Il solver: costruire, risolvere, interpretare

Indice

I comandi generali

I comandi per i modelli lineari

Obiettivi (1/2)

Alla fine di questo capitolo sarai in grado di:

- costruire un LP in `gurobipy` in cinque passi;
- dichiarare le variabili libere, ricordando il lower bound predefinito 0;
- controllare `Status` prima di leggere `ObjVal`, `X` e `Slack`;

Obiettivi (2/2)

Alla fine di questo capitolo sarai in grado di:

- leggere `Pi`, `RC`, `SARHS` e `SAObj` come le grandezze del capitolo dei richiami;
- riconoscere gli errori tipici, dal modello non ammissibile al termine noto che contiene costanti.

Prerequisiti

- Da questo modulo, Capitolo 2 «Richiami: programmazione lineare e analisi di sensitività»: duali, costi ridotti, scarti e intervalli di validità, che il solver restituisce come attributi

I comandi generali

Introduzione: I comandi generali

- Questo capitolo è la **cassetta degli attrezzi** del laboratorio
- Come si costruisce un modello con `gurobipy`, come si fa girare, come si recupera la soluzione
- E — la parte più importante — come si **interpreta l'output**
- Gli esempi usano gli stessi LP del capitolo di richiami: insieme danno le basi per tutti i modelli dei capitoli successivi

Installazione e licenza

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

- Il pacchetto include una **licenza dimostrativa** (fino a 2000 variabili e 2000 vincoli), sufficiente per quasi tutti i modelli della dispensa
- **Licenza accademica** completa e gratuita: registrarsi su `https://portal.gurobi.com` con l'email istituzionale
- Richiedere una *Named-User Academic License* ed eseguire il comando `grbgetkey` indicato dal portale (sotto rete di ateneo o VPN)

I cinque passi per costruire un modello

- Riprendiamo l'LP 2×2 del capitolo di richiami:

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

- Lo costruiamo in `gurobipy` in cinque passi: contenitore, variabili, vincoli, obiettivo, scrittura su file

I cinque passi (1/2)

```
import gurobipy as gp
from gurobipy import GRB

# Passo 1: il contenitore del modello
m = gp.Model("lp_2x2")

# Passo 2: le variabili decisionali
#          (default: continue, >= 0)
x1 = m.addVar(name="x1")
x2 = m.addVar(name="x2")
```

I cinque passi (2/2)

```
# Passo 3: i vincoli (conservare l'oggetto
#           per leggere poi il duale)
v1 = m.addConstr(x1 + 3*x2 <= 90, name="vincolo1")
v2 = m.addConstr(2*x1 + x2 <= 80, name="vincolo2")

# Passo 4: la funzione obiettivo
m.setObjective(30*x1 + 50*x2, GRB.MAXIMIZE)

# Passo 5 (debug): scrivere il modello
#           in formato leggibile
m.write("lp_2x2.lp")
```

Le tre regole d'oro di gurobipy (1/2)

1. **Variabili indicizzate** con `addVars`:

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

   - crea la famiglia `x[i,t]` sugli insiemi di indici `I` e `T`
   - le somme si scrivono `gp.quicksum(...)` o `x.sum(i, "*")`

2. **Variabili libere** (che possono essere negative, come l'intercetta b della regressione o la soglia η del CVaR):

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

Le tre regole d'oro di gurobipy (2/2)

- Dimenticare `lb` è l'**errore più frequente** del laboratorio: il default è ≥ 0!

