Modeling 2

Agenda

- Understanding advanced modeling techniques takes some time and experience
  - No exercises today
  - Ask questions!

- Part 1: Overview of selected modeling techniques
  - Background
  - 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

- Part 2: We go through the whole model development process
  - From problem description to mathematical model to Python model

Background – It's automated!

- Gurobi Optimizer 7.0 introduced General Constraints for popular logical expressions
  - Absolute value
  - Min/Max value
  - And/Or over binary variables
  - Indicator (if-then logic)

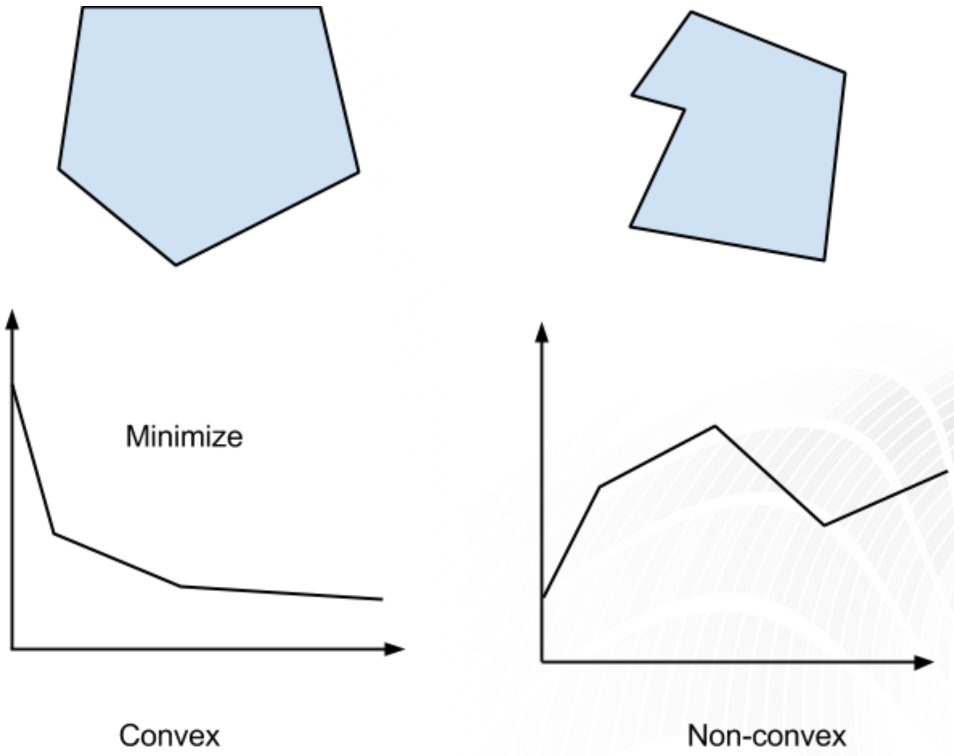
- The General Constraint syntax is a safe way to implement model logic

- Let's see
  - How these logical expressions work
  - How to build models with complex logic

Background – Indicator variables and convexity

- Many advanced models are based on binary indicator variables
  - Indicate whether or not some condition holds

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



Background – Special Ordered Sets

- Special Ordered Set of type 1 (SOS-1) – at most one variable in set may be non-zero

- Special Ordered Set of type 2 (SOS-2) – an *ordered* set where
    - At most two variables may be non-zero
    - Non-zero variables must be adjacent

- Variables need not be integer

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 (semantic sugar-coating)
- If you need 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

Non-linear functions

- General non-linear functions (of decision variables) are not directly supported by Gurobi

- Examples:
  - log(x)
  - sqrt(x)
  - cos(x), sin(x), tan(x), …
  - 2^x
  - …

- Non-convex quadratic functions are not supported either
  - Directly supported in some special cases (ex: binary decision variables)

- However, we can linearize or approximate some of these through modeling techniques

Absolute value – Convex case

- Simply substitute if absolute value function creates a convex model

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

Absolute value – Non-convex case

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

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

- 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

- 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 version

- Q: Which will perform better?

