The solver: building, solving, interpreting

Sapienza Università di Roma

Outline

The general commands

The commands for linear models

Learning objectives (1/2)

At the end of this chapter you will be able to:

- build an LP in `gurobipy` in five steps;
- declare free variables, remembering the default lower bound 0;
- check `Status` before reading `ObjVal`, `X` and `Slack`;

Learning objectives (2/2)

At the end of this chapter you will be able to:

- read `Pi`, `RC`, `SARHS` and `SAObj` as the quantities of the background chapter;
- recognise the typical errors, from an infeasible model to a right-hand side that contains constants.

Prerequisites

- From this module, Chapter 2 «Background: linear programming and sensitivity analysis»: duals, reduced costs, slacks and validity ranges, which the solver returns as attributes

The general commands

Introduction: The general commands

- This chapter is the **toolbox** of the laboratory
- How to build a model with `gurobipy`, how to run it, how to retrieve the solution
- And — the most important part — how to **interpret the output**
- The examples use the same LPs as the background chapter: together they provide the basis for all the models of the following chapters

Installation and licence

```
python3 -m pip install gurobipy
```

- The package includes a **demonstration licence** (up to 2000 variables and 2000 constraints), sufficient for almost all the models in these lecture notes
- Full and free **academic licence**: register at https://portal.gurobi.com with your institutional email address
- Request a *Named-User Academic License* and run the command `grbgetkey` indicated by the portal (on the university network or VPN)

The five steps to build a model

- Let us take up again the 2×2 LP of the background chapter:

$$\begin{aligned} \max \quad & 30x_1 + 50x_2 \\ & x_1 + 3x_2 \leq 90 \qquad \text{(constraint 1)} \\ & 2x_1 + x_2 \leq 80 \qquad \text{(constraint 2)} \\ & x_1, x_2 \geq 0 \end{aligned}$$

- We build it in `gurobipy` in five steps: container, variables, constraints, objective, writing to file

The five steps (1/2)

```
import gurobipy as gp
from gurobipy import GRB

# Step 1: the model container
m = gp.Model("lp_2x2")

# Step 2: the decision variables
#         (default: continuous, >= 0)
x1 = m.addVar(name="x1")
x2 = m.addVar(name="x2")
```

The five steps (2/2)

```
# Step 3: the constraints (keep the object
#         to read the dual later)
v1 = m.addConstr(x1 + 3*x2 <= 90, name="vincolo1")
v2 = m.addConstr(2*x1 + x2 <= 80, name="vincolo2")

# Step 4: the objective function
m.setObjective(30*x1 + 50*x2, GRB.MAXIMIZE)

# Step 5 (debug): write the model
#         in readable format
m.write("lp_2x2.lp")
```

The three golden rules of gurobipy (1/2)

1. **Indexed variables** with `addVars`:

```
x = m.addVars(I, T, name="x")
```

- creates the family `x[i,t]` over the index sets `I` and `T`
- the sums are written `gp.quicksum(...)` or `x.sum(i, "*")`

2. **Free variables** (which may be negative, such as the intercept b of the regression or the threshold η of the CVaR):

```
m.addVar(lb=-GRB.INFINITY)
```

The three golden rules of gurobipy (2/2)

- Forgetting `lb` is the **most frequent mistake** in the laboratory: the default is ≥ 0!

3. **Families of constraints** with a generator:

```
m.addConstrs((x.sum("*", t) <= b[t] for t in T),
             name="cap")
```

Running the model

```
m.Params.OutputFlag = 0   # 0 = silent, 1 = log
m.Params.TimeLimit  = 60  # seconds
m.optimize()
```

- Parameters are set *before* `optimize()`

Reading the log

The solver log, when active, must be read at three points:

```
Optimize a model with 2 rows, 2 columns and 4 nonzeros
Coefficient statistics: Matrix range [1e+00, 3e+00]
Optimal objective  1.900000000e+03
```

- line 1: is the **size** the expected one?
- line 2: are the data **well scaled**?
- line 3: **status** and optimal value

If the coefficient ranges span from 10^{-6} to 10^{9} the model is badly scaled: change the units of measurement (thousands of euros, MWh, ...) before looking for other errors

