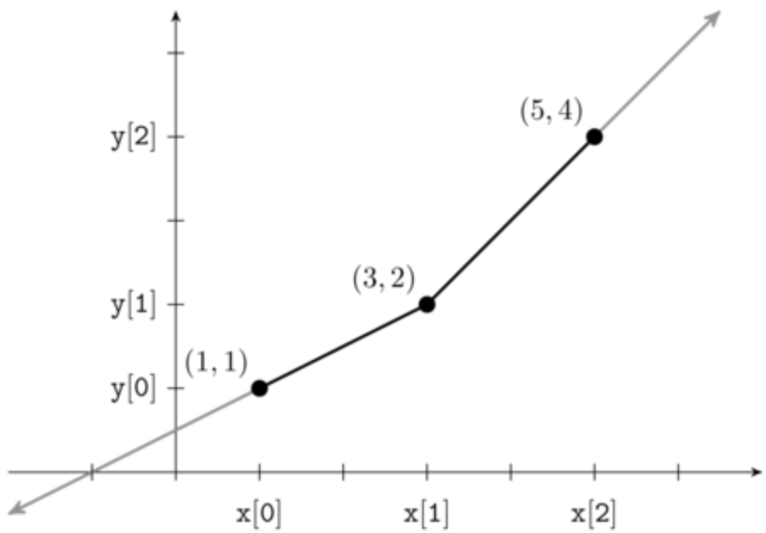


Convex Non-convex

- New in 6.0: Piecewise linear API with special support in the solver

Piecewise linear functions

- Specify function breakpoints
- x must be non-decreasing
  - Repeat x value for a jump



- Python example:

```
model.setPWLObj(x, [1, 3, 5], [1, 2, 4])
```

Min/max functions: convex

- Easy to minimize the largest value (minimax) or maximize the smallest value (maximin)

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

Min/max functions: non-convex

- Much more challenging to minimize the smallest value (minimin) or maximize the largest value (maximax)
  - Use indicator variables and a big-M value

$$\min\left\{\min_i x_i\right\} \Rightarrow \begin{aligned} &\min z \\ &z \geq x_i - M(1 - y_i) \\ &\sum_i y_i = 1 \\ &y_i \in \{0, 1\} \end{aligned}$$

Logical conditions on binary variables

- And
  $x_1 = 1$ and $x_2 = 1$
  $$x_1 + x_2 = 2$$
- Or
  $x_1 = 1$ or $x_2 = 1$
  $$x_1 + x_2 \geq 1$$
- Exclusive or (not both)
  $x_1 = 1$ xor $x_2 = 1$
  $$x_1 + x_2 = 1$$
- At least / at most / counting
  $x_i = 1$ for at least 3 i's
  $$\sum_i x_i \geq 3$$
- If-then
  if $x_1 = 1$ then $x_2 = 1$
  $$x_1 \leq x_2$$

Logical conditions: variable result

- And

  $y = (x_1 = 1 \text{ and } x_2 = 1)$

$$y \le x_1$$
$$y \le x_2$$
$$y \ge x_1 + x_2 - 1$$

- Or

  $y = (x_1 = 1 \text{ or } x_2 = 1)$

$$y \ge x_1$$
$$y \ge x_2$$
$$y \le x_1 + x_2$$

- Exclusive or (not both)

  $y = (x_1 = 1 \text{ xor } x_2 = 1)$

$$y \ge x_1 - x_2$$
$$y \ge x_2 - x_1$$
$$y \le x_1 + x_2$$
$$y \le 2 - x_1 - x_2$$

Logical conditions on constraints

- Add indicator variables for each constraint

- Enforce logical conditions via constraints on indicator variables

Logical conditions: And

- Trivial – constraints are always combined with “and” operator!

- All other logical conditions require indicator variables

Logical conditions: Or with inequalities

- Use indicator for the satisfied constraint, plus big-M values

$$\sum_i a_i^1 x_i \le b^1$$

or

$$\sum_i a_i^2 x_i \le b^2$$

or

$$\sum_i a_i^3 x_i \le b^3$$

$$\sum_i a_i^1 x_i \le b^1 + M(1 - y^1)$$

$$\sum_i a_i^2 x_i \le b^2 + M(1 - y^2)$$

$$\sum_i a_i^3 x_i \le b^3 + M(1 - y^3)$$

$$y^1 + y^2 + y^3 \ge 1$$

$$y^1, y^2, y^3 \in \{0, 1\}$$

Logical conditions: Or with equalities

- Add a *free* slack variable to each equality constraint
- Use indicator variable to designate whether slack is zero

$$\sum_i a_i^k x_i = b^k$$

$$\begin{aligned}
\sum_i a_i^k x_i + w^k &= b^k \\
w^k &\leq M(1 - y^k) \\
w^k &\geq -M(1 - y^k) \\
y^k &\in \{0, 1\}
\end{aligned}$$

Logical conditions: at least

- Generalizes the “or” constraint
- Use indicator for the satisfied constraints
- Count the binding constraints via a constraint on indicator variables

- Ex: at least 4 constraints must be satisfied:

$$y_1 + y_2 + \ldots + y_m \geq 4$$

What about if-then logic?

- Logically, A → B is the same as ~A ∨ B
- If constraint A is linear, this cannot be implemented

$$\sim\left(\sum_i a_i x_i \leq b\right) \Leftrightarrow \sum_i a_i x_i > b$$

- LP does not allow strict inequality (< or >)
- You cannot force an indicator to be 1 when constraint is satisfied with equality
- To implement a "not" constraint, all variables must be integer

Example of violated if-then constraint

$$\begin{aligned} &\text{if } x_1 + x_2 \le 10 \\ &\text{then } x_1 - x_2 \le 3 \end{aligned}$$

$$x_1 = 8$$
$$x_2 = 2$$
$$y^1 = 0$$

$$\begin{aligned} x_1 + x_2 &\ge 10 - 10y^1 \\ x_1 - x_2 &\le 3 + 1000(1 - y^1) \\ y^1 &\in \{0, 1\} \end{aligned}$$

Lesson of logical constraints

- Standard logical operators can be used with constraints involving only integer variables
- With continuous variables, the if or not operators cannot be used

Semi-continuous variables

- Many models have special kind of "or" constraint

$$x = 0 \text{ or } 40 \leq x \leq 100$$

- This is a semi-continuous variable

- Semi-continuous variables are common in manufacturing, inventory, power generation, etc.

- A semi-integer variable has a similar form, plus the restriction that the variable must be integer

Two techniques for semi-continuous variables

1. Add the indicator yourself

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

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

2. Let Gurobi handle variables you designate as semi-continuous
   - Only practical option when upper bound is large or non-existent

Example: Combined logical constraints

- Limit on number of non-zero semi-continuous variables

- Easy if you use indicator variables

$$40\, y_i \le x_i \le 100\, y_i$$

$$\sum_i y_i \le 30$$

- Perfect example when you should model logic yourself and not trust a black-box automated interface

Selecting big-M values

- Want big-M as tight (small) as possible
  - Ex: for $x_1 + x_2 \le 10 + My$, if $x_1, x_2 \le 100$ then $M = 190$
- Presolve will do its best to tighten big-M values
- Tight, constraint-specific big-M values are better than one giant big-M that is large enough for all constraints
  - Too large leads to poor performance and numerical problems
  - Pick big-M values specifically for each constraint