SOS constraints vs big-M representation

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

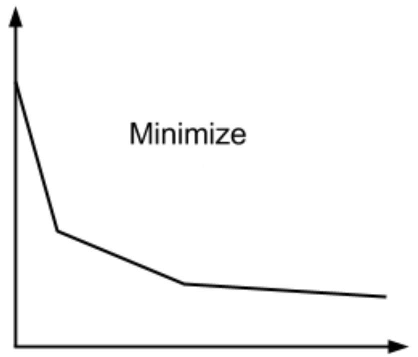
- SOS advantages:
  - Always valid (no reasonable M in some cases)
  - Numerically stable

- Big-M advantages:
  - LP relaxation is tighter
  - Typically results in better performance for Gurobi's algorithms as long as M is relatively small

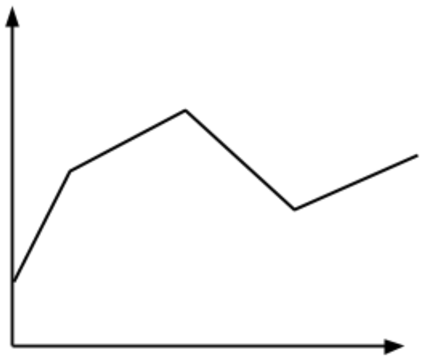
- In fact, Gurobi will try to reformulate SOS constraints into a big-M representation during presolve
  - User has control over this behavior with `PreSOS1BigM` and `PreSOS2BigM` parameters
  - Establishes limit on the largest big-M necessary

Piecewise linear functions

- 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
    - Built-in algorithmic support for the convex case
    - Conversion to MIP is transparent to the user
- Q: What are some potential applications?



Convex



Non-convex

Piecewise linear functions – Applications

- Piecewise linear functions appear in models all the time
- Examples:
  - Fixed costs in manufacturing due to setup
  - Economies of scale when discounts are applied after buying a certain number of items
  - …

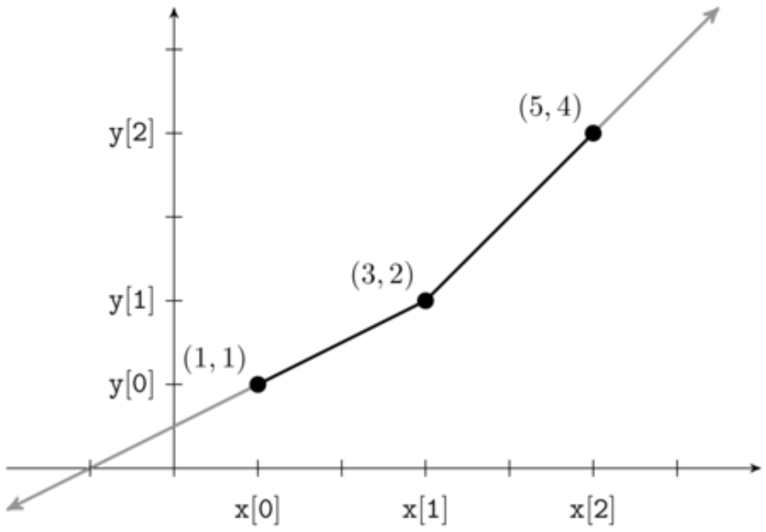
- Also useful when approximating non-linear functions
  - More pieces provide for a better approximation
- Examples:
  - Unit commitment models in energy sector
  - …