Retrieving the solution

Always check the status before reading any number:

```
if m.Status == GRB.OPTIMAL:
    print(m.ObjVal, x1.X, x2.X)
elif m.Status == GRB.INFEASIBLE:
    # which constraints contradict each other
    m.computeIIS(); m.write("conflitto.ilp")
elif m.Status == GRB.UNBOUNDED:
    # almost always a constraint or a bound is missing
    pass
```

The main attributes

Attribute	Object	Meaning
`Status`	model	`OPTIMAL`=2, `INFEASIBLE`=3, `UNBOUNDED`=5, `TIME_LIMIT`=9
`ObjVal`	model	optimal value of the objective
`X`	variable	value of the variable at the optimum
`Slack`	constraint	slack: 0 $\Rightarrow$ active constraint

- The attributes specific to LPs (shadow prices, reduced costs, validity ranges) are in the section that follows

Summary

Building:

- Five steps: `Model`, `addVar(s)`, `addConstr(s)`, `setObjective`, `write`
- Beware of the default ≥ 0: free variables need `lb=-GRB.INFINITY`

Solving and reading:

- In the log: size, scaling of the coefficients, status
- Before reading `ObjVal`, `X`, `Slack`: always check `Status`

The commands for linear models

Introduction: The commands for linear models

- For LPs the solver returns much more than the solution: **shadow prices**, **reduced costs** and **validity ranges**
- We see how to read them, with which sign, and a checklist to interpret the output
- We close with two complete readings: the 2×2 LP and the "all the cases" LP

The bridge with the background notation

The solver attributes are exactly the theoretical quantities of the background chapter:

Theory	Gurobi attribute	Object
shadow price $\tilde{\pi}_i$	`Pi`	constraint
reduced cost $\bar{c}_j$	`RC`	variable
slack $\bar{s}_i$	`Slack`	constraint
range $[b_i^{\min}, b_i^{\max}]$	`SARHSLow`–`SARHSUp`	constraint
range $[c_j^{\min}, c_j^{\max}]$	`SAObjLow`–`SAObjUp`	variable

The stored RHS

Observation 1: The RHS is not always the one you wrote

If a constraint contains a constant on the left-hand side, such as

```
quicksum(x[v,t] ...)
   + base[t] <= C
```

Gurobi moves it into the right-hand side: the stored RHS is `C - base[t]`.

- Writing `constr.RHS = nuovaC` in fact imposes `quicksum(x) <= nuovaC`, which is something else
- The correct form is `constr.RHS = nuovaC - base[t]`
- This mistake actually happened while preparing the case study on electric vehicle charging

Interpreting the output: the checklist (1/2)

1. Is the status `OPTIMAL`? If not, stop and diagnose (`computeIIS` for infeasibility, the `.lp` file for unboundedness)
2. Does the optimal value have the expected order of magnitude?
3. Which constraints are active (`Slack` $= 0$)? Are they the expected ones?
4. How large are the shadow prices? Which resource is worth expanding first? (Signs: in a minimization problem, a constraint $\leq$ has `Pi` ≤ 0)

Interpreting the output: the checklist (2/2)

5. Are there variables at zero? Read their reduced cost (`RC`): it says by how much the objective coefficient must improve for them to be worth activating
6. Is the solution stable? Re-run with data perturbed by 5%
7. Translate everything into a three-line managerial recommendation

Example: complete reading of the 2×2 (1/3)

Running the code of the five steps:

```
Status: 2 (OPTIMAL)        ObjVal: 1900.0
x1 = 30.0    RC = 0.0    SAObj = [16.7, 100.0]
x2 = 20.0    RC = 0.0    SAObj = [15.0, 90.0]
vincolo1: Slack=0.0  Pi=14.0   SARHS=[40.0, 240.0]
vincolo2: Slack=0.0  Pi=8.0    SARHS=[30.0, 180.0]
```

Example: complete reading of the 2×2 (2/3)

- Both constraints are **active** (binding)
- One extra unit of b_1 is worth 14, and the valuation remains valid as long as b_1 stays in $[40, 240]$; beyond that, everything must be recomputed
- If only one right-hand side could be increased, better b_1 ($14 > 8$ per unit, at equal cost)

