Gurobi Implementation
2015 summer
Outlines

- Introduction
- Installing Gurobi
- Creating a visual studio C++ project for Gurobi
- Building and solving Gurobi model
Gurobi is a state-of-the-art solver engine for optimization problems, including

- Linear Problem (LP)
- Mixed-Integer Linear Programming (MILP)
- Quadratic Problem (QP) and Mixed-Integer Quadratic Problem (MIQP) (Gurobi 4.0 and later version)
- Quadratically constrained programming (QCP) and Mixed-integer quadratically constrained programming (MIQCP) (Gurobi 5.0 and later version)

Gurobi supports parallel computing for the modern multicores PCs. It also offers Groubi Cloud on the Amazon Elastic Computing Cloud (EC2).
Gurobi provides different interfaces for different users:

- Gurobi Command Line
- Gurobi Interactive Shell
- programming language:
  - C, C++, C#, Java, Python, VB, MATLAB or R.
- Different modeling systems: AMPL, GAMS, AIMMS, Microsoft Solver Foundation, and etc.
Gurobi supports most platforms including Windows, Linux, and Mac OS X.

The platforms for Gurobi Optimizer 6.0 include:

<table>
<thead>
<tr>
<th>Platform</th>
<th>Operating System</th>
<th>Compiler</th>
</tr>
</thead>
<tbody>
<tr>
<td>Windows 64-bit (win64)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Linux 64-bit (linux64)</td>
<td>Red Hat, SUSE, Ubuntu</td>
<td>GCC 4.1, 4.3, 4.4, 4.6</td>
</tr>
<tr>
<td>Mac OS 64-bit (mac64)</td>
<td>Mac OS X 10.7 - 10.10</td>
<td>Xcode 4, 5, 6</td>
</tr>
</tbody>
</table>
Groubi offers flexible software licensing strategies in different license types, including:

- Free evaluation license (all the features and power).
- Commercial licenses: named-user, single-use, unlimited-use, compute server, on demand cloud, and prepaid cloud licenses.
- Free academic license.

For free academic license, you need to update your license every six months, and one license only for one PC.
Installation
Installation (1/8)

- Go to Gurobi’s website: http://www.gurobi.com/
- Register an Account
Fill out the form and submit, then you can activate your account and get the password from e-mail.
Login your account, and go to Download Center to download the newest version of Gurobi Optimizer according to your PC’s platform.
Installation (4/8)

- Install the Gurobi application.
- Restart the computer.
Login you Gurobi’s account, and request for a Free Academic License.

Free Academic License
Request a free academic license

To request a free academic license, please read and accept the End User License Agreement.

End User License Agreement (View in PDF)
I accept the End User License Agreement

Conditions for the use of an Academic License: An academic license may only be used by a faculty member, a student, or a member of the research or administrative staffs of a degree-granting academic institution. The code may be used only for research and educational purposes. Access for commercial purposes is forbidden.
I accept these conditions

We urge academic users to upgrade to the latest version of Gurobi Optimizer. Some features, such as `grbgetkey()`, may not work correctly in older releases.

Request License
Login your Gurobi’s account, and request for a Free Academic License.

License Detail
License ID 98313

Information and installation instructions

To install this license on a computer where Gurobi Optimizer is installed only or a command/terminal prompt (any system):

```
grbgetkey cfebab50-401a-0000-0000
```

The `grbgetkey` command requires an active internet connection. If you get no response or an error message such as "Unable to contact key server", please click here for additional instructions.

Copy the key.
安装 (7/8)

确保将 grbgetkey 和 lincense 复制并粘贴到开始菜单的搜索框（仅限 Windows）。

使用 `alt + Space, e, p` 来粘贴密钥。

按 Enter。密钥文件 (.lic) 将存储在预定义的文件夹中。
Copy and paste `grbgetkey` and `lincense` and to the Start/Run menu (Windows only).

Don’t move the file.
When a C++ program uses the Gurobi solver, but an error with code number 10009 is shown. Then, (1) check the expired day of the license, or (2) check the location of Gurobi.
The default location for Gurobi is in C drive (C:\gurobi\xxx). If you install Gurobi to other drives, the environment variables of Windows needs to be modified.

