Gurobi Modeling Cheat Sheet

GUROBI OPTIMIZATION

Variable Type	Description	Typical Use Case
Continuous	Fractional values are allowed.	Production quantities, investment amounts, flow rates, proportions, etc.
Binary	Restricted to values of 0 or 1.	Ideal for modeling yes/no decisions, such as facility location, on/off switches, and logical constraints.
Integer	Requires Discrete quantities.	Number of machines, people, or units produced.
Semi-continuous	Must be either 0 or within a specified continuous range.	Production decisions where a machine must either be off or operate above a certain minimum level.

All variables are binary

Logical Proposition	Required Constraints	Convenience Feature in Gurobi
At most one of $z_1, z_2, \dots, z_n$ holds	$\sum_i z_i \leq 1$	model.addSOS(type=GRB.SOS_TYPE1)
Exactly one of $z_1, z_2, \dots, z_n$ holds	$\sum_i z_i = 1$	
At least one of $z_1, z_2, \dots, z_n$ holds	$\sum_i z_i \geq 1$	
At most k of $z_1, z_2, \dots, z_n$ holds	$\sum_i z_i \leq k$	
At least k of $z_1, z_2, \dots, z_n$ holds	$\sum_i z_i \geq k$	
$z = \text{NOT } x$	$z = 1 - x$	
$z = x \text{ OR } y$	$x \leq z; y \leq z; z \leq x + y$	model.addGenConstrOr()
$z = x \text{ AND } y$	$x \geq z; y \geq z; z \geq x + y - 1$	model.addGenConstrAnd()
If $x = 1$, then $z = 1$	$x \leq z$	

z is binary; x, y can be continuous

Implication	Required Constraints	Convenience Feature in Gurobi
If $x > 0$, then $z = 1$	$x \leq Mz$	
$x = 0$ or in $[a, b]$	$az \leq x \leq bz$	Use variable type vtype='S' instead
If $z = 1$, then $a^T x \leq b$; $a^T x - b \leq M$	$a^T x \leq b + M(1 - z)$	model.addGenConstrIndicator()
$y = \lvert x \rvert$	$x = x^+ - x^-; y = x^+ + x^-;$ $x^+ \leq Mz; x^- \leq M(1 - z)$	model.addGenConstrAbs()
$y = \max_i\{x_i\}$	$\sum_i z_i = 1; x_i \leq y \leq x_i + M(1 - z_i)\ \forall i$	model.addGenConstrMax()
$y = \min_i\{x_i\}$	$\sum_i z_i = 1; x_i - M(1 - z_i) \leq y \leq x_i\ \forall i$	model.addGenConstrMin()

Solution Pool

To find the top-n solutions, run model.setParam("PoolSolutions", n) and model.setParam("PoolSearchMode", 2), where larger values will impact performance. After optimization has completed, you can retrieve solutions from the pool using a few parameters and attributes:

The model attribute **SolCount** indicates how many solutions were obtained. Solutions in the pool are ordered from best to worst. The best solution can always be obtained through the variable attribute **X**. Suboptimal solutions can be obtained by first setting the parameter **SolutionNumber** and then querying the variable attribute **Xn** or the model attribute **PoolObjVal**.

Numerical Issues and Infeasibility

Look for suspicious coefficients in the MPS file like long sequences of "0000" or "9999". Consider hints in the log file: a) coefficient statistics that indicate any range wider than 1e+08 or magnitudes below tolerances (1e-06), or b) warning messages about large ranges; switching to quad precision; tightening Markowitz tolerance; postponed nodes in the B&B tree of a MIP; max constraint violations; variables dropped from basis. Experiment with parameters that are known to alleviate the effects of numerical issues, such as **NumericFocus**, **IntegralityFocus**, **ObjScale**, **ScaleFlag**, **Presolve**, **Aggregate**, **AggFill**, **Cuts**, **BarHomogeneous**.

If a model is declared as infeasible, run the method model.computeIIS(). If it's taking too long to finish, try the alternative heuristic approach by running the destructive method model.feasRelaxS() and then call optimize on the resulting model.

Multi-Objective Optimization

Ways to manage the trade-offs between objectives: 1) hierarchical (a.k.a. lexicographic) approach using priorities; 2) Blended approach using a weighing scheme; 3) all of the above, where one optimization phase is performed in order of decreasing priority and all objectives with the same priority are blended together according to their weights. By default, Gurobi only considers solutions that would not degrade the objective values of higher-priority objectives, but you can allow for performance degradation by setting relative and/or absolute tolerances. All of this is done by using the method model.setObjectiveN() and its arguments *Priority, Weight, Abstol, and Reltol*. You can set a multi-objective environment for each priority to control the behavior of the solver in the corresponding optimization phase by passing its index (0: highest priority) to the method model.getMultiObjEnv().