# Column Generation

- Give Gurobi a small portion of the variables
  - Variables left out of model are effectively fixed to 0
- Iteratively add additional columns
  - Solve LP with a subset of all possible variables
  - Identify and add variables that would improve the solution
- More columns than rows
  - Low proportion of variables have nonzero values
  - Number of potential variables may even be exponential
  - Ratio of rows to columns should be close to 0

# Column Generation Theory

$$
\begin{aligned}
z^* = \min_{x} \quad & \sum_{i \in I} c_i x_i \\
\text{s.t.} \quad & \sum_{i \in I} a_{ij} x_i = b_j : \pi_j, \quad j \in J \\
& x_i \geq 0, \quad i \in I.
\end{aligned}
$$

- The reduced cost of variable $i$ can be computed as: $c_i - \sum_j a_{ij}\pi_j$
- If $c_i - \sum_j a_{ij}\pi_j \geq 0$ for all $i \in I$, we have a proof of optimality.
- Can compute the reduced cost even for variables not yet added to the model.

# What you need to know

- Models can be solved iteratively
  - Model remains in memory after a call to Model.optimize().
  - Can then call addVar to add new decision variables, then re-optimize.
- Variables are added after constraints are added.
  - Pass a Column object in to Model.addVar to back-populate constraints.
- Compute reduced costs from dual prices.
  - Choose variable(s) with negative reduced costs to add to the model.
  - If all reduced costs are non-negative, then the existing variables produced an optimal solution.

# Columnwise modeling in Gurobi

- Rowwise Pattern
  - Create variables (columns)
  - Create LinExpr list of (coef, Var) pairs
  - Create constraints (rows) in terms of LinExpr
- Columnwise Pattern
  - Create constraints (rows)
  - Create Column list of (coef, GRBConstr) pairs
  - Create variables in terms of Column

# Gurobi API for column generation

- Model.addVar(lb, ub, obj, vtype, name, column)
- Column(coeffs, constrs)
  - col = Column(3, c1)
  - col = Column([1, 2], [c1, c2])
- Dual values come from Pi attribute of Constr objects.

# Transportation Problem

$$
\begin{aligned}
\min_{x} \quad & \sum_{i \in I} \sum_{j \in J} c_{ij} x_{ij} \quad \text{(minimize shipping costs)} \\
\text{s.t.} \quad & \sum_{i \in I} x_{ij} = d_j, \quad j \in J \quad \text{(satisfy demand)} \\
& \sum_{j \in J} x_{ij} \leq u_i, \quad i \in I \quad \text{(don't exceed capacity)} \\
& x_{ij} \geq 0, \quad i \in I,\ j \in J \quad \text{(ship nonnegative quantities)}
\end{aligned}
$$

Consider $x_{ij}$ for a particular warehouse $i$ and customer $j$.

```
col = Column([1, 1], [capacity_constrs[i], demand_constrs[j]])
var = model.addVar(obj=ship_costs[i][j], column=col)
```

# Exercise

- Write addColumn method in FacilityLocationColumn.
- Problem should solve exactly like the row-oriented model

# Column Generation Loop

- Can add any columns with negative reduced costs
- Need to add at least one column per iteration
  - Add columns with negative reduced costs
  - Can add multiple columns per iteration to reduce number of iterations

# Exercise

- Write getRC(warehouse, customer) method
- Write addColumns() method
- Experiment with different strategies
    - Add first variable with negative reduced cost
    - Add all variables with negative reduced costs
    - Add variable with most negative reduced cost

# What is a Convex Program?

## Definition

$$
\begin{aligned}
\text{minimize:} \quad & f(x_1, x_2, \ldots, x_n) \\
\text{subject to:} \quad & g_i(x_1, x_2, \ldots, x_n) \le b_i, \quad i = 1, \ldots, m \\
& x_j \ge 0, \quad j = 1, \ldots, n
\end{aligned}
$$

f, $g_i$ are all convex.