1) Right click on the Computer icon and choose Properties option.
2) Click on Advanced system settings in the left pane
3) Select Advanced tab and click on Environment Variables
4) Add a new User variable click on New button.
5) Give Variable name = GRB_LICENSE_FILE and Variable value = D:\gurobi604\gurobi.lic (location of the file)

6) Restart your Visual Studio!
Creating Visual C++ Project
Creating Visual C++ Project (1/9)

1. File → New → Project
2. Visual C++ → Win32
3. Win32 Console Application
4. Give a name, choose a location, then click OK
Creating Visual C++ Project (2/9)

Win32 Application Wizard - CR8_example

Overview

1. Application Settings

Application type:
- Windows application
- Console application
- DLL
- Static library

Additional options:
- Empty project
- Export symbols
- Precompiled header
- Security Development Lifecycle (SDL) checks

Add common header files for:
- ATL
- MFC

2. Finish

3. Cancel
If the CPP file exists, click «Add» → «Existing Item»

Otherwise, click «Add» → «New Item»
Choose『C++ File (.cpp)』, fill in『Name』 and 『Location』. Then, we can find a blank cpp file shown in 『Solution Explorer』.
Creating Visual C++ Project (5/9)

![Image of Visual Studio project settings]

- **Additional Include Directories**: C:\gurobi6.04\win64\include

---

2015/8/12 page: 26
Depending on the version of your Gurobi and Visual Studio. For example, if you use 2013, then the first line should be `C:\gurobi604\win64\lib\gurobi_c++mtdd2013.lib`
Creating Visual C++ Project (7/9)

In the image, the properties page of a Visual C++ project is displayed. The focus is on the 'Runtime Library' setting, which is set to 'Multi-threaded (Debug) (MTD32)'. This is important for enabling multi-threaded debugging in your project.

Enable String Pooling: Yes
Enable Minimal Rebuild: Yes
Enable C++ Exceptions: Yes
Enable Language: C++
Enable Minimal Rebuild: Yes
Enable C++ Exceptions: Yes
Small Type Check: No
Basic Runtime Checks: Both (RTC1, equiv. to RTC2a) (RTC1)
Runtime Library: Multi-threaded (Debug) (MTD32)

Other settings include:
- Struct Member Alignment: Default
- Security Check: Enable Security Check (YES)
- Enable Function-Level Linking: Not Set
- Enable Parallel Code Generation: Not Set
- Enable Enhanced Instruction Set: Not Set
- Floating Point Model: Precise (fp:precise)
- Enable Floating Point Exceptions: Not Set
- Create Hotpatchable Image: Not set

Enable String Pooling enables the compiler to create a single read-only copy of identical strings in the program image and in memory during execution, resulting in smaller programs and an optimization called string pool optimization.
For 64-bits Gurobi libraries, Active solution platform has to be modified for 64-bits environment.

Press the Configuration Manager... button. Under Active solution platform, select New.
Set the new platform to x64, and press OK.
Linear Programming – Example 1
YZ Co. produces clay bowls and pots with aboriginal designs and colors. The two primary resources used by the company are skilled and special pottery clay. The two products have the following resource requirements for production and profit per item:

<table>
<thead>
<tr>
<th>Product</th>
<th>Labor (hr/unit)</th>
<th>Clay (lb/unit)</th>
<th>Profit ($/unit)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Bowl</td>
<td>1</td>
<td>4</td>
<td>40</td>
</tr>
<tr>
<td>Pot</td>
<td>2</td>
<td>3</td>
<td>50</td>
</tr>
<tr>
<td>Available</td>
<td>40</td>
<td>120</td>
<td></td>
</tr>
</tbody>
</table>
Decision variables:

- \( X_1 \) : number of bowls to produce
- \( X_2 \) : number of pots to produce

Objective function:

Max \( Z = 40X_1 + 50X_2 \)

Constraints:

\[
\begin{align*}
X_1 + 2X_2 & \leq 40 \\
4X_1 + 3X_2 & \leq 120 \\
X_1 & \geq 0 \\
X_2 & \geq 0
\end{align*}
\]