Example: complete reading of the 2×2 (3/3)

- The reduced costs (attribute `x1.RC`) are null because both variables are basic
- Careful: there may also exist a variable *basic at value zero* (degeneracy), so $\texttt{RC} = 0$ alone does not say that the variable is used
- Adding the third variable of the background chapter the solver would give $x_3 = 0$ with $\texttt{RC} = -2$
- Its coefficient must rise by at least 2 (that is, to 22) for it to enter the solution, with range $\texttt{SAObj} = (-\infty, 22]$

In the general case: shadow prices and sign (1/2)

- **Shadow prices** (attribute `Pi`, one per constraint): always the "derivative of the optimal value with respect to the right-hand side"
- The sign is deduced in two moves:
  - increasing b_i **enlarges** the feasible region (constraint $\leq$) or **shrinks** it (constraint $\geq$)
  - a wider region can never worsen the optimum

In the general case: shadow prices and sign (2/2)

- *Maximization* problem: resource constraint $\leq \Rightarrow$ `Pi` ≥ 0 (more resource, more profit); constraint $\geq \Rightarrow$ `Pi` ≤ 0
- *Minimization* problem: demand constraint $\geq \Rightarrow$ `Pi` ≥ 0 (more demand, more cost); capacity constraint $\leq \Rightarrow$ `Pi` ≤ 0 (more capacity, less cost)
- *Equality* constraint $\Rightarrow$ any sign, as prescribed by the rules of duality
- Inactive constraint $\Rightarrow$ `Pi` $= 0$; validity within the range `SARHSLow`–`SARHSUp`

In the general case: reduced costs

Reduced costs (attribute `RC`, one per variable):

- *basic* variable: $\texttt{RC} = 0$ — as a rule it lies strictly between its bounds, but it may also sit on a bound (*degenerate* basis)
- variable at its *lower bound* (typically at zero): $\texttt{RC} \geq 0$ in a minimization (by how much its cost must fall), $\texttt{RC} \leq 0$ in a maximization (by how much its margin must rise); validity in `SAObjLow`–`SAObjUp`
- variable at its *upper bound*: `RC` is the shadow price of the bound — in a minimum-cost flow a saturated arc has $\texttt{RC} < 0$: one extra unit of capacity saves $|\texttt{RC}|$

The complete loop

For *all* the constraints and variables (LP only):

```
for v in m.getConstrs():
    print(v.ConstrName, v.Pi, v.SARHSLow, v.SARHSUp)
for v in m.getVars():
    print(v.VarName, v.X, v.RC, v.SAObjLow, v.SAObjUp)
```

Example: all the cases (1/5)

- We implement the "all the cases" LP of the background chapter: three constraint directions, three variable signs

$$\begin{aligned} \min \quad & 5x_1 + 8x_2 - 9x_3 \\ & x_1 + x_2 \geq 30 \\ & x_1 + x_2 - x_3 = 100 \\ & x_1 - 2x_2 \leq -20 \\ & x_1 \geq 0, \quad x_2 \text{ free}, \quad x_3 \leq 0 \end{aligned}$$

- We read *everything* the solver returns

Example: all the cases (2/5)

```
m = gp.Model("tutti_i_casi")
x1 = m.addVar(name="x1")                          # x1 >= 0
x2 = m.addVar(lb=-GRB.INFINITY, name="x2") # x2 free
x3 = m.addVar(lb=-GRB.INFINITY, ub=0,
              name="x3")                          # x3 <= 0
v1 = m.addConstr(x1 + x2 >= 30,       name="vincolo1")
v2 = m.addConstr(x1 + x2 - x3 == 100,name="vincolo2")
v3 = m.addConstr(x1 - 2*x2 <= -20,    name="vincolo3")
m.setObjective(5*x1 + 8*x2 - 9*x3, GRB.MINIMIZE)
m.optimize()
```