# Convexity

- Averages are better than extremes.
- If $x$ and $y$ are feasible, then $\frac{x+y}{2}$ is feasible.
- The objective function evaluated at $\frac{x+y}{2}$ must be better than the average of $x$ and $y$.
- All local optimum are globally optimal

# What does convex look like?

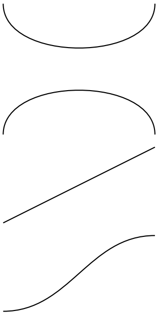
- Slopes are non-decreasing as we move from left to right.
- Not as obvious in multiple dimensions.

Convex

Concave

Both

Neither



# Convex Quadratic Programming

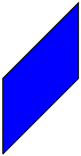
$$\text{minimize: } \sum_i c_i x_i + \sum_{i,j} q_{ij} x_i x_j$$

$$\text{subject to: } \sum_i a_{ik} x_i + \sum_{i,j} d_{ijk} x_i x_j \leq b_k$$

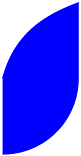
- All functions must be convex (for minimization, $\leq$ constraints).
- Unlike in the linear case, minimize/maximize; $\leq$, $\geq$ are not interchangeable.

# Feasible Region

Linear



Quadratic



# What you can't do with Quadratic Expressions

- nonlinear equality constraints
- $x_i(1 - x_i) \leq 0$ to simulate binary variables.
- "Bilinear" programming ($\sum_i x_i \cdot y_i$).
  - pricing and allocations (price and sales variables)
  - blending problems (concentrations and quantities)

# What is allowed

- $x^t Qx$ Hessian Matrix $Q$ must be positive definite.
  - All eigenvalues are positive.
- $f(\alpha x + (1-\alpha)y) \leq \alpha f(x) + (1-\alpha)f(y)$
- Maximizing a concave function
- lower bound on concave function ($f(x) \geq b$ for concave $f$).

# Convexity

- $f(x)$ convex $\equiv -f(x)$ concave
- if $f(x)$ and $-f(x)$ are both convex then $f(x)$ is linear
  - $f(x) = b \equiv f(x) \leq b$ **and** $f(x) \geq b$
  - only linear equality constraints are allowed
- Hard to verify in general
  - Harder than actually optimizing
  - Your responsibility to give Gurobi convex problems
  - Gurobi ErrorCodes if it discovers non-convexity
    - QCP_EQUALITY_CONSTRAINT
    - Q_NOT_PSD

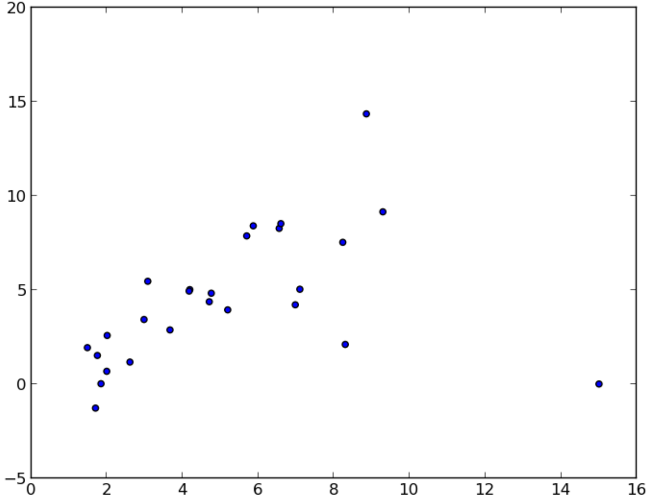
# Building a Quadratic Model with Gurobi

- GRBQuadExpr
  - GRBQuadExpr.addTerm(coef, var1, var2)
  - GRBQuadExpr.addTerm(coef, var)

# Example: Linear Regression

- Input: (x, y) pairs
- Regression Model: $y_i = \text{slope} \cdot x_i + \text{intercept} + \text{residual}_i$
- Minimize: $\sum_i \text{residual}_i^2$

# Example: Sample Data



# Exercise

- Solve regression as a QP
- Compare results with Minimize: $|\sum_i \text{residual}_i|$