<table>
<thead>
<tr>
<th>Product</th>
<th>Labor (hr/unit)</th>
<th>Clay (lb/unit)</th>
<th>Profit ($/unit)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Bowl</td>
<td>1</td>
<td>4</td>
<td>40</td>
</tr>
<tr>
<td>Pot</td>
<td>2</td>
<td>3</td>
<td>50</td>
</tr>
<tr>
<td>Available</td>
<td>40</td>
<td>120</td>
<td></td>
</tr>
</tbody>
</table>
7 elements for Gurobi model

1. Basic Elements
2. Decision Variables
3. Lazy Update
4. Constraint
5. Objective Function
6. Optimization
7. Output Results
Linear Programming – model

Decision variables:

$X_1$: number of bowls to produce

$X_2$: number of pots to produce

Objective function:

Max $Z = 40X_1 + 50X_2$

Constraints:

$1X_1 + 2X_2 \leq 40$

$4X_1 + 3X_2 \leq 120$

$X_1 \geq 0$

$X_2 \geq 0$

<table>
<thead>
<tr>
<th>Product</th>
<th>Labor (hr/unit)</th>
<th>Clay (lb/unit)</th>
<th>Profit ($/unit)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Bowl</td>
<td>1</td>
<td>4</td>
<td>40</td>
</tr>
<tr>
<td>Pot</td>
<td>2</td>
<td>3</td>
<td>50</td>
</tr>
<tr>
<td>Available</td>
<td>40</td>
<td>120</td>
<td></td>
</tr>
</tbody>
</table>
1. Basic Elements – Gurobi Objects (1/2)

1. **Environment Object** - `GRBEnv()`
   ```
   GRBEnv EnvName = GRBEnv();
   ```

2. **Model Object** - `GRBModel (const GRBEnv& env)`
   ```
   GRBModel ModelName = GRBModel (EnvName);
   ```

   The words in blue is the Gurobi’s identifier

   The words in orange is the variable identifier that can be named by yourself

   //1.1 Basic elements declaration
   ```
   GRBEnv env = GRBEnv();
   GRBModel model = GRBModel(env);
   ```
1. Basic Elements – Parameters (2/2)

Give known parameters and coefficients by declaring a matrix or reading data from a file.

```cpp
// 1.2 Parameters definition
const int N = 2;  // number of resources
const int M = 2;  // number of products (D.V.)
int a[N][M] = {{1, 2},
               {4, 3}};  // coefficients in the constraints
int b[N] = {40, 120};  // coefficients of the RHS (Right-Hand-Side)
int c[M] = {40, 50};   // coefficients of objective function
```
2. Decision Variable Declarations (1/2)

```c
GRBVar VarName = ModelName.addVar(lb, ub, obj, type);
```

- **lb**: Lower bound for the variable
- **ub**: Upper bound for the variable (If the upper bound is unlimited, then `ub` is given to `GRB_INFINITY`)
- **obj**: Objective coefficient for the variable.
- **type**: `GRB_INTEGER` – Integer variable
  - `GRB_BINARY` – Binary variable (0 or 1)
  - `GRB_CONTINUOUS` – Continuous variable
  - `GRB_SEMICONT` – Semi-continuous variable (Ex: \(x = 0\) or \(2 \leq x \leq 4\))
  - `GRB_SEMIINT` – Semi-integer variable
2. Decision Variable Declarations (2/2)

```
// 2. Decision Variables
GRBVar x[M];
for(int j=0; j<M; j++) {
    x[j] = model.addVar(0.0, GRB_INFINITY, 0.0, GRB_CONTINUOUS);
}
```

The objective coefficients can be set to arbitrary value, and the true values are given later.
3. Lazy Update

Groubi update model in batch mode, so model must be updated after adding variables into the model

```java
ModelName.update();
```

```java
//3. Integrate variables into model
model.update();
```
4. Constraint Declaration (1/2)

**GRBLinExpr** Linexpr = 0;

**ModelName.addConstr**(lhsExpr, sense, rhsExpr);

*lhsExpr* : Left-hand side (LHS) expression for new linear constraint.

*sense* : **GRB_LESS_EQUAL** – LHS is less than and equal to RHS (<=).

**GRB_EQUAL** – LHS is equal to RHS (==).

**GRB_GREATER_EQUAL** – LHS is greater than and equal to RHS (>=).

*rhsExpr* : Right-hand side (RHS) expression for new linear constraint.