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])
```

- `x` must be non-decreasing
  - Repeat `x` value for a jump (or discontinuity)

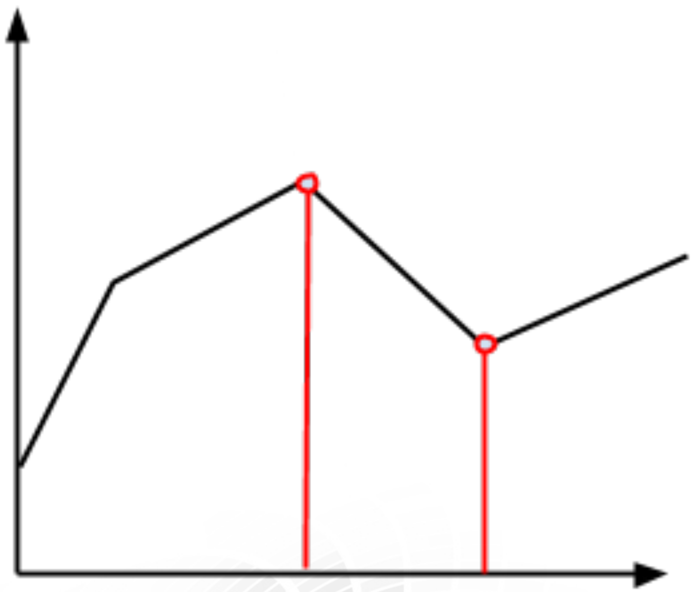


Piecewise linear functions – SOS-2 constraint

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

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

$$x = \sum_i \lambda_i x_i$$
$$y = \sum_i \lambda_i y_i$$
$$\sum_i \lambda_i = 1$$
$$\lambda_i \geq 0, \text{SOS-2}$$



- SOS-2 constraint is redundant if f is convex

- Binary representation also exists

Min/max functions – Convex case

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

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

- Ex: minimize 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

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

General Constraints for Logical Expressions

GUROBI OPTIMIZATION

Function	Python syntax
$y = \min\{x_1, x_2, x_3\}$	`addGenConstrMin(y, [x1,x2,x3])`
$y = \max\{x_1, x_2, x_3\}$	`addGenConstrMax(y, [x1,x2,x3])`
$y = \text{abs}\, x$	`addGenConstrAbs(y, x)`

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

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)

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

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

General Constraints for Logical Conditions

GUROBI OPTIMIZATION

Condition	Python syntax
$y = (x_1 = 1 \text{ and } x_2 = 1)$	`addGenConstrAnd(y, [x1,x2])`
$y = (x_1 = 1 \text{ or } x_2 = 1)$	`addGenConstrOr(y, [x1,x2])`

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

Logical conditions on constraints – Overview

- Add indicator variables for each constraint
- Enforce logical conditions via constraints on indicator variables

Logical conditions on constraints – And

- Trivial – constraints are always combined with "and" operator!
- All other logical conditions require indicator variables

Logical conditions on inequalities – Or

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

$$\sum_i a_i^1 x_i \le b^1$$
or
$$\sum_i a_i^2 x_i \le b^2$$
or
$$\sum_i a_i^3 x_i \le b^3$$

$$\Rightarrow$$

$$\sum_i a_i^1 x_i \le b^1 + M(1 - y^1)$$
$$\sum_i a_i^2 x_i \le b^2 + M(1 - y^2)$$
$$\sum_i a_i^3 x_i \le b^3 + M(1 - y^3)$$
$$y^1 + y^2 + y^3 \ge 1$$
$$y^1, y^2, y^3 \in \{0, 1\}$$

Logical conditions on equalities – Or

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

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

Logical conditions on constraints – 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 with $y_1 + y_2 + \ldots + y_m \geq 4$

Logical conditions on constraints – If-then

- Indicator General Constraint represents if-then logic
  - If $z = 1$ then $x_1 + 2x_2 - x_3 \geq 2$
  - Syntax: addGenConstrIndicator(z, 1, x1+2*x2-x3 >= 2)

- The condition $(z = 1)$ must be a binary variable (z) and a value (0 or 1)
  - Q: How do you transform this to other types of logic?

Semi-continuous variables

- Many models have special kind of "or" constraint

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

- 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

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

$$\sum_i y_i \le 30$$

- By modeling the logic yourself, fewer variables are needed

Selecting big-M values

- Want big-M as tight (small) as possible
  - Ex: for $x_1 + x_2 \leq 10 + My$, if $x_1, x_2 \leq 100$ 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

Numerical issues

Numerical issues can be problematic

- 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
- Can lead to slow performance or wrong answers
  - Optimal objective from Gurobi Optimizer: -1.47e+08
  - Optimal objective from other solver: -2.72e+07
- Typical causes of numerical errors
  - Rounding of numerical coefficients
    - Ex: Don't write 1/3 as 0.333
  - Scaling – too large of a range for numerical coefficients
    - Ex: big-M values

Example – Trickle flow 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

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

From the business problem to the mathematical problem to the Python implementation

Factory Planning

- Example from our website:
  http://www.gurobi.com/resources/examples/factory-planning-I

- Download Jupyter Notebook and Python source
  http://files.gurobi.com/training/factory.zip

- In production planning problems, choices must be made about how many of **what products to produce using what resources (variables)** in order to **maximize profits or minimize costs (objective function)**, while meeting a **range of constraints**. These problems are common across a broad range of manufacturing situations.

- We will develop the mathematical model, the Python Implementation and a nice tabular output of the result all within a single Jupyter Notebook.

Factory Planning: Interactive Model Development

Demo 3 - Factory Planning

Source: http://www.gurobi.com/resources/examples/factory-planning-I

