Gliederung

Programmierung und Lösung mit Gurobi

Reales Problem — Problemstellung definieren und systematisch darstellen

Formulierung mathematisches Modell — Zielfunktion, Entscheidungsvariablen, Nebenbedingungen

Programmierung und Lösung des Modells — Gurobipy ist ein Python Paket zur Lösung mathematischer Probleme mit Gurobi

Importiere Gurobi	Das Paket Gurobipy muss zunächst importiert werden. `from gurobipy import *`
Deklariere Modell	Dann kann die Deklaration eines leeren Modells erfolgen. `m = Model ()`
Definiere Daten	Übergabe der Daten an das programmierte Modell
Erstelle Variablen	Deklaration der Entscheidungsvariablen `m.addVar()` `m.addVars()`
Erstelle Zielfunktion	Übergabe der Zielfunktion `m.setObjective()`
Erstelle Restriktionen	Übergabe der Nebenbedingungen `m.addConstr()` `m.addConstrs()`
Starte Optimierung	Optimierung des Modells `m.optimize`

Lösung expliziter Modelle

Importiere Gurobi
`from gurobipy import *`

Deklariere Modell
`m = Model()`

Definiere Daten

Erstelle Variablen
`m.addVar()`
`m.addVars()`

Erstelle Zielfunktion
`m.setObjective()`

Erstelle Restriktionen
`m.addConstr()`
`m.addConstrs()`

Starte Optimierung
`m.optimize`

Mathematisches Modell → Programmiertes Modell

$$max \quad x_1 + x_2 + 2x_3$$

$$u.d.N.: \quad x_1 + x_2 + 3x_3 \leq 4$$
$$x_1 + x_2 \geq 1$$

$$x_1, x_2, x_3 \in \{0,1\}$$

1. Schritt: Gurobi initialisieren

```
1  from gurobipy import *
```

2. Schritt: Modell benennen

```
2  m = Model()
```

3. Schritt: Entscheidungsvariablen deklarieren

```
3  x1 = m.addVar(vtype=GRB.BINARY)
4  x2 = m.addVar(vtype=GRB.BINARY)
5  x3 = m.addVar(vtype=GRB.BINARY)
```

Variablentyp muss dann angegeben werden, wenn er von dem Default-Typ (entspricht *float*) abweichen soll.

4. Schritt: Zielfunktion festlegen

```
6  m.setObjective (x1+x2+2*x3, GRB.MAXIMIZE)
```

Vorgabe der Optimierungsrichtung; Minimierungsproblem GRB.MINIMIZE

5. Schritt: Nebenbedingungen festlegen

```
7  m.addConstr(x1+x2+3*x3 <= 4)
8  m.addConstr(x1+x2 >= 1)
```

6. Schritt: Modell optimieren lassen

```
9  m.optimize ()
```

Lösung impliziter Modelle

Importiere Gurobi
`from gurobipy import *`

Deklariere Modell
`m = Model()`

Definiere Daten

Erstelle Variablen
`m.addVar()`
`m.addVars()`

Erstelle Zielfunktion
`m.setObjective()`

Erstelle Restriktionen
`m.addConstr()`
`m.addConstrs()`

Starte Optimierung
`m.optimize`

Mathematisches Modell

$$max \quad \sum_{i \in I} d_i x_i$$

$$u.d.N.: \quad \sum_{i \in I} a_{ij} x_i \quad \leq b_j \quad \forall j \in J$$

$$x_i \quad \geq 0 \quad \forall i \in I$$

Daten

$$d = [5 \quad 4 \quad 3 \quad 4 \quad 6]$$

$$b = [80 \quad 140 \quad 60]$$

$$a = \begin{bmatrix} 1 & 6 & 2 \\ 3 & 2 & 8 \\ 1 & 3 & 2 \\ 2 & 2 & 2 \\ 2 & 1 & 3 \end{bmatrix}$$

Programmiertes Modell

1. Schritt: Gurobi initialisieren und Modell benennen

```
1  from gurobipy import *
2  m = Model()
```

2. Schritt: Aufnehmen aller Mengen und Werte

```
3  d = [5, 4, 3, 4, 6]
4  b = [80, 140, 60]
5  a = [   [1, 6, 2],
6          [3, 2, 8],
7          [1, 3, 2],
8          [2, 2, 2],
9          [2, 1, 3]]
10 I = range (5)
11 J = range (3)
```

3. Schritt: Entscheidungsvariablen deklarieren

```
12 x = m.addVars(I)
```

4. Schritt: Zielfunktion festlegen

```
13 m.setObjective (sum(d[i]*x[i]for i in I), GRB.MAXIMIZE)
```

5. Schritt: Nebenbedingungen festlegen

```
14 m.addConstrs(sum(a[i][j]*x[i]for i in I)<= b[j] for j in J)
```

6. Schritt: Modell optimieren lassen

```
9  m.optimize ()
```

Ergebnisse ausgeben

Mathematisches Modell

$$max \quad \sum_{i \in I} d_i x_i$$

$$u.d.N.: \quad \sum_{i \in I} a_{ij} x_i \quad \leq b_j \quad \forall j \in J$$

$$x_i \quad \geq 0 \quad \forall i \in I$$

Daten

$$d = [5 \quad 4 \quad 3 \quad 4 \quad 6]$$

$$b = [80 \quad 140 \quad 60]$$

$$a = \begin{bmatrix} 1 & 6 & 2 \\ 3 & 2 & 8 \\ 1 & 3 & 2 \\ 2 & 2 & 2 \\ 2 & 1 & 3 \end{bmatrix}$$

7. Schritt: Ergebnis ausgeben lassen

```
24 if m.status == GRB.OPTIMAL:
25     print (m.ObjVal)
26 else:
27     print(m.status)
```

Gibt den optimalen Zielfunktionswert wieder

- Hiermit würde jedoch nur falls vorhanden das Maximum ausgegeben. Um auch die Wert für Entscheidungsvariablen zu erhalten, müssen auch diese aufgerufen werden.

```
29 for i in I:
30     print("Optinmaler Wert für x[%s] ist %f" %(i, x[i].x))
```

Mit „.x“ wird auf den Wert einer Gurobi-Variable zugegriffen

- Abschließend empfiehlt es sich, das enumerierte Modell als Textdatei ausschreiben zu lassen. Damit können Fehler identifiziert werden.

```
32 m.write("Testmodell.lp")
```

```
2 Maximize
3   5 C0 + 4 C1 + 3 C2 + 4 C3 + 6 C4
4 Subject To
5  R0: C0 + 3 C1 + C2 + 2 C3 + 2 C4 <= 80
6  R1: 6 C0 + 2 C1 + 3 C2 + 2 C3 + C4 <= 140
7  R2: 2 C0 + 8 C1 + 2 C2 + 2 C3 + 3 C4 <= 60
8 Bounds
9 End
```