3. **Famiglie di vincoli** con generatore:

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

Far girare il modello

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

- I parametri si impostano *prima* di `optimize()`

Leggere il log

Il log del solver, quando attivo, va letto in tre punti:

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

- riga 1: le **dimensioni** sono quelle attese?
- riga 2: i dati sono **ben scalati**?
- riga 3: **esito** e valore ottimo

Se i range dei coefficienti spaziano da 10^{-6} a 10^{9} il modello è mal scalato: cambiare unità di misura (migliaia di euro, MWh, . . .) prima di cercare altri errori

Recuperare la soluzione

Controllare sempre lo stato prima di leggere qualunque numero:

```
if m.Status == GRB.OPTIMAL:
    print(m.ObjVal, x1.X, x2.X)
elif m.Status == GRB.INFEASIBLE:
    # quali vincoli si contraddicono
    m.computeIIS(); m.write("conflitto.ilp")
elif m.Status == GRB.UNBOUNDED:
    pass  # quasi sempre manca un vincolo o un bound
```

Gli attributi principali

Attributo	Oggetto	Significato
`Status`	modello	`OPTIMAL`=2, `INFEASIBLE`=3, `UNBOUNDED`=5, `TIME_LIMIT`=9
`ObjVal`	modello	valore ottimo dell'obiettivo
`X`	variabile	valore della variabile all'ottimo
`Slack`	vincolo	scarto: 0 $\Rightarrow$ vincolo attivo

- Gli attributi specifici degli LP (prezzi ombra, costi ridotti, intervalli di validità) sono nella sezione che segue

Riepilogo

Costruire:

- Cinque passi: `Model`, `addVar(s)`, `addConstr(s)`, `setObjective`, `write`
- Attenzione al default ≥ 0: le variabili libere vogliono `lb=-GRB.INFINITY`

Risolvere e leggere:

- Nel log: dimensioni, scalatura dei coefficienti, esito
- Prima di leggere `ObjVal`, `X`, `Slack`: controllare sempre `Status`

I comandi per i modelli lineari

Introduzione: I comandi per i modelli lineari

- Per gli LP il solver restituisce molto più della soluzione: **prezzi ombra**, **costi ridotti** e **intervalli di validità**
- Vediamo come si leggono, con che segno, e una checklist per interpretare l'output
- Chiudiamo con due letture complete: l'LP 2×2 e l'LP "tutti i casi"

Il ponte con la notazione dei richiami

Gli attributi del solver sono esattamente le quantità teoriche dei richiami:

Teoria	Attributo Gurobi	Oggetto
prezzo ombra $\tilde{\pi}_i$	`Pi`	vincolo
costo ridotto $\bar{c}_j$	`RC`	variabile
scarto $\bar{s}_i$	`Slack`	vincolo
intervallo $[b_i^{\min}, b_i^{\max}]$	`SARHSLow`–`SARHSUp`	vincolo
intervallo $[c_j^{\min}, c_j^{\max}]$	`SAObjLow`–`SAObjUp`	variabile

Il RHS memorizzato

Osservazione 1: Il RHS non è sempre quello che avete scritto

Se un vincolo contiene una costante a sinistra, come

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

Gurobi la sposta nel termine noto: il RHS memorizzato è `C - base[t]`.

- Chi scrive `constr.RHS = nuovaC` sta in realtà imponendo `quicksum(x) <= nuovaC`, cioè un'altra cosa
- La forma corretta è `constr.RHS = nuovaC - base[t]`
- Errore capitato davvero preparando il caso della ricarica dei veicoli elettrici

Interpretare l'output: la checklist (1/2)

1. Lo stato è `OPTIMAL`? Se no, fermarsi e diagnosticare (`computeIIS` per l'inammissibilità, file `.lp` per l'illimitatezza)
2. Il valore ottimo ha l'ordine di grandezza atteso?
3. Quali vincoli sono attivi (`Slack` $= 0$)? Sono quelli previsti?
4. Quanto valgono i prezzi ombra? Quale risorsa conviene potenziare per prima? (Segni: in un problema di minimo, un vincolo $\leq$ ha `Pi` ≤ 0)

Interpretare l'output: la checklist (2/2)