**ModelName.addConstr**(GRBTempConstr& tc);
4. Constraint Declaration

Original:

\[ \begin{align*} 
1X_1 & + 2X_2 \leq 40 \\
4X_1 & + 3X_2 \leq 120 
\end{align*} \]

General Form:

\[ \sum_{j=1}^{M} a_{ij}x_j \leq b_i \quad 1 \leq i \leq N \]

//4. Constraint Declaration

```cpp
for(int i=0; i<N; i++) {
    GRBLinExpr LHS=0;
    for(int j=0; j<M; j++) {
        LHS += a[i][j]*x[j];
    }
    model.addConstr(LHS,GRB_LESS_EQUAL,b[i]);
}
```
5. Objective Function

ModelName.set(GRB_IntAttr_ModelSense, sense);
sense = \begin{align*}
1 & : \text{Minimization (default )} \\
-1 & : \text{Maximization}
\end{align*}

ModelName setObjective(GRBLinExpr or GRBQuadExpr);
GRBQuadExpr is the quadratic expression.

//5. set the model to maximization
model.set(GRB_IntAttr_ModelSense, -1);

GRBLinExpr Obj = 0;
for(int j=0; j<M; j++)
    Obj += c[j]*x[j];
model.setObjective(Obj);

Original:
Max Z = 40X_1 + 50X_2

General Form:
Max Z = \sum_{j=1}^{M} c_j x_j
6. Optimization

`ModelName.optimize();`

`// 6. Optimize the model`
`model.optimize();`
7. Check Optimality and Output Results

7.1 Check Optimality

Get optimality status: `int status = ModelName.get(GRB_IntAttr_Status);`

Status types include `GRB_OPTIMAL`, `GRB_INF_OR_UNBD`, `GRB_INFEASIBLE`, `GRB_UNBOUNDED`, etc.

```cpp
// 7.1 Check optimality
int status = model.get(GRB_IntAttr_Status);
if (status == GRB_OPTIMAL) {
    // 7.2 Output the objective value and solutions
} else if (status == GRB_INF_OR_UNBD) {
    cout << "Infeasible or unbounded" << endl;
} else if (status == GRB_INFEASIBLE) {
    cout << "Infeasible" << endl;
} else if (status == GRB_UNBOUNDED) {
    cout << "Unbounded" << endl;
} else {
    cout << "Optimization was stopped with status" << status << endl;
}
```
7. Check Optimality and Output Results

7.2 Output Results

Get objective value:
ModelName.get(GRB_DoubleAttr_ObjVal);

Get solution value:
VarName.get(GRB_DoubleAttr_X);

//7.2 Output the objective value and solutions
double ObjValue = model.get(GRB_DoubleAttr_ObjVal);
cout<<"total cost= "<<ObjValue<<endl;

for(int i=0; i<M; i++) {
    cout<<"x "<<i<<" ="<<x[i].get(GRB_DoubleAttr_X)<<endl;
}
Using exception handling to **show unexpected errors of Gurobi.**

Put the following code into the main function to **wrap previously mentioned steps.**

```cpp
//8 Output the objective value and solutions
int main() {
    try {
        // Step 1 to step 7 ...
    }
    catch(GRBException e) {
        cout << "Error code = " << e.getErrorCode() << endl;
        cout << e.getMessage() << endl;
    }
    catch(...) {
        cout << "Exception during optimization" << endl;
    }
}
```
Comparing Different Forms

**General Form**

```java
//2. Decision Variables
GRBVar x[M];
for(int j=0; j<M; j++)
    x[j] = model.addVar(0.0, GRB_INFINITY, 0.0,
                        GRB_CONTINUOUS);
model.update(); //3. Integrate variables into model

//4. Constraint Declaration
for(int i=0; i<N; i++)
    {
        GRBLinExpr LHS = 0;
        for(int j=0; j<N; j++)
            LHS += a[i][j]*x[j];
        model.addConstr(LHS <= b[i]);
    }
//5. set the model to maximization
model.set(GRB_IntAttr_ModelSense, -1);
GRBLinExpr Obj = 0;
for(int j=0; j<M; j++)
    Obj += c[j]*x[j];
model.setObjective(Obj);
```

**Expression Form**

```java
//2. Decision Variables
GRBVar x1, x2;
    x1 = model.addVar(0.0, GRB_INFINITY, 0.0,
                        GRB_CONTINUOUS);
x2 = model.addVar(0.0, GRB_INFINITY, 0.0,
                        GRB_CONTINUOUS);
model.update(); //3. Integrate variables into model

//4. Constraint Declaration
    model.addConstr(1*x1 + 2*x2 <= 40);
    model.addConstr(4*x1 + 3*x2 <= 120);
//5. set the model to maximization
    model.set(GRB_IntAttr_ModelSense, -1);
    model.setObjective(40*x1 + 50*x2);
```
Integer Programming – Example 2
A post office requires full-time employees to work on a 7 days/week schedule. Every employee has to work on consecutive five days and then takes two-day off. How many employees are required for the job?

<table>
<thead>
<tr>
<th>MON</th>
<th>TUE</th>
<th>WED</th>
<th>THU</th>
<th>FRI</th>
<th>SAT</th>
<th>SUN</th>
</tr>
</thead>
<tbody>
<tr>
<td>4</td>
<td>5</td>
<td>5</td>
<td>10</td>
<td>12</td>
<td>12</td>
<td>7</td>
</tr>
</tbody>
</table>
Integer Programming – model

Decision variables:

\( X_i \): the number of workers start their work on the \( i \)th day of a week, \( i=1,2,\ldots,7 \)

Objective function:

\[
\text{Min } z = X_1 + X_2 + X_3 + X_4 + X_5 + X_6 + X_7
\]

Constraints:

\[
\begin{align*}
X_1 & \quad X_4 + X_5 + X_6 + X_7 \geq 4 \\
X_1 + X_2 & \quad X_5 + X_6 + X_7 \geq 5 \\
X_1 + X_2 + X_3 & \quad X_6 + X_7 \geq 5 \\
X_1 + X_2 + X_3 + X_4 & \quad X_7 \geq 10 \\
X_1 + X_2 + X_3 + X_4 + X_5 & \quad \geq 12 \\
X_2 + X_3 + X_4 + X_5 + X_6 & \quad \geq 12 \\
X_3 + X_4 + X_5 + X_6 + X_7 & \quad \geq 7
\end{align*}
\]
Integer Programming – model

Decision variables:

- $X_i$: the number of workers start their work on the $i$th day of a week, $i=1,2,\ldots,7$

Objective function:

$$\text{Min } z = X_1 + X_2 + X_3 + X_4 + X_5 + X_6 + X_7$$

Constraints:

1. \[ X_1 + X_2 + X_3 + X_4 + X_5 + X_6 + X_7 \geq 4 \]
2. \[ X_1 + X_2 + X_3 + X_4 + X_5 + X_6 + X_7 \geq 5 \]
3. \[ X_1 + X_2 + X_3 + X_4 + X_5 + X_6 + X_7 \geq 5 \]
4. \[ X_1 + X_2 + X_3 + X_4 + X_5 + X_6 + X_7 \geq 10 \]
5. \[ X_1 + X_2 + X_3 + X_4 + X_5 + X_6 + X_7 \geq 12 \]
6. \[ X_2 + X_3 + X_4 + X_5 + X_6 + X_7 \geq 12 \]
7. \[ X_3 + X_4 + X_5 + X_6 + X_7 \geq 7 \]
1. Basic Elements

//1.1 Basic elements declaration
GRBEnv env = GRBEnv();
GRBModel model = GRBModel(env);

//1.2 Parameters definition
const int N = 7; // 7 days per week
int a[N][N] = {{1,0,0,0,1,1,1},
{1,1,0,0,0,1,1},
{1,1,1,0,0,0,1},
{1,1,1,1,0,0,0},
{0,1,1,1,1,0,0},
{0,0,1,1,1,1,0},
{0,0,0,1,1,1,1}}; // coefficients for the constraints
int b[N] = {4,5,5,10,12,12,7}; // coefficients for the RHS

\[
\begin{align*}
X_1 & \leq 4 \\
X_1 + X_2 & \leq 5 \\
X_1 + X_2 + X_3 & \leq 5 \\
X_1 + X_2 + X_3 + X_4 & \leq 10 \\
X_1 + X_2 + X_3 + X_4 + X_5 & \geq 12 \\
X_2 + X_3 + X_4 + X_5 + X_6 & \geq 12 \\
X_3 + X_4 + X_5 + X_6 + X_7 & \geq 7
\end{align*}
\]
2. Decision Variable Declarations

3. Lazy Update

// 2. Decision Variables
GRBVar x[N];
for(int i=0; i<N; i++) {
    x[i] = model.addVar(0.0, GRB_INFINITY, 0.0, GRB_INTEGER);
}

// 3. Integrate variables into model
model.update();
4. Constraint Declaration

General Form:

\[
\sum_{j=1}^{N} a_{ij} x_j \geq d_i \quad \forall i
\]

//4. Constraint Declaration

for(int i=0; i<N; i++) {
    GRBLinExpr LHS=0;
    for(int j=0; j<N; j++) {
        LHS += a[i][j]*x[j];
    }
    model.addConstr(LHS,GRB_GREATER_EQUAL,d[i]);
}
5. Objective Function

6. Optimization

//5. set the model to minimization
model.set(GRB_IntAttr_ModelSense, 1);

GRBLinExpr Obj = 0;
for(int i=0; i<N; i++)
    Obj += x[i];
model.setObjective(Obj);

//6. Optimize the model
model.optimize();
7. Output Results

// 7. Output the objective value and solutions

```cpp
double ObjValue = model.get(GRB_DoubleAttr_ObjVal);
cout<<"total cost= "<<ObjValue<<endl;

for(int i=0; i<N; i++) {
    cout<<"x "<<i<<" ="<<x[i].get(GRB_DoubleAttr_X)<<endl;
}
```
Parameter Setting of Gurobi
Parameters (1/3) - Time limitation

```java
GRBEnv EnvName = ModelName.getEnv();
ModelName.set(GRB_DoubleParam_ TimeLimit, time);
memory: Limit the total time expended (in seconds).

//6. Optimize the model
GRBEnv modelEnv = model.getEnv();
modelEnv.set(GRB_DoubleParam_ TimeLimit, 3600.0);
model.optimize();
```
Parameters (2/3) - Gap

```
GRBEnv EnvName = ModelName.getEnv();
ModelName.set(GRB_DoubleParam_ MIPGap, gap);
gap : The MIP solver will terminate (with an optimal result) when the relative gap between the lower and upper objective bound is less than \textit{MIPGap} times the upper bound
```

```
//6. Optimize the model
GRBEnv modelEnv = model.getEnv();
modelEnv.set(GRB_DoubleParam_ TimeLimit, 3600.0);
model.optimize();
```
GRBEnv EnvName = ModelName.getEnv();
EnvName.set(GRB_DoubleParam_NodefileStart, memory);
EnvName.set(GRB_StringParam_NodefileDir, path);

memory: Controls the point at which MIP tree nodes are written to disk. Whenever node storage exceeds the specified value (in GBytes), nodes are written to disk.

path: Determines the directory into which nodes are written when node memory usage exceeds the specified NodefileStart value.

Note: this is much more efficient than relying on virtual memory !!

//6. Optimize the model
GRBEnv modelEnv = model.getEnv();
modelEnv.set(GRB_DoubleParam_NodefileStart,0.1);
modelEnv.set(GRB_StringParam_NodefileDir,"G://GRBStore");
model.optimize();
Obtain the solving time (in seconds) for most recent optimization.

```cpp
ModelName.get(GRB_DoubleAttr_Runtime);
double Runtime = model.get(GRB_DoubleAttr_Runtime);
```

Obtain the best bound on current solution (lower bound for minimization, upper bound for maximization).

```cpp
ModelName.get(GRB_DoubleAttr_ObjBound);
double bound = model.get(GRB_DoubleAttr_ObjBound);
```

// Output the elapsed time
```cpp
double Runtime = model.get(GRB_DoubleAttr_Runtime);
cout << "elapsed time = " << Runtime << endl;
```

// Output the lower bound (minimization problem)
```cpp
double LBound = model.get(GRB_DoubleAttr_ObjBound);
cout << "lower bound = " << LBound << endl;
```
Attributes for model, variable, constraints, etc:
http://www.gurobi.com/doc/40/refman/node571.html#sec:Attributes

Parameters for solving scheme:
http://www.gurobi.com/doc/40/refman/node572.html#sec:Parameters

Status codes for optimization:
http://www.gurobi.com/doc/40/refman/node576.html#sec:StatusCodes