Example: all the cases (3/5)

```
for v in m.getVars():
    print(v.VarName, v.X, v.RC, v.SAObjLow, v.SAObjUp)
for c in m.getConstrs():
    print(c.ConstrName, c.Slack, c.Pi,
          c.SARHSLow, c.SARHSUp)
```

```
Status: 2 (OPTIMAL)     ObjVal: 620.0
x1: X =  60.0  RC =  0.0  SAObj = [-inf, 8.0]
x2: X =  40.0  RC =  0.0  SAObj = [5.0, 17.0]
x3: X =   0.0  RC = -3.0  SAObj = [-inf, -6.0]
vincolo1: Slack=-70.0 Pi= 0.0 SARHS=[-inf, 100.0]
vincolo2: Slack=  0.0 Pi= 6.0 SARHS=[30.0, inf]
vincolo3: Slack=  0.0 Pi=-1.0 SARHS=[-200.0, inf]
```

Example: all the cases (4/5)

Here all the cases of the previous explanation appear:

- the **inactive** constraint with $\texttt{Pi} = 0$ ($\texttt{vincolo1}$, $\texttt{Slack} \neq 0$)
- the **equality** with a positive dual ($+6$)
- the $\leq$ in a **minimization** with a negative dual (-1)

Example: all the cases (5/5)

- two **basic** variables with $\texttt{RC} = 0$
- the variable x_3 stuck at its **upper bound** (0) with $\texttt{RC} = -3$ and profitability threshold $\texttt{SAObjUp} = -6$
- The verification by hand of each of these numbers, with the rules of duality and by perturbation, is in the "all the cases" example of the background chapter

Summary

LP attributes:

- `Pi`, `RC`, `Slack` and the ranges `SARHS`/`SAObj` are the theoretical quantities of the background chapter
- The stored RHS includes the constants moved from the left-hand side

Interpretation:

- The sign of `Pi` follows from the constraint direction and the optimization sense
- `RC` $= 0$ is not enough to say that a variable is used (degeneracy)
- Follow the checklist up to the managerial recommendation

Exercise: shadow price by perturbation

- Implement the 2×2 LP
- Print `Pi`, `Slack`, `SARHSLow/Up` of both constraints
- Verify the shadow price of the first constraint by re-optimizing with $b_1 : 90 \rightarrow 91$

Solution

- The solver prints the values of the complete reading: `Pi` $= 14$ and 8, `Slack` $= 0$, ranges $[40, 240]$ and $[30, 180]$
- With $b_1 = 91$ both constraints remain active:

$$x_1 + 3x_2 = 91, \quad 2x_1 + x_2 = 80 \quad \Rightarrow \quad x_1 = 29.8, \quad x_2 = 20.4$$

- New optimum $30 \cdot 29.8 + 50 \cdot 20.4 = 1914$: the increase is $1914 - 1900 = 14 =$ `Pi` of constraint 1

Exercise: missing constraint and objective sense

- Remove the second constraint and observe the outcome
- Then change the objective to minimization with no other changes: why is the result $x = 0$?

Solution

- Without constraint 2 the problem is still bounded: the vertices are $(0, 0)$, $(90, 0)$, $(0, 30)$ with values 0, 2700, 1500
- Status `OPTIMAL` with $x_1 = 90$, $x_2 = 0$, `ObjVal` $= 2700$
- In minimization the coefficients 30 and 50 are positive and the variables are ≥ 0 (default!): the objective is ≥ 0 and equals 0 at $x = 0$, which is feasible
- Moral: without constraints that "push" (demand to serve) a minimum-cost problem does nothing

Exercise: diagnosing infeasibility

- Deliberately build an infeasible model, with two incompatible constraints:

$$x_1 + x_2 \geq 10, \qquad x_1 + x_2 \leq 8$$

- Use `computeIIS` to identify the conflict

Solution

- `m.Status` is `INFEASIBLE` (3): no number of the solution should be read
- `m.computeIIS()` computes an *irreducible* infeasible subset of constraints: here the two constraints on $x_1 + x_2$
- `m.write("conflitto.ilp")` writes them to file; for each constraint the attribute `IISConstr` is 1 if it belongs to the conflict

Final summary

- **Building**: five steps in `gurobipy`, beware of the default ≥ 0 of the variables
- **Solving**: read the log (size, scaling, status) and always check `Status`
- **Interpreting**: `Pi`, `RC`, `Slack` and ranges, with the right signs and the checklist