5. Ci sono variabili a zero? Leggere il loro costo ridotto (`RC`): dice di quanto deve migliorare il coefficiente in obiettivo perché convenga attivarle
6. La soluzione è stabile? Rieseguire con dati perturbati del 5%
7. Tradurre tutto in una raccomandazione manageriale di tre righe

Esempio: lettura completa del 2×2 (1/3)

Eseguendo il codice dei cinque passi:

```
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]
```

Esempio: lettura completa del 2×2 (2/3)

- Entrambi i vincoli sono **attivi** (saturi)
- Una unità in più di b_1 vale 14, e la valutazione resta valida finché b_1 resta in $[40, 240]$; oltre, va ricalcolato tutto
- Se si potesse aumentare un solo termine noto, meglio b_1 ($14 > 8$ per unità, a parità di costo)

Esempio: lettura completa del 2×2 (3/3)

- I costi ridotti (attributo `x1.RC`) sono nulli perché entrambe le variabili sono in base
- Attenzione: può esistere anche una variabile *in base a valore zero* (degenerazione), quindi $\texttt{RC} = 0$ da solo non dice che la variabile è usata
- Aggiungendo la terza variabile dei richiami il solver darebbe $x_3 = 0$ con $\texttt{RC} = -2$
- Il suo coefficiente deve salire di almeno 2 (cioè a 22) perché entri in soluzione, con range $\texttt{SAObj} = (-\infty, 22]$

Nel caso generale: prezzi ombra e segno (1/2)

- **Prezzi ombra** (attributo `Pi`, uno per vincolo): sempre la "derivata del valore ottimo rispetto al termine noto"
- Il segno si deduce in due mosse:
  - aumentare b_i **allarga** la regione ammissibile (vincolo $\leq$) o la **restringe** (vincolo $\geq$)
  - una regione più ampia non può mai peggiorare l'ottimo

Nel caso generale: prezzi ombra e segno (2/2)

- Problema di *massimo*: vincolo $\leq$ di risorsa $\Rightarrow$ `Pi` ≥ 0 (più risorsa, più profitto); vincolo $\geq \Rightarrow$ `Pi` ≤ 0
- Problema di *minimo*: vincolo $\geq$ di domanda $\Rightarrow$ `Pi` ≥ 0 (più domanda, più costo); vincolo $\leq$ di capacità $\Rightarrow$ `Pi` ≤ 0 (più capacità, meno costo)
- Vincolo di *uguaglianza* $\Rightarrow$ segno qualunque, come da regole della dualità
- Vincolo non attivo $\Rightarrow$ `Pi` $= 0$; validità nel range `SARHSLow`–`SARHSUp`

Nel caso generale: costi ridotti

Costi ridotti (attributo `RC`, uno per variabile):

- variabile *in base*: $\texttt{RC} = 0$ — di norma è strettamente tra i suoi bound, ma può anche trovarsi su un bound (base *degenere*)
- variabile al *bound inferiore* (tipicamente a zero): $\texttt{RC} \geq 0$ in un minimo (di quanto deve scendere il costo), $\texttt{RC} \leq 0$ in un massimo (di quanto deve salire il margine); validità in `SAObjLow`–`SAObjUp`
- variabile al *bound superiore*: `RC` è il prezzo ombra del bound — in un flusso di minimo costo un arco saturo ha $\texttt{RC} < 0$: un'unità di capacità in più fa risparmiare $|\texttt{RC}|$

Il ciclo completo

Per *tutti* i vincoli e le variabili (solo LP):

```
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)
```

Esempio: tutti i casi (1/5)

- Implementiamo l'LP "tutti i casi" dei richiami: tre versi di vincolo, tre segni di variabile

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

- Leggiamo *tutto* quello che il solver restituisce

Esempio: tutti i casi (2/5)

```
m = gp.Model("tutti_i_casi")
x1 = m.addVar(name="x1")                          # x1 >= 0
x2 = m.addVar(lb=-GRB.INFINITY, name="x2") # x2 libera
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()
```

Esempio: tutti i casi (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]
```

Esempio: tutti i casi (4/5)

Qui compaiono tutti i casi della spiegazione precedente:

- il vincolo **non attivo** con $\texttt{Pi} = 0$ ($\texttt{vincolo1}$, $\texttt{Slack} \neq 0$)
- l'**uguaglianza** con duale positivo ($+6$)
- il $\leq$ in un **minimo** con duale negativo (-1)

Esempio: tutti i casi (5/5)

- due variabili **in base** con $\texttt{RC} = 0$
- la variabile x_3 ferma al suo **bound superiore** (0) con $\texttt{RC} = -3$ e soglia di convenienza $\texttt{SAObjUp} = -6$
- La verifica a mano di ognuno di questi numeri, con le regole della dualità e per perturbazione, è nell'esempio "tutti i casi" del capitolo di richiami

Riepilogo

Attributi degli LP:

- `Pi`, `RC`, `Slack` e i range `SARHS`/`SAObj` sono le quantità teoriche dei richiami
- Il RHS memorizzato include le costanti spostate dal lato sinistro

Interpretazione:

- Il segno di `Pi` si deduce da verso del vincolo e senso di ottimizzazione
- `RC` $= 0$ non basta a dire che una variabile è usata (degenerazione)
- Seguire la checklist fino alla raccomandazione manageriale

Esercizio: prezzo ombra per perturbazione

- Implementare l'LP 2×2
- Stampare `Pi`, `Slack`, `SARHSLow/Up` di entrambi i vincoli
- Verificare il prezzo ombra del primo vincolo ri-ottimizzando con $b_1 : 90 \rightarrow 91$

Soluzione

- Il solver stampa i valori della lettura completa: `Pi` $= 14$ e 8, `Slack` $= 0$, range $[40, 240]$ e $[30, 180]$
- Con $b_1 = 91$ entrambi i vincoli restano attivi:

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

- Nuovo ottimo $30 \cdot 29.8 + 50 \cdot 20.4 = 1914$: l'aumento è $1914 - 1900 = 14 =$ `Pi` del vincolo 1

Esercizio: vincolo mancante e senso dell'obiettivo

- Togliere il secondo vincolo e osservare l'esito
- Poi cambiare l'obiettivo in minimizzazione senza altri cambiamenti: perché il risultato è $x = 0$?

Soluzione

- Senza il vincolo 2 il problema resta limitato: i vertici sono $(0, 0)$, $(90, 0)$, $(0, 30)$ con valori 0, 2700, 1500
- Esito `OPTIMAL` con $x_1 = 90$, $x_2 = 0$, `ObjVal` $= 2700$
- In minimizzazione i coefficienti 30 e 50 sono positivi e le variabili sono ≥ 0 (default!): l'obiettivo è ≥ 0 e vale 0 in $x = 0$, che è ammissibile
- Morale: senza vincoli che "spingono" (domanda da servire) un problema di minimo costo non fa nulla

Esercizio: diagnosticare l'inammissibilità

- Costruire apposta un modello inammissibile, con due vincoli incompatibili:

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

- Usare `computeIIS` per individuare il conflitto

Soluzione

- `m.Status` vale `INFEASIBLE` (3): nessun numero della soluzione va letto
- `m.computeIIS()` calcola un sottoinsieme *irriducibile* di vincoli inammissibile: qui i due vincoli su $x_1 + x_2$
- `m.write("conflitto.ilp")` li scrive su file; per ciascun vincolo l'attributo `IISConstr` vale 1 se fa parte del conflitto

Riepilogo finale

- **Costruire**: cinque passi in `gurobipy`, attenzione al default ≥ 0 delle variabili
- **Risolvere**: leggere il log (dimensioni, scalatura, esito) e controllare sempre `Status`
- **Interpretare**: `Pi`, `RC`, `Slack` e range, con i segni giusti e la checklist