Problem Description

A factory makes seven products (Prod 1 to Prod 7) using a range of machines including:

- Four grinders
- Two vertical drills
- Three horizontal drills
- One borer
- One planer

Each product has a defined profit contribution per unit sold (defined as the sales price per unit minus the cost of raw materials). In addition, the manufacturing of each product requires a certain amount of time on each machine (in hours). The contribution and manufacturing time value are shown below. A dash indicates the manufacturing product for the given product does not require that machine.

	PROD 1	PROD 2	PROD 3	PROD 4	PROD 5	PROD 6	PROD 7
Contribution to profit	10	6	8	4	11	9	3
Grinding	0.5	0.7	-	-	0.3	0.2	0.5
Vertical drilling	0.1	0.2	-	0.3	-	0.6	-
Horizontal drilling	0.2	-	0.8	-	-	-	0.6
Boring	0.05	0.03		0.07	0.1	-	0.08
Planing	-	-	0.01	-	0.05	-	0.05

In each of the six months covered by this model, one or more of the machines is scheduled to be down for maintenance and as a result will not be available to use for production that month. The maintenance schedule is as follows:

Month	Machine
January	One Grinder
February	Two Horizontal Drills
March	One borer
April	One vertical drill
May	One grinder and one vertical drill
June	One horizontal drill

There limitations to how many of each product can be sold in a given month. These limits are shown below:

Month	PROD 1	PROD 2	PROD 3	PROD 4	PROD 5	PROD 6	PROD 7
January	500	1000	300	300	800	200	100
February	600	500	200	0	400	300	150
March	300	600	0	0	500	400	100
April	200	300	400	500	200	0	100
May	0	100	500	100	1000	300	0
June	500	500	100	300	1100	500	60

Up to 100 units of each product may be stored in inventory at a cost of $0.50 per unit per month. At the start of January there is no product inventory. However, by the end of June there should be 50 units of each product in inventory.

The factory produces product six days a week using two eight-hour shifts per day. It may be assumed that each month consists of 24 working days. Also, for the purposes of this model, there are no production sequencing issues that need to be taken into account.

What should the production plan look like? Also, recommend any price increases and identify the value of acquiring any new machines.

Model Formulation

Sets

Let T be a set of time periods (months), where $t_0 \in T$ is the first month and $t_e \in T$ the last month.

Let P be a set of products and M be a set of machines.

Parameters

- For each product $p \in P$ and each type of machine $m \in M$ we are given the time $f_{p,m}$ (in hours) the product $p \in P$ needs to be manufactured on the machine $m \in M$.
- For each month $t \in T$ and each product $p \in P$ we are given the upper limit on sales of $l_{t,p}$ for that product in that month.
- For each product $p \in P$ we are given the profit k_p.
- For each month $t \in T$ and each machine $m \in M$ we are given the number of available machines $q_{t,m}$.
- Each machine can work g hours a month.
- There can be z products of each type stored in each month and storing costs r per product per month occur.

The capacity constraints ensure that per month the time all products needs on a certain kind of machines is lower or equal than the available hours for that machine in that month multiplied by the number of available machines in that month. Each product needs some machine hours on different machines. Each machine is down in one or more months due to maintenance, so the number of available machines varies per month. There can be multiple machines per machine type.

$$\sum_{p \in P} f_{p,m} \cdot b_{t,p} \leq g \cdot q_{t,m} \quad \forall t \in T, \forall m \in M$$

Python Implementation

Import `gurobipy` module:

In [1]:
```
from gurobipy import *
```

Data definition

Define sets P, M and T:

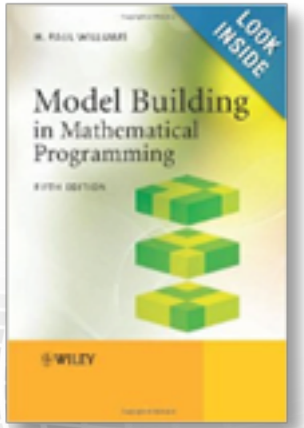
In [2]:
```
products = ["Prod1", "Prod2", "Prod3", "Prod4", "Prod5", "Prod6", "Prod7"]
machines = ["grinder", "vertDrill", "horiDrill", "borer", "planer"]
months = ["Jan", "Feb", "Mar", "Apr", "May", "Jun"]
```

Values for parameter k_p (profit contribution per product $p \in P$):

In [3]:
```
profit_contribution = { "Prod1" : 10, "Prod2" : 6, "Prod3" : 8, "Prod4" : 4,
                        "Prod5" : 11,  "Prod6" : 9, "Prod7" : 3 }
```

Additional resources

- Visit http://www.gurobi.com/documentation/ for more information on Gurobi interfaces
  - Quick Start Guide
  - Reference Manual
- Explore our examples at http://www.gurobi.com/resources/examples/example-models-overview
  - Functional Examples
  - Modeling Examples
  - Interactive Examples
- Read *Model Building in Mathematical Programming* by H. Paul Williams
  - Great introduction to modeling business problems with math programming



- For more guidance on numeric issues, refer to http://files.gurobi.com/Numerics.pdf

Thank